MiniSail

Mark P. Wassell, University of Cambridge

September 29, 2020

Contents

Chapter 1

Introduction

Syntax and Semantics of MiniSail. This is a kernel language for Sail, an instruction set architecture specification language. The idea behind this language is to capture the key and novel features of Sail in terms of their syntax, typing rules and operational semantics and to confirm that they work together by proving progress and preservation lemmas. We use the Nominal2 library to handle binding.

Chapter 2

Prelude

Some useful generic lemmas. Many of these are from Launchbury.Nominal-Utils.

2.1 Lemmas helping with equivariance proofs

lemma *perm-rel-lemma*:
  assumes $\bigwedge \pi\ x\ y.\ r\ (\pi \cdot x)\ (\pi \cdot y) \Longrightarrow r\ x\ y$
  shows $r\ (\pi \cdot x)\ (\pi \cdot y) \longleftrightarrow r\ x\ y$ (**is** $?l \longleftrightarrow ?r$)
by (*metis* (*full-types*) *assms permute-minus-cancel*(*2*))

lemma *perm-rel-lemma2*:
  assumes $\bigwedge \pi\ x\ y.\ r\ x\ y \Longrightarrow r\ (\pi \cdot x)\ (\pi \cdot y)$
  shows $r\ x\ y \longleftrightarrow r\ (\pi \cdot x)\ (\pi \cdot y)$ (**is** $?l \longleftrightarrow ?r$)
by (*metis* (*full-types*) *assms permute-minus-cancel*(*2*))

lemma *fun-eqvtI*:
  assumes *f-eqvt*[*eqvt*]: $(\bigwedge\ p\ x.\ p \cdot (f\ x) = f\ (p \cdot x))$
  shows $p \cdot f = f$ **by** *perm-simp rule*

lemma *eqvt-at-apply*:
  assumes *eqvt-at f x*
  shows $(p \cdot f)\ x = f\ x$
by (*metis* (*hide-lams*, *no-types*) *assms eqvt-at-def permute-fun-def permute-minus-cancel*(*1*))

lemma *eqvt-at-apply′*:
  assumes *eqvt-at f x*
  shows $p \cdot f\ x = f\ (p \cdot x)$
by (*metis* (*hide-lams*, *no-types*) *assms eqvt-at-def*)

lemma *eqvt-at-apply″*:
  assumes *eqvt-at f x*
  shows $(p \cdot f)\ (p \cdot x) = f\ (p \cdot x)$
by (*metis* (*hide-lams*, *no-types*) *assms eqvt-at-def permute-fun-def permute-minus-cancel*(*1*))

lemma *size-list-eqvt*[*eqvt*]: $p \cdot$ *size-list f x* = *size-list* $(p \cdot f)\ (p \cdot x)$
proof (*induction x*)

case (*Cons x xs*)
have $f\ x = p \cdot (f\ x)$ **by** (*simp add*: *permute-pure*)
also have $... = (p \cdot f)\ (p \cdot x)$ **by** *simp*
with *Cons*
show *?case* **by** (*auto simp add*: *permute-pure*)
qed *simp*

2.2 Freshness via equivariance

lemma *eqvt-fresh-cong1*: $(\bigwedge p\ x.\ p \cdot (f\ x) = f\ (p \cdot x)) \Longrightarrow a \sharp x \Longrightarrow a \sharp f\ x$
apply (*rule fresh-fun-eqvt-app*[*of f*])
apply (*rule eqvtI*)
apply (*rule eq-reflection*)
apply (*rule ext*)
apply (*metis permute-fun-def permute-minus-cancel*(*1*))
apply *assumption*
done

lemma *eqvt-fresh-cong2*:
assumes *eqvt*: $(\bigwedge p\ x\ y.\ p \cdot (f\ x\ y) = f\ (p \cdot x)\ (p \cdot y))$
and *fresh1*: $a \sharp x$ **and** *fresh2*: $a \sharp y$
shows $a \sharp f\ x\ y$
proof−
have *eqvt* $(\lambda\ (x,y).\ f\ x\ y)$
using *eqvt*
apply −
apply (*auto simp add*: *eqvt-def*)
apply (*rule ext*)
apply *auto*
by (*metis permute-minus-cancel*(*1*))
moreover
have $a \sharp (x,\ y)$ **using** *fresh1 fresh2* **by** *auto*
ultimately
have $a \sharp (\lambda\ (x,y).\ f\ x\ y)\ (x,\ y)$ **by** (*rule fresh-fun-eqvt-app*)
thus *?thesis* **by** *simp*
qed

lemma *eqvt-fresh-star-cong1*:
assumes *eqvt*: $(\bigwedge p\ x.\ p \cdot (f\ x) = f\ (p \cdot x))$
and *fresh1*: $a \sharp* x$
shows $a \sharp* f\ x$
by (*metis fresh-star-def eqvt-fresh-cong1 assms*)

lemma *eqvt-fresh-star-cong2*:
assumes *eqvt*: $(\bigwedge p\ x\ y.\ p \cdot (f\ x\ y) = f\ (p \cdot x)\ (p \cdot y))$
and *fresh1*: $a \sharp* x$ **and** *fresh2*: $a \sharp* y$
shows $a \sharp* f\ x\ y$
by (*metis fresh-star-def eqvt-fresh-cong2 assms*)

lemma *eqvt-fresh-cong3*:
assumes *eqvt*: $(\bigwedge p\ x\ y\ z.\ p \cdot (f\ x\ y\ z) = f\ (p \cdot x)\ (p \cdot y)\ (p \cdot z))$
and *fresh1*: $a \sharp x$ **and** *fresh2*: $a \sharp y$ **and** *fresh3*: $a \sharp z$

shows $a \sharp f\ x\ y\ z$
proof−
have *eqvt* $(\lambda\ (x,y,z).\ f\ x\ y\ z)$
using *eqvt*
apply −
apply (*auto simp add*: *eqvt-def*)
apply (*rule ext*)
apply *auto*
by (*metis permute-minus-cancel*(*1*))
moreover
have $a \sharp (x,\ y,\ z)$ **using** *fresh1 fresh2 fresh3* **by** *auto*
ultimately
have $a \sharp (\lambda\ (x,y,z).\ f\ x\ y\ z)\ (x,\ y,\ z)$ **by** (*rule fresh-fun-eqvt-app*)
thus *?thesis* **by** *simp*
qed

lemma *eqvt-fresh-star-cong3*:
assumes *eqvt*: $(\bigwedge p\ x\ y\ z.\ p \cdot (f\ x\ y\ z) = f\ (p \cdot x)\ (p \cdot y)\ (p \cdot z))$
and *fresh1*: $a \sharp* x$ **and** *fresh2*: $a \sharp* y$ **and** *fresh3*: $a \sharp* z$
shows $a \sharp* f\ x\ y\ z$
by (*metis fresh-star-def eqvt-fresh-cong3 assms*)

2.3 Additional simplification rules

lemma *not-self-fresh*[*simp*]: $atom\ x \sharp x \longleftrightarrow False$
by (*metis fresh-at-base*(*2*))

lemma *fresh-star-singleton*: $\{\ x\ \} \sharp* e \longleftrightarrow x \sharp e$
by (*simp add*: *fresh-star-def*)

2.4 Additional equivariance lemmas

lemma *eqvt-cases*:
fixes $f\ x\ \pi$
assumes *eqvt*: $\bigwedge x.\ \pi \cdot f\ x = f\ (\pi \cdot x)$
obtains $f\ x\ f\ (\pi \cdot x) \mid \neg f\ x \quad \neg f\ (\pi \cdot x)$
using *assms*[*symmetric*]
by (*cases f x*) *auto*

lemma *range-eqvt*: $\pi \cdot range\ Y = range\ (\pi \cdot Y)$
unfolding *image-eqvt UNIV-eqvt* **..**

lemma *case-option-eqvt*[*eqvt*]:
$\pi \cdot case\text{-}option\ d\ f\ x = case\text{-}option\ (\pi \cdot d)\ (\pi \cdot f)\ (\pi \cdot x)$
by(*cases x*)(*simp-all*)

lemma *supp-option-eqvt*:
$supp\ (case\text{-}option\ d\ f\ x) \subseteq supp\ d \cup supp\ f \cup supp\ x$
apply (*cases x*)
apply (*auto simp add*: *supp-Some*)
apply (*metis* (*mono-tags*) *Un-iff subsetCE supp-fun-app*)

done

lemma *funpow-eqvt*[*simp*,*eqvt*]:
$\pi \cdot ((f :: 'a \Rightarrow 'a{::}pt) \hat{\ }\hat{\ } n) = (\pi \cdot f) \hat{\ }\hat{\ } (\pi \cdot n)$
apply (*induct n*)
apply *simp*
apply (*rule ext*)
apply *simp*
apply *perm-simp*
apply *simp*
done

lemma *delete-eqvt*[*eqvt*]:
$\pi \cdot AList.delete\ x\ \Gamma = AList.delete\ (\pi \cdot x)\ (\pi \cdot \Gamma)$
by (*induct* Γ, *auto*)

lemma *restrict-eqvt*[*eqvt*]:
$\pi \cdot AList.restrict\ S\ \Gamma = AList.restrict\ (\pi \cdot S)\ (\pi \cdot \Gamma)$
unfolding *AList.restrict-eq* **by** *perm-simp rule*

lemma *supp-restrict*:
$supp\ (AList.restrict\ S\ \Gamma) \subseteq supp\ \Gamma$
by (*induction* Γ) (*auto simp add*: *supp-Pair supp-Cons*)

lemma *clearjunk-eqvt*[*eqvt*]:
$\pi \cdot AList.clearjunk\ \Gamma = AList.clearjunk\ (\pi \cdot \Gamma)$
by (*induction* Γ *rule*: *clearjunk.induct*) *auto*

lemma *map-ran-eqvt*[*eqvt*]:
$\pi \cdot map\text{-}ran\ f\ \Gamma = map\text{-}ran\ (\pi \cdot f)\ (\pi \cdot \Gamma)$
by (*induct* Γ, *auto*)

lemma *dom-perm*:
$dom\ (\pi \cdot f) = \pi \cdot (dom\ f)$
unfolding *dom-def* **by** (*perm-simp*) (*simp*)

lemmas *dom-perm-rev*[*simp*,*eqvt*] = *dom-perm*[*symmetric*]

lemma *ran-perm*[*simp*]:
$\pi \cdot (ran\ f) = ran\ (\pi \cdot f)$
unfolding *ran-def* **by** (*perm-simp*) (*simp*)

lemma *map-add-eqvt*[*eqvt*]:
$\pi \cdot (m1 +\!\!+ m2) = (\pi \cdot m1) +\!\!+ (\pi \cdot m2)$
unfolding *map-add-def*
by (*perm-simp*, *rule*)

lemma *map-of-eqvt*[*eqvt*]:
$\pi \cdot map\text{-}of\ l = map\text{-}of\ (\pi \cdot l)$
apply (*induct l*)
apply (*simp add*: *permute-fun-def*)
apply *simp*

apply *perm-simp*
apply *auto*
done

lemma *concat-eqvt*[*eqvt*]: $\pi \cdot$ *concat* l = *concat* $(\pi \cdot l)$
by (*induction* l)(*auto simp add*: *append-eqvt*)

lemma *tranclp-eqvt*[*eqvt*]: $\pi \cdot$ *tranclp* $P\ v_1\ v_2$ = *tranclp* $(\pi \cdot P)\ (\pi \cdot v_1)\ (\pi \cdot v_2)$
unfolding *tranclp-def* **by** *perm-simp rule*

lemma *rtranclp-eqvt*[*eqvt*]: $\pi \cdot$ *rtranclp* $P\ v_1\ v_2$ = *rtranclp* $(\pi \cdot P)\ (\pi \cdot v_1)\ (\pi \cdot v_2)$
unfolding *rtranclp-def* **by** *perm-simp rule*

lemma *Set-filter-eqvt*[*eqvt*]: $\pi \cdot$ *Set.filter* $P\ S$ = *Set.filter* $(\pi \cdot P)\ (\pi \cdot S)$
unfolding *Set.filter-def*
by *perm-simp rule*

lemma *Sigma-eqvt'*[*eqvt*]: $\pi \cdot$ *Sigma* = *Sigma*
apply (*rule ext*)
apply (*rule ext*)
apply (*subst permute-fun-def*)
apply (*subst permute-fun-def*)
unfolding *Sigma-def*
apply *perm-simp*
apply (*simp add*: *permute-self*)
done

lemma *override-on-eqvt*[*eqvt*]:
$\pi \cdot$ (*override-on m1 m2 S*) = *override-on* $(\pi \cdot m1)\ (\pi \cdot m2)\ (\pi \cdot S)$
by (*auto simp add*: *override-on-def*)

lemma *card-eqvt*[*eqvt*]:
$\pi \cdot$ (*card S*) = *card* $(\pi \cdot S)$
by (*cases finite S*, *induct rule*: *finite-induct*) (*auto simp add*: *card-insert-if mem-permute-iff permute-pure*)

lemma *Projl-permute*:
assumes *a*: $\exists\, y.\ f$ = *Inl y*
shows $(p \cdot$ (*Sum-Type.projl f*)) = *Sum-Type.projl* $(p \cdot f)$
using *a* **by** *auto*

lemma *Projr-permute*:
assumes *a*: $\exists\, y.\ f$ = *Inr y*
shows $(p \cdot$ (*Sum-Type.projr f*)) = *Sum-Type.projr* $(p \cdot f)$
using *a* **by** *auto*

2.5 Freshness lemmas

lemma *fresh-list-elem*:
assumes $a \sharp \Gamma$
and $e \in$ *set* Γ

shows $a \sharp e$
using *assms*
by(*induct* Γ)(*auto simp add*: *fresh-Cons*)

lemma *set-not-fresh*:
$x \in set\ L \Longrightarrow \neg(atom\ x \sharp L)$
by (*metis fresh-list-elem not-self-fresh*)

lemma *pure-fresh-star*[*simp*]: $a \sharp* (x :: 'a :: pure)$
by (*simp add*: *fresh-star-def pure-fresh*)

lemma *supp-set-mem*: $x \in set\ L \Longrightarrow supp\ x \subseteq supp\ L$
by (*induct L*) (*auto simp add*: *supp-Cons*)

lemma *set-supp-mono*: $set\ L \subseteq set\ L2 \Longrightarrow supp\ L \subseteq supp\ L2$
by (*induct L*)(*auto simp add*: *supp-Cons supp-Nil dest*:*supp-set-mem*)

lemma *fresh-star-at-base*:
fixes $x :: 'a :: at\text{-}base$
shows $S \sharp* x \longleftrightarrow atom\ x \notin S$
by (*metis fresh-at-base*(*2*) *fresh-star-def*)

2.6 Freshness and support for subsets of variables

lemma *supp-mono*: *finite* $(B::'a::fs\ set) \Longrightarrow A \subseteq B \Longrightarrow supp\ A \subseteq supp\ B$
by (*metis infinite-super subset-Un-eq supp-of-finite-union*)

lemma *fresh-subset*:
finite $B \Longrightarrow x \sharp (B :: 'a::at\text{-}base\ set) \Longrightarrow A \subseteq B \Longrightarrow x \sharp A$
by (*auto dest*:*supp-mono simp add*: *fresh-def*)

lemma *fresh-star-subset*:
finite $B \Longrightarrow x \sharp* (B :: 'a::at\text{-}base\ set) \Longrightarrow A \subseteq B \Longrightarrow x \sharp* A$
by (*metis fresh-star-def fresh-subset*)

lemma *fresh-star-set-subset*:
$x \sharp* (B :: 'a::at\text{-}base\ list) \Longrightarrow set\ A \subseteq set\ B \Longrightarrow x \sharp* A$
by (*metis fresh-star-set fresh-star-subset*[*OF finite-set*])

2.7 The set of free variables of an expression

definition $fv :: 'a::pt \Rightarrow 'b::at\text{-}base\ set$
where $fv\ e = \{v.\ atom\ v \in supp\ e\}$

lemma *fv-eqvt*[*simp*,*eqvt*]: $\pi \cdot (fv\ e) = fv\ (\pi \cdot e)$
unfolding *fv-def* **by** *simp*

lemma *fv-Nil*[*simp*]: $fv\ [] = \{\}$
by (*auto simp add*: *fv-def supp-Nil*)
lemma *fv-Cons*[*simp*]: $fv\ (x\ \#\ xs) = fv\ x \cup fv\ xs$
by (*auto simp add*: *fv-def supp-Cons*)

lemma *fv-Pair*[*simp*]: *fv* (*x*, *y*) = *fv x* ∪ *fv y*
by (*auto simp add*: *fv-def supp-Pair*)
lemma *fv-append*[*simp*]: *fv* (*x* @ *y*) = *fv x* ∪ *fv y*
by (*auto simp add*: *fv-def supp-append*)
lemma *fv-at-base*[*simp*]: *fv a* = {*a*::′*a*::*at-base*}
by (*auto simp add*: *fv-def supp-at-base*)
lemma *fv-pure*[*simp*]: *fv* (*a*::′*a*::*pure*) = {}
by (*auto simp add*: *fv-def pure-supp*)

lemma *fv-set-at-base*[*simp*]: *fv* (*l* :: (′*a* :: *at-base*) *list*) = *set l*
by (*induction l*) *auto*

lemma *flip-not-fv*: *a* ∉ *fv x* ⟹ *b* ∉ *fv x* ⟹ (*a* ↔ *b*) ∙ *x* = *x*
by (*metis flip-def fresh-def fv-def mem-Collect-eq swap-fresh-fresh*)

lemma *fv-not-fresh*: *atom x* ♯ *e* ⟷ *x* ∉ *fv e*
unfolding *fv-def fresh-def* **by** *blast*

lemma *fresh-fv*: *finite* (*fv e* :: ′*a set*) ⟹ *atom* (*x* :: (′*a*::*at-base*)) ♯ (*fv e* :: ′*a set*) ⟷ *atom x* ♯ *e*
unfolding *fv-def fresh-def*
by (*auto simp add*: *supp-finite-set-at-base*)

lemma *finite-fv*[*simp*]: *finite* (*fv* (*e*::′*a*::*fs*) :: (′*b*::*at-base*) *set*)
proof−
have *finite* (*supp e*) **by** (*metis finite-supp*)
hence *finite* (*atom* −‘ *supp e* :: ′*b set*)
apply (*rule finite-vimageI*)
apply (*rule inj-onI*)
apply (*simp*)
done
moreover
have (*atom* −‘ *supp e* :: ′*b set*) = *fv e* **unfolding** *fv-def* **by** *auto*
ultimately
show *?thesis* **by** *simp*
qed

definition *fv-list* :: ′*a*::*fs* ⇒ ′*b*::*at-base list*
where *fv-list e* = (*SOME l*. *set l* = *fv e*)

lemma *set-fv-list*[*simp*]: *set* (*fv-list e*) = (*fv e* :: (′*b*::*at-base*) *set*)
proof−
have *finite* (*fv e* :: ′*b set*) **by** (*rule finite-fv*)
from *finite-list*[*OF finite-fv*]
obtain *l* **where** *set l* = (*fv e* :: ′*b set*)..
thus *?thesis*
unfolding *fv-list-def* **by** (*rule someI*)
qed

lemma *fresh-fv-list*[*simp*]:
a ♯ (*fv-list e* :: ′*b*::*at-base list*) ⟷ *a* ♯ (*fv e* :: ′*b*::*at-base set*)
proof−
have *a* ♯ (*fv-list e* :: ′*b*::*at-base list*) ⟷ *a* ♯ *set* (*fv-list e* :: ′*b*::*at-base list*)

by (*rule fresh-set*[*symmetric*])
also have ... $\longleftrightarrow$ $a \sharp$ (*fv e* :: $'b$::*at-base set*) **by** *simp*
finally show *?thesis*.
qed

2.8 Other useful lemmas

lemma *pure-permute-id*: *permute p* = (λ *x*. (*x*::$'a$::*pure*))
by *rule* (*simp add*: *permute-pure*)

lemma *supp-set-elem-finite*:
assumes *finite S*
and (*m*::$'a$::*fs*) $\in$ *S*
and *y* $\in$ *supp m*
shows *y* $\in$ *supp S*
using *assms supp-of-finite-sets*
by *auto*

lemmas *fresh-star-Cons* = *fresh-star-list*(*2*)

lemma *mem-permute-set*:
shows *x* $\in$ *p* $\cdot$ *S* $\longleftrightarrow$ (− *p* $\cdot$ *x*) $\in$ *S*
by (*metis mem-permute-iff permute-minus-cancel*(*2*))

lemma *flip-set-both-not-in*:
assumes *x* $\notin$ *S* **and** *x'* $\notin$ *S*
shows ((*x'* $\leftrightarrow$ *x*) $\cdot$ *S*) = *S*
unfolding *permute-set-def*
by (*auto*) (*metis assms flip-at-base-simps*(*3*))+

lemma *inj-atom*: *inj atom* **by** (*metis atom-eq-iff injI*)

lemmas *image-Int*[*OF inj-atom*, *simp*]

lemma *eqvt-uncurry*: *eqvt f* $\Longrightarrow$ *eqvt* (*case-prod f*)
unfolding *eqvt-def*
by *perm-simp simp*

lemma *supp-fun-app-eqvt2*:
assumes *a*: *eqvt f*
shows *supp* (*f x y*) $\subseteq$ *supp x* $\cup$ *supp y*
proof−
from *supp-fun-app-eqvt*[*OF eqvt-uncurry* [*OF a*]]
have *supp* (*case-prod f* (*x*,*y*)) $\subseteq$ *supp* (*x*,*y*).
thus *?thesis* **by** (*simp add*: *supp-Pair*)
qed

lemma *supp-fun-app-eqvt3*:
assumes *a*: *eqvt f*
shows *supp* (*f x y z*) $\subseteq$ *supp x* $\cup$ *supp y* $\cup$ *supp z*
proof−
from *supp-fun-app-eqvt2*[*OF eqvt-uncurry* [*OF a*]]

have *supp* (*case-prod f* (*x*,*y*) *z*) ⊆ *supp* (*x*,*y*) ∪ *supp z*.
thus *?thesis* **by** (*simp add*: *supp-Pair*)
qed

lemma *permute-0*[*simp*]: *permute 0* = (λ *x*. *x*)
by *auto*
lemma *permute-comp*[*simp*]: *permute x* ∘ *permute y* = *permute* (*x* + *y*) **by** *auto*

lemma *map-permute*: *map* (*permute p*) = *permute p*
apply *rule*
apply (*induct-tac x*)
apply *auto*
done

lemma *fresh-star-restrictA*[*intro*]: *a* ♯∗ Γ ⟹ *a* ♯∗ *AList.restrict V* Γ
by (*induction* Γ) (*auto simp add*: *fresh-star-Cons*)

lemma *Abs-lst-Nil-eq*[*simp*]: [[]]*lst*. (*x*::′*a*::*fs*) = [*xs*]*lst*. *x* ′ ⟷ (([],*x*) = (*xs*, *x* ′))
apply *rule*
apply (*frule Abs-lst-fcb2*[**where** *f* = λ *x y* - . (*x*,*y*) **and** *as* = [] **and** *bs* = *xs* **and** *c* = ()])
apply (*auto simp add*: *fresh-star-def*)
done

lemma *Abs-lst-Nil-eq2*[*simp*]: [*xs*]*lst*. (*x*::′*a*::*fs*) = [[]]*lst*. *x* ′ ⟷ ((*xs*,*x*) = ([], *x* ′))
by (*subst eq-commute*) *auto*

lemma *prod-cases8* [*cases type*]:
obtains (*fields*) *a b c d e f g h* **where** *y* = (*a*, *b*, *c*, *d*, *e*, *f*, *g*,*h*)
by (*cases y*, *case-tac g*) *blast*

lemma *prod-induct8* [*case-names fields*, *induct type*]:
(⋀*a b c d e f g h*. *P* (*a*, *b*, *c*, *d*, *e*, *f*, *g*, *h*)) ⟹ *P x*
by (*cases x*) *blast*

lemma *prod-cases9* [*cases type*]:
obtains (*fields*) *a b c d e f g h i* **where** *y* = (*a*, *b*, *c*, *d*, *e*, *f*, *g*,*h*,*i*)
by (*cases y*, *case-tac h*) *blast*

lemma *prod-induct9* [*case-names fields*, *induct type*]:
(⋀*a b c d e f g h i*. *P* (*a*, *b*, *c*, *d*, *e*, *f*, *g*, *h*, *i*)) ⟹ *P x*
by (*cases x*) *blast*

named-theorems *nominal-prod-simps*

named-theorems *ms-fresh Facts for helping with freshness proofs*

lemma *fresh-prod2*[*nominal-prod-simps*,*ms-fresh*]: *x* ♯ (*a*,*b*) = (*x* ♯ *a* ∧ *x* ♯ *b*)

using *fresh-def supp-Pair* **by** *fastforce*

lemma *fresh-prod3*[*nominal-prod-simps*,*ms-fresh*]: $x \sharp (a,b,c) = (x \sharp a \wedge x \sharp b \wedge x \sharp c)$
using *fresh-def supp-Pair* **by** *fastforce*

lemma *fresh-prod4*[*nominal-prod-simps*,*ms-fresh*]: $x \sharp (a,b,c,d) = (x \sharp a \wedge x \sharp b \wedge x \sharp c \wedge x \sharp d)$
using *fresh-def supp-Pair* **by** *fastforce*

lemma *fresh-prod5*[*nominal-prod-simps*,*ms-fresh*]: $x \sharp (a,b,c,d,e) = (x \sharp a \wedge x \sharp b \wedge x \sharp c \wedge x \sharp d \wedge x \sharp e)$
using *fresh-def supp-Pair* **by** *fastforce*

lemma *fresh-prod6*[*nominal-prod-simps*,*ms-fresh*]: $x \sharp (a,b,c,d,e,f) = (x \sharp a \wedge x \sharp b \wedge x \sharp c \wedge x \sharp d \wedge x \sharp e \wedge x \sharp f)$
using *fresh-def supp-Pair* **by** *fastforce*

lemma *fresh-prod7*[*nominal-prod-simps*,*ms-fresh*]: $x \sharp (a,b,c,d,e,f,g) = (x \sharp a \wedge x \sharp b \wedge x \sharp c \wedge x \sharp d \wedge x \sharp e \wedge x \sharp f \wedge x \sharp g)$
using *fresh-def supp-Pair* **by** *fastforce*

lemma *fresh-prod8*[*nominal-prod-simps*,*ms-fresh*]: $x \sharp (a,b,c,d,e,f,g,h) = (x \sharp a \wedge x \sharp b \wedge x \sharp c \wedge x \sharp d \wedge x \sharp e \wedge x \sharp f \wedge x \sharp g \wedge x \sharp h)$
using *fresh-def supp-Pair* **by** *fastforce*

lemma *fresh-prod9*[*nominal-prod-simps*,*ms-fresh*]: $x \sharp (a,b,c,d,e,f,g,h,i) = (x \sharp a \wedge x \sharp b \wedge x \sharp c \wedge x \sharp d \wedge x \sharp e \wedge x \sharp f \wedge x \sharp g \wedge x \sharp h \wedge x \sharp i)$
using *fresh-def supp-Pair* **by** *fastforce*

lemma *fresh-prod10*[*nominal-prod-simps*,*ms-fresh*]: $x \sharp (a,b,c,d,e,f,g,h,i,j) = (x \sharp a \wedge x \sharp b \wedge x \sharp c \wedge x \sharp d \wedge x \sharp e \wedge x \sharp f \wedge x \sharp g \wedge x \sharp h \wedge x \sharp i \wedge x \sharp j)$
using *fresh-def supp-Pair* **by** *fastforce*

lemma *fresh-prod12*[*nominal-prod-simps*,*ms-fresh*]: $x \sharp (a,b,c,d,e,f,g,h,i,j,k,l) = (x \sharp a \wedge x \sharp b \wedge x \sharp c \wedge x \sharp d \wedge x \sharp e \wedge x \sharp f \wedge x \sharp g \wedge x \sharp h \wedge x \sharp i \wedge x \sharp j \wedge x \sharp k \wedge x \sharp l)$
using *fresh-def supp-Pair* **by** *fastforce*

lemmas *fresh-prodN = fresh-Pair fresh-prod3 fresh-prod4 fresh-prod5 fresh-prod6 fresh-prod7 fresh-prod8 fresh-prod9 fresh-prod10 fresh-prod12*

lemma *fresh-prod2I*:
fixes x **and** *x1* **and** *x2*
assumes $x \sharp x1$ **and** $x \sharp x2$
shows $x \sharp (x1,x2)$ **using** *fresh-prod2 assms* **by** *auto*

lemma *fresh-prod3I*:
fixes x **and** *x1* **and** *x2* **and** *x3*
assumes $x \sharp x1$ **and** $x \sharp x2$ **and** $x \sharp x3$
shows $x \sharp (x1,x2,x3)$ **using** *fresh-prod3 assms* **by** *auto*

lemma *fresh-prod4I*:

fixes x **and** $x1$ **and** $x2$ **and** $x3$ **and** $x4$
assumes $x \sharp x1$ **and** $x \sharp x2$ **and** $x \sharp x3$ **and** $x \sharp x4$
shows $x \sharp (x1,x2,x3,x4)$ **using** *fresh-prod4 assms* **by** *auto*

lemma *fresh-prod5I*:
fixes x **and** $x1$ **and** $x2$ **and** $x3$ **and** $x4$ **and** $x5$
assumes $x \sharp x1$ **and** $x \sharp x2$ **and** $x \sharp x3$ **and** $x \sharp x4$ **and** $x \sharp x5$
shows $x \sharp (x1,x2,x3,x4,x5)$ **using** *fresh-prod5 assms* **by** *auto*

lemma *flip-collapse*[*simp*]:
fixes $b1::'a::pt$ **and** $bv1::'b::at$ **and** $bv2::'b::at$
assumes $atom\ bv2 \sharp b1$ **and** $atom\ c \sharp (bv1,bv2,b1)$ **and** $bv1 \neq bv2$
shows $(bv2 \leftrightarrow c) \cdot (bv1 \leftrightarrow bv2) \cdot b1 = (bv1 \leftrightarrow c) \cdot b1$
proof −
have $c \neq bv1$ **and** $bv2 \neq bv1$ **using** *assms* **by** *auto+*
hence $(bv2 \leftrightarrow c) + (bv1 \leftrightarrow bv2) + (bv2 \leftrightarrow c) = (bv1 \leftrightarrow c)$ **using** *flip-triple*[*of c bv1 bv2*] *flip-commute*
by *metis*
hence $(bv2 \leftrightarrow c) \cdot (bv1 \leftrightarrow bv2) \cdot (bv2 \leftrightarrow c) \cdot b1 = (bv1 \leftrightarrow c) \cdot b1$ **using** *permute-plus* **by** *metis*
thus *?thesis* **using** *assms flip-fresh-fresh* **by** *force*
qed

lemma *triple-eqvt*[*simp*]:
$p \cdot (x, b,c) = (p \cdot x,\ \ p \cdot b\ ,\ p \cdot c)$
proof −
have $(x,b,c) = (x,(b,c))$ **by** *simp*
thus *?thesis* **using** *Pair-eqvt* **by** *simp*
qed

lemma *lst-fst*:
fixes $x::'a::at$ **and** $t1::'b::fs$ **and** $x'::'a::at$ **and** $t2::'c::fs$
assumes $([[atom\ x]]lst.\ (t1,t2) = [[atom\ x']]lst.\ (t1',t2'))$
shows $([[atom\ x]]lst.\ t1 = [[atom\ x']]lst.\ t1')$
proof −
have $(\forall c.\ atom\ c \sharp (t2,t2') \longrightarrow atom\ c \sharp (x,\ x',\ t1,\ t1') \longrightarrow (x \leftrightarrow c) \cdot t1 = (x' \leftrightarrow c) \cdot t1')$
proof(*rule*,*rule*,*rule*)
fix $c::'a$
assume $atom\ c \sharp\ (t2,t2')$ **and** $atom\ c \sharp (x,\ x',\ t1,\ t1')$
hence $atom\ c \sharp (x,\ x',\ (t1,t2),\ (t1',t2'))$ **using** *fresh-prod2* **by** *simp*
thus $(x \leftrightarrow c) \cdot t1 = (x' \leftrightarrow c) \cdot t1'$ **using** *assms Abs1-eq-iff-all*(*3*) *Pair-eqvt* **by** *simp*
qed
thus *?thesis* **using** *Abs1-eq-iff-all*(*3*)[*of x t1 x' t1' (t2,t2')*] **by** *simp*
qed

lemma *lst-snd*:
fixes $x::'a::at$ **and** $t1::'b::fs$ **and** $x'::'a::at$ **and** $t2::'c::fs$
assumes $([[atom\ x]]lst.\ (t1,t2) = [[atom\ x']]lst.\ (t1',t2'))$
shows $([[atom\ x]]lst.\ t2 = [[atom\ x']]lst.\ t2')$
proof −
have $(\forall c.\ atom\ c \sharp (t1,t1') \longrightarrow atom\ c \sharp (x,\ x',\ t2,\ t2') \longrightarrow (x \leftrightarrow c) \cdot t2 = (x' \leftrightarrow c) \cdot t2')$

proof(*rule*,*rule*,*rule*)
fix *c*::*'a*
assume *atom c* ♯ (*t1*,*t1'*) **and** *atom c* ♯ (*x*, *x'*, *t2*, *t2'*)
hence *atom c* ♯ (*x*, *x'*, (*t1*,*t2*), (*t1'*,*t2'*)) **using** *fresh-prod2* **by** *simp*
thus (*x* ↔ *c*) ∙ *t2* = (*x'* ↔ *c*) ∙ *t2'* **using** *assms Abs1-eq-iff-all*(*3*) *Pair-eqvt* **by** *simp*
qed
thus *?thesis* **using** *Abs1-eq-iff-all*(*3*)[*of x t2 x' t2'* (*t1*,*t1'*)] **by** *simp*
qed

lemma *lst-head-cons-pair*:
fixes *y1*::*'a* ::*at* **and** *y2*::*'a*::*at* **and** *x1*::*'b*::*fs* **and** *x2*::*'b*::*fs* **and** *xs1*::(*'b*::*fs*) *list* **and** *xs2*::(*'b*::*fs*) *list*
assumes [[*atom y1*]]*lst*. (*x1* # *xs1*) = [[*atom y2*]]*lst*. (*x2* # *xs2*)
shows [[*atom y1*]]*lst*. (*x1*,*xs1*) = [[*atom y2*]]*lst*. (*x2*,*xs2*)
proof(*subst Abs1-eq-iff-all*(*3*)[*of y1* (*x1*,*xs1*) *y2* (*x2*,*xs2*)],*rule*,*rule*,*rule*)
fix *c*::*'a*
assume *atom c* ♯ (*x1*#*xs1*,*x2*#*xs2*) **and** *atom c* ♯ (*y1*, *y2*, (*x1*, *xs1*), *x2*, *xs2*)
thus (*y1* ↔ *c*) ∙ (*x1*, *xs1*) = (*y2* ↔ *c*) ∙ (*x2*, *xs2*) **using** *assms Abs1-eq-iff-all*(*3*) **by** *auto*
qed

lemma *lst-head-cons-neq-nil*:
fixes *y1*::*'a* ::*at* **and** *y2*::*'a*::*at* **and** *x1*::*'b*::*fs* **and** *x2*::*'b*::*fs* **and** *xs1*::(*'b*::*fs*) *list* **and** *xs2*::(*'b*::*fs*) *list*
assumes [[*atom y1*]]*lst*. (*x1* # *xs1*) = [[*atom y2*]]*lst*. (*xs2*)
shows *xs2* ≠ []
proof
assume *as*:*xs2* = []
thus *False* **using** *Abs1-eq-iff*(*3*)[*of y1 x1*#*xs1 y2 Nil*] *assms as* **by** *auto*
qed

lemma *lst-head-cons*:
fixes *y1*::*'a* ::*at* **and** *y2*::*'a*::*at* **and** *x1*::*'b*::*fs* **and** *x2*::*'b*::*fs* **and** *xs1*::(*'b*::*fs*) *list* **and** *xs2*::(*'b*::*fs*) *list*
assumes [[*atom y1*]]*lst*. (*x1* # *xs1*) = [[*atom y2*]]*lst*. (*x2* # *xs2*)
shows [[*atom y1*]]*lst*. *x1* = [[*atom y2*]]*lst*. *x2* **and** [[*atom y1*]]*lst*. *xs1* = [[*atom y2*]]*lst*. *xs2*
using *lst-head-cons-pair lst-fst lst-snd assms* **by** *metis+*

lemma *lst-pure*:
fixes *x1*::*'a* ::*at* **and** *t1*::*'b*::*pure* **and** *x2*::*'a* ::*at* **and** *t2*::*'b*::*pure*
assumes [[*atom x1*]]*lst*. *t1* = [[*atom x2*]]*lst*. *t2*
shows *t1*=*t2*
using *assms Abs1-eq-iff-all*(*3*) *pure-fresh flip-fresh-fresh*
by (*metis Abs1-eq*(*3*) *permute-pure*)

lemma *projl-inl-eqvt*:
fixes π :: *perm*
shows π ∙ (*projl* (*Inl x*)) = *projl* (*Inl* (π ∙ *x*))
unfolding *projl-def Inl-eqvt* **by** *simp*

end

sledgehammer-params[*debug*=*true*, *timeout*=*600*, *provers*= *cvc4 spass e vampire z3*,*isar-proofs*=*true*,*smt-proofs*=*fals*

Chapter 3

Syntax

Syntax of MiniSail and contexts

3.1 Program Syntax

3.1.1 Datatypes

type-synonym *num-nat* = *nat*

atom-decl *x*
atom-decl *u*
atom-decl *bv*

type-synonym *f* = *string*
type-synonym *dc* = *string*
type-synonym *tyid* = *string*

Base types

nominal-datatype *b* =
  B-int
| *B-bool*
| *B-id tyid*
| *B-pair b b* ([- , -]^b)
| *B-unit*
| *B-bitvec*
| *B-var bv*
| *B-app tyid b*

nominal-datatype *bit* = *BitOne* | *BitZero*

Literals

nominal-datatype *l* =
  L-num int
| *L-true*
| *L-false*
| *L-unit*
| *L-bitvec bit list*

Values

nominal-datatype v =
  V-lit l ([-]^{v})
| *V-var x* ([-]^{v})
| *V-pair v v* ([- , -]^{v})
| *V-cons tyid dc v*
| *V-consp tyid dc b v*

Binary Operations

nominal-datatype *opp* = *Plus* (*plus*) | *LEq* (*leq*)

Expressions

nominal-datatype e =
  AE-val v ([-]^{e})
| *AE-app f v* ([- (-)]^{e})
| *AE-appP f b v* ([- [-] (-)]^{e})
| *AE-op opp v v* ([- - -]^{e})
| *AE-concat v v* ([- @@ -]^{e})
| *AE-fst v* ([#1-]^{e})
| *AE-snd v* ([#2-]^{e})
| *AE-mvar u* ([-]^{e})
| *AE-len v* ([| - |]^{e})
| *AE-split v v*

Expressions for Constraints

nominal-datatype *ce* =
  CE-val v ([-]^{ce})
| *CE-op opp ce ce* ([- - -]^{ce})
| *CE-concat ce ce* ([- @@ -]^{ce})
| *CE-fst ce* ([#1-]^{ce})
| *CE-snd ce* ([#2-]^{ce})
| *CE-len ce* ([| - |]^{ce})

Constraints

nominal-datatype c =
  C-true (*TRUE* [] *50*)
| *C-false* (*FALSE* [] *50*)
| *C-conj c c* (- *AND* - [*50*, *50*] *50*)
| *C-disj c c* (- *OR* - [*50*,*50*] *50*)
| *C-not c* (¬ - [] *50*)
| *C-imp c c* (- *IMP* - [*50*, *50*] *50*)
| *C-eq ce ce* (- == - [*50*, *50*] *50*)

Refined type

nominal-datatype τ =
  T-refined-type x::*x b c*::*c* **binds** *x* **in** *c* ({| - : - | - |} [*50*, *50*] *1000*)

value {| *z* : *b-of* τ | ([*v*]^{ce} == [[*L-false*]^{v}]^{ce}) *IMP* (*c-of* τ *z*) |}

Statements

nominal-datatype

```
s =
  AS-val v                                  ( [-]^s)
| AS-let x::x  e s::s binds x in s          ( (LET - = - IN -))
| AS-let2 x::x τ  s s::s binds x in s       ( (LET - : - = - IN -))
| AS-if v s s                               ( (IF - THEN - ELSE -) [0, 61, 0] 61)
| AS-var u::u τ v s::s  binds u in s        ( (VAR - : - = - IN -))
| AS-assign u  v                            ( (- ::= -))
| AS-match v branch-list                    ( (MATCH - WITH { - }))
| AS-while s s                              ( (WHILE - DO { - } ) [0, 0] 61)
| AS-seq s s                                ( ( - ;; - )  [1000, 61] 61)
| AS-assert c s                             ( (ASSERT - IN - ) )
and branch-s =
  AS-branch dc x::x s::s binds x in s  ( ( - - ⇒ - ))
and branch-list =
  AS-final  branch-s                        ( { - } )
| AS-cons  branch-s branch-list             ( ( - | -  ))

term LET x = [plus [x]^v [x]^v]^e IN [[x]^v]^s
```

Function and union type definitions

```
nominal-datatype fun-typ =
    AF-fun-typ x::x b c::c τ::τ s::s binds x in c τ s

nominal-datatype fun-typ-q =
    AF-fun-typ-some bv::bv ft::fun-typ binds bv in ft
  | AF-fun-typ-none fun-typ

nominal-datatype fun-def =
    AF-fundef f fun-typ-q

nominal-datatype type-def =
   AF-typedef string (string * τ) list
   |  AF-typedef-poly string bv::bv dclist::(string * τ) list  binds bv in dclist

lemma check-typedef-poly:
  AF-typedef-poly ''option'' bv [ (''None'', {| zz : B-unit | TRUE |}), (''Some'', {| zz : B-var bv | TRUE
|}) ] =
    AF-typedef-poly ''option'' bv2 [ (''None'', {| zz : B-unit | TRUE |}), (''Some'', {| zz : B-var bv2 |
TRUE |}) ]
  by auto

nominal-datatype var-def =
   AV-def u τ v
```

Programs

```
nominal-datatype p =
  AP-prog type-def list fun-def list var-def list s

declare l.supp [simp] v.supp [simp]  e.supp [simp] s-branch-s-branch-list.supp [simp]  τ.supp [simp]
c.supp [simp] b.supp[simp]
```

3.1.2 Lemmas

Atoms

lemma *x-not-in-u-atoms*[*simp*]:
fixes *u*::*u* **and** *x*::*x* **and** *us*::*u set*
shows *atom x* ∉ *atom*‘*us*
by (*simp add*: *image-iff*)

lemma *x-fresh-u*[*simp*]:
fixes *u*::*u* **and** *x*::*x*
shows *atom x* ♯ *u*
by *auto*

lemma *x-not-in-b-set*[*simp*]:
fixes *x*::*x* **and** *bs*::*bv fset*
shows *atom x* ∉ *supp bs*
by(*induct bs*,*auto*, *simp add*: *supp-finsert supp-at-base*)

lemma *x-fresh-b*[*simp*]:
fixes *x*::*x* **and** *b*::*b*
shows *atom x* ♯ *b*
apply (*induct b rule*: *b.induct*, *auto simp*: *pure-supp*)
using *pure-supp fresh-def* **by** *blast*+

lemma *x-fresh-bv*[*simp*]:
fixes *x*::*x* **and** *bv*::*bv*
shows *atom x* ♯ *bv*
using *fresh-def supp-at-base* **by** *auto*

lemma *u-not-in-x-atoms*[*simp*]:
fixes *u*::*u* **and** *x*::*x* **and** *xs*::*x set*
shows *atom u* ∉ *atom*‘*xs*
by (*simp add*: *image-iff*)

lemma *bv-not-in-x-atoms*[*simp*]:
fixes *bv*::*bv* **and** *x*::*x* **and** *xs*::*x set*
shows *atom bv* ∉ *atom*‘*xs*
by (*simp add*: *image-iff*)

lemma *u-not-in-b-atoms*[*simp*]:
fixes *b* :: *b* **and** *u*::*u*
shows *atom u* ∉ *supp b*
by (*induct b rule*: *b.induct*,*auto simp*: *pure-supp supp-at-base*)

lemma *u-not-in-b-set*[*simp*]:
fixes *u*::*u* **and** *bs*::*bv fset*

shows *atom u* $\notin$ *supp bs*
by(*induct bs*, *auto simp add*: *supp-at-base supp-finsert*)

lemma *u-fresh-b*[*simp*]:
fixes *x*::*u* **and** *b*::*b*
shows *atom x* $\sharp$ *b*
by(*induct b rule*: *b.induct*, *auto simp*: *pure-fresh*)

lemma *supp-b-v-disjoint*:
fixes *x*::*x* **and** *bv*::*bv*
shows *supp* (*V-var x*) $\cap$ *supp* (*B-var bv*) = {}
by (*simp add*: *supp-at-base*)

lemma *supp-b-u-disjoint*[*simp*]:
fixes *b*::*b* **and** *u*::*u*
shows *supp u* $\cap$ *supp b* = {}
by(*nominal-induct b rule*:*b.strong-induct*,(*auto simp add*: *pure-supp b.supp supp-at-base*)+)

lemma *u-fresh-bv*[*simp*]:
fixes *u*::*u* **and** *b*::*bv*
shows *atom u* $\sharp$ *b*
using *fresh-at-base* **by** *simp*

Base Types

nominal-function *b-of* :: $\tau \Rightarrow b$ **where**
b-of $\{\!|$ *z* : *b* | *c* $|\!\}$ = *b*
apply(*auto*,*simp add*: *eqvt-def b-of-graph-aux-def*)
by (*meson* τ.*exhaust*)
nominal-termination (*eqvt*) **by** *lexicographic-order*

lemma *supp-b-empty*[*simp*]:
fixes *b* :: *b* **and** *x*::*x*
shows *atom x* $\notin$ *supp b*
by (*induct b rule*: *b.induct*, *auto simp*: *pure-supp supp-at-base x-not-in-b-set*)

lemma *flip-b-id*[*simp*]:
fixes *x*::*x* **and** *b*::*b*
shows (*x* $\leftrightarrow$ *x'*) $\cdot$ *b* = *b*
by(*rule flip-fresh-fresh*, *auto simp add*: *fresh-def*)

lemma *flip-x-b-cancel*[*simp*]:
fixes *x*::*x* **and** *y*::*x* **and** *b*::*b* **and** *bv*::*bv*
shows (*x* $\leftrightarrow$ *y*) $\cdot$ *b* = *b* **and** (*x* $\leftrightarrow$ *y*) $\cdot$ *bv* = *bv*
using *flip-b-id* **apply** *simp*
by (*metis b.eq-iff*(7) *b.perm-simps*(7) *flip-b-id*)

lemma *flip-bv-x-cancel*[*simp*]:

fixes *bv*::*bv* **and** *z*::*bv* **and** *x*::*x*
shows (*bv* ↔ *z*) ∙ *x* = *x* **using** *flip-fresh-fresh*[*of bv x z*] *fresh-at-base* **by** *auto*

lemma *flip-bv-u-cancel*[*simp*]:
fixes *bv*::*bv* **and** *z*::*bv* **and** *x*::*u*
shows (*bv* ↔ *z*) ∙ *x* = *x* **using** *flip-fresh-fresh*[*of bv x z*] *fresh-at-base* **by** *auto*

Literals

lemma *supp-bitvec-empty*:
fixes *bv*::*bit list*
shows *supp bv* = {}
proof(*induct bv*)
case *Nil*
then show *?case* **using** *supp-Nil* **by** *auto*
next
case (*Cons a bv*)
then show *?case* **using** *supp-Cons bit.supp*
by (*metis* (*mono-tags*, *hide-lams*) *bit.strong-exhaust l.supp*(*5*) *sup-bot.right-neutral*)
qed

lemma *bitvec-pure*[*simp*]:
fixes *bv*::*bit list* **and** *x*::*x*
shows *atom x* ♯ *bv* **using** *fresh-def supp-bitvec-empty* **by** *auto*

lemma *supp-l-empty*[*simp*]:
fixes *l*:: *l*
shows *supp* (*V-lit l*) = {}
apply(*nominal-induct l rule*: *l.strong-induct*)
apply(*auto simp add*: *l.supp l.strong-exhaust pure-supp v.fv-defs*)[*4*]
using *l.supp pure-supp supp-of-atom-list supp-bitvec-empty* **by** *simp*

lemma *type-l-nosupp*[*simp*]:
fixes *x*::*x* **and** *l*::*l*
shows *atom x* ∉ *supp* ({| *z* : *b* | $[[z]^v]^{ce}$ == $[[l]^v]^{ce}$ |})
using *supp-at-base supp-l-empty ce.supp*(*1*) *c.supp* τ.*supp* **by** *force*

lemma *flip-bitvec0*:
fixes *x*::*bit list*
assumes *atom c* ♯ (*z*, *x*, *z*′)
shows (*z* ↔ *c*) ∙ *x* = (*z*′ ↔ *c*) ∙ *x*
proof −
have *atom z* ♯ *x* **and** *atom z*′ ♯ *x*
using *flip-fresh-fresh assms supp-bitvec-empty fresh-def* **by** *blast*+
moreover have *atom c* ♯ *x* **using** *supp-bitvec-empty fresh-def* **by** *auto*
ultimately show *?thesis* **using** *assms flip-fresh-fresh* **by** *metis*
qed

lemma *flip-bitvec*:
assumes *atom c* ♯ (*z*, *L-bitvec x*, *z*′)
shows (*z* ↔ *c*) ∙ *x* = (*z*′ ↔ *c*) ∙ *x*
proof −

have *atom* $z \sharp x$ **and** *atom* $z' \sharp x$
using *flip-fresh-fresh assms supp-bitvec-empty fresh-def* **by** *blast+*
moreover have *atom* $c \sharp x$ **using** *supp-bitvec-empty fresh-def* **by** *auto*
ultimately show *?thesis* **using** *assms flip-fresh-fresh* **by** *metis*
qed

lemma *type-l-eq*:
shows $\{\!|\ z : b\ |\ [[z]^v]^{ce} == [V\text{-}lit\ l]^{ce}\ |\!\} = (\{\!|\ z' : b\ |\ [[z']^v]^{ce} == [V\text{-}lit\ l]^{ce}\ |\!\})$
by(*auto*,*nominal-induct l rule*: *l.strong-induct*,*auto*, *metis permute-pure*, *auto simp add*: *flip-bitvec*)

lemma *flip-l-eq*:
fixes x::l
shows $(z \leftrightarrow c) \cdot x = (z' \leftrightarrow c) \cdot x$
proof −
have *atom* $z \sharp x$ **and** *atom* $c \sharp x$ **and** *atom* $z' \sharp x$
using *flip-fresh-fresh fresh-def supp-l-empty* **by** *fastforce+*
thus *?thesis* **using** *flip-fresh-fresh* **by** *metis*
qed

lemma *flip-l-eq1*:
fixes x::l
assumes $(z \leftrightarrow c) \cdot x = (z' \leftrightarrow c) \cdot x'$
shows $x' = x$
proof −
have *atom* $z \sharp x$ **and** *atom* $c \sharp x'$ **and** *atom* $c \sharp x$ **and** *atom* $z' \sharp x'$
using *flip-fresh-fresh fresh-def supp-l-empty* **by** *fastforce+*
thus *?thesis* **using** *flip-fresh-fresh assms* **by** *metis*
qed

Types

lemma *flip-base-eq*:
fixes b::b **and** x::x **and** y::x
shows $(x \leftrightarrow y) \cdot b = b$
using *b.fresh* **by** (*simp add*: *flip-fresh-fresh fresh-def*)

Obtain an alpha-equivalent type where the bound variable is fresh in some term t

lemma *has-fresh-z0*:
fixes t::$'b$::fs
shows $\exists z.\ atom\ z \sharp (c',t) \wedge (\{\!|z' : b\ |\ c'\ |\!\}) = (\{\!|\ z : b\ |\ (z \leftrightarrow z')\cdot c'\ |\!\})$
proof −
obtain z::x **where** *fr*: *atom* $z \sharp (c',t)$ **using** *obtain-fresh* **by** *blast*
moreover hence $(\{\!|\ z' : b\ |\ c'\ |\!\}) = (\{\!|\ z : b\ |\ (z \leftrightarrow z')\cdot c'\ |\!\})$
using τ*.eq-iff Abs1-eq-iff*
by (*metis flip-commute flip-fresh-fresh fresh-PairD*(*1*))
ultimately show *?thesis* **by** *fastforce*
qed

lemma *has-fresh-z*:
fixes t::$'b$::fs
shows $\exists z\ b\ c.\ atom\ z \sharp t \wedge \tau = \{\!|\ z : b\ |\ c\ |\!\}$
proof −
obtain z' **and** b **and** c' **where** *teq*: $\tau = (\{\!|\ z' : b\ |\ c'\ |\!\})$ **using** τ*.exhaust* **by** *blast*

obtain $z::x$ **where** *fr*: *atom* $z \sharp (t,c')$ **using** *obtain-fresh* **by** *blast*
hence $(\{\!\!\{ z' : b \mid c' \}\!\!\}) = (\{\!\!\{ z : b \mid (z \leftrightarrow z') \cdot c' \}\!\!\})$ **using** τ*.eq-iff Abs1-eq-iff flip-commute flip-fresh-fresh fresh-PairD*(*1*) **by** (*metis fresh-PairD*(*2*))
hence *atom* $z \sharp t \wedge \tau = (\{\!\!\{ z : b \mid (z \leftrightarrow z') \cdot c' \}\!\!\})$ **using** *fr teq* **by** *force*
thus *?thesis* **using** *teq fr* **by** *fast*
qed

lemma *obtain-fresh-z*:
fixes $t::'b::fs$
obtains z **and** b **and** c **where** *atom* $z \sharp t \wedge \tau = \{\!\!\{ z : b \mid c \}\!\!\}$
using *has-fresh-z* **by** *blast*

lemma *has-fresh-z2*:
fixes $t::'b::fs$
shows $\exists z\ c.$ *atom* $z \sharp t \wedge \tau = \{\!\!\{ z : b\text{-}of\ \tau \mid c \}\!\!\}$
proof −
obtain z **and** b **and** c **where** *atom* $z \sharp t \wedge \tau = \{\!\!\{ z : b \mid c \}\!\!\}$ **using** *obtain-fresh-z* **by** *metis*
moreover then have *b-of* $\tau = b$ **using** τ*.eq-iff* **by** *simp*
ultimately show *?thesis* **using** *obtain-fresh-z* τ*.eq-iff* **by** *auto*
qed

lemma *obtain-fresh-z2*:
fixes $t::'b::fs$
obtains z **and** c **where** *atom* $z \sharp t \wedge \tau = \{\!\!\{ z : b\text{-}of\ \tau \mid c \}\!\!\}$
using *has-fresh-z2* **by** *blast*

Value

lemma *u-notin-supp-v*[*simp*]:
fixes $u::u$ **and** $v::v$
shows *atom* $u \notin$ *supp* v
proof(*nominal-induct v rule*: *v.strong-induct*)
case (*V-lit l*)
then show *?case* **using** *supp-l-empty* **by** *auto*
next
case (*V-var x*)
then show *?case*
by (*simp add*: *supp-at-base*)
next
case (*V-pair v1 v2*)
then show *?case* **by** *auto*
next
case (*V-cons tyid list v*)
then show *?case* **using** *pure-supp* **by** *auto*
next
case (*V-consp tyid list b v*)
then show *?case* **using** *pure-supp* **by** *auto*
qed

lemma *u-fresh-xv*[*simp*]:
fixes $u::u$ **and** $x::x$ **and** $v::v$
shows *atom* $u \sharp (x,v)$

proof −
have *atom u* ♯ *x* **using** *fresh-def* **by** *fastforce*
moreover have *atom u* ♯ *v* **using** *fresh-def u-notin-supp-v* **by** *metis*
ultimately show *?thesis* **using** *fresh-prod2* **by** *auto*
qed

Part of effort to make the proofs across cases more uniform by distilling the non-uniform parts into lemmas like this

lemma *v-flip-eq*:
fixes *v*::*v* **and** *va*::*v* **and** *x*::*x* **and** *c*::*x*
assumes *atom c* ♯ (*v*, *va*) **and** *atom c* ♯ (*x*, *xa*, *v*, *va*) **and** (*x* ↔ *c*) · *v* = (*xa* ↔ *c*) · *va*
shows ((*v* = *V-lit l* ⟶ (∃ *l'*. *va* = *V-lit l'* ∧ (*x* ↔ *c*) · *l* = (*xa* ↔ *c*) · *l'*))) ∧
((*v* = *V-var y* ⟶ (∃ *y'*. *va* = *V-var y'* ∧ (*x* ↔ *c*) · *y* = (*xa* ↔ *c*) · *y'*))) ∧
((*v* = *V-pair vone vtwo* ⟶ (∃ *v1' v2'*. *va* = *V-pair v1' v2'* ∧ (*x* ↔ *c*) · *vone* = (*xa* ↔ *c*) · *v1'* ∧ (*x* ↔ *c*) · *vtwo* = (*xa* ↔ *c*) · *v2'*))) ∧
((*v* = *V-cons tyid dc vone* ⟶ (∃ *v1'*. *va* = *V-cons tyid dc v1'* ∧ (*x* ↔ *c*) · *vone* = (*xa* ↔ *c*) · *v1'*))) ∧
((*v* = *V-consp tyid dc b vone* ⟶ (∃ *v1'*. *va* = *V-consp tyid dc b v1'* ∧ (*x* ↔ *c*) · *vone* = (*xa* ↔ *c*) · *v1'*)))
using *assms* **proof**(*nominal-induct v rule*:*v.strong-induct*)
case (*V-lit l*)
then show *?case* **using** *assms v.perm-simps*
empty-iff flip-def fresh-def fresh-permute-iff supp-l-empty swap-fresh-fresh v.fresh
by (*metis permute-swap-cancel2 v.distinct*)
next
case (*V-var x*)
then show *?case* **using** *assms v.perm-simps*
empty-iff flip-def fresh-def fresh-permute-iff supp-l-empty swap-fresh-fresh v.fresh
by (*metis permute-swap-cancel2 v.distinct*)
next
case (*V-pair v1 v2*)
have (*V-pair v1 v2* = *V-pair vone vtwo* ⟶ (∃ *v1' v2'*. *va* = *V-pair v1' v2'* ∧ (*x* ↔ *c*) · *vone* = (*xa* ↔ *c*) · *v1'* ∧ (*x* ↔ *c*) · *vtwo* = (*xa* ↔ *c*) · *v2'*)) **proof**
assume *V-pair v1 v2*= *V-pair vone vtwo*
thus (∃ *v1' v2'*. *va* = *V-pair v1' v2'* ∧ (*x* ↔ *c*) · *vone* = (*xa* ↔ *c*) · *v1'* ∧ (*x* ↔ *c*) · *vtwo* = (*xa* ↔ *c*) · *v2'*)
using *V-pair assms*
by (*metis* (*no-types*, *hide-lams*) *flip-def permute-swap-cancel v.perm-simps*(*3*))
qed
thus *?case* **using** *V-pair* **by** *auto*
next
case (*V-cons tyid dc v1*)
have (*V-cons tyid dc v1* = *V-cons tyid dc vone* ⟶ (∃ *v1'*. *va* = *V-cons tyid dc v1'* ∧ (*x* ↔ *c*) · *vone* = (*xa* ↔ *c*) · *v1'*)) **proof**
assume *as*: *V-cons tyid dc v1* = *V-cons tyid dc vone*
hence (*x* ↔ *c*) · (*V-cons tyid dc vone*) = *V-cons tyid dc* ((*x* ↔ *c*) · *vone*) **proof** −
have (*x* ↔ *c*) · *dc* = *dc* **using** *pure-permute-id* **by** *metis*
moreover have (*x* ↔ *c*) · *tyid* = *tyid* **using** *pure-permute-id* **by** *metis*
ultimately show *?thesis* **using** *v.perm-simps*(*4*) **by** *simp*
qed
then obtain *v1'* **where** (*xa* ↔ *c*) · *va* = *V-cons tyid dc v1'* ∧ (*x* ↔ *c*) · *vone* = *v1'* **using** *assms V-cons*

using *as* **by** *fastforce*
hence *va = V-cons tyid dc ((xa ↔ c) · v1') ∧ (x ↔ c) · vone = v1'* **using** *permute-flip-cancel empty-iff flip-def fresh-def supp-b-empty swap-fresh-fresh*
by (*metis pure-fresh v.perm-simps*(*4*))

thus (∃ *v1'. va = V-cons tyid dc v1' ∧ (x ↔ c) · vone = (xa ↔ c) · v1'*)
using *V-cons assms* **by** *simp*
qed
thus *?case* **using** *V-cons* **by** *auto*
next
case (*V-consp tyid dc b v1*)
have (*V-consp tyid dc b v1 = V-consp tyid dc b vone* ⟶ (∃ *v1'. va = V-consp tyid dc b v1' ∧ (x ↔ c) · vone = (xa ↔ c) · v1'*)) **proof**
assume *as*: *V-consp tyid dc b v1 = V-consp tyid dc b vone*
hence (*x ↔ c*) · (*V-consp tyid dc b vone*) = *V-consp tyid dc b* ((*x ↔ c*) · *vone*) **proof** −
have (*x ↔ c*) · *dc = dc* **using** *pure-permute-id* **by** *metis*
moreover have (*x ↔ c*) · *tyid = tyid* **using** *pure-permute-id* **by** *metis*
ultimately show *?thesis* **using** *v.perm-simps*(*4*) **by** *simp*
qed
then obtain *v1'* **where** (*xa ↔ c*) · *va = V-consp tyid dc b v1' ∧ (x ↔ c) · vone = v1'* **using** *assms V-consp*
using *as* **by** *fastforce*
hence *va = V-consp tyid dc b ((xa ↔ c) · v1') ∧ (x ↔ c) · vone = v1'* **using** *permute-flip-cancel empty-iff flip-def fresh-def supp-b-empty swap-fresh-fresh*
pure-fresh v.perm-simps
by (*metis* (*mono-tags*, *hide-lams*))
thus (∃ *v1'. va = V-consp tyid dc b v1' ∧ (x ↔ c) · vone = (xa ↔ c) · v1'*)
using *V-consp assms* **by** *simp*
qed
thus *?case* **using** *V-consp* **by** *auto*

qed

lemma *flip-eq*:
fixes *x*::*x* **and** *xa*::*x* **and** *s*::*'a*::*fs* **and** *sa*::*'a*::*fs*
assumes (∀ *c. atom c ♯* (*s*, *sa*) ⟶ *atom c ♯* (*x*, *xa*, *s*, *sa*) ⟶ (*x ↔ c*) · *s* = (*xa ↔ c*) · *sa*) **and** *x ≠ xa*
shows (*x ↔ xa*) · *s = sa*
proof −
have ([[*atom x*]]*lst. s* = [[*atom xa*]]*lst. sa*) **using** *assms Abs1-eq-iff-all* **by** *simp*
hence (*xa = x ∧ sa = s ∨ xa ≠ x ∧ sa* = (*xa ↔ x*) · *s ∧ atom xa ♯ s*) **using** *assms Abs1-eq-iff*[*of xa sa x s*] **by** *simp*
thus *?thesis* **using** *assms*
by (*metis flip-commute*)
qed

lemma *swap-v-supp*:
fixes *v*::*v* **and** *d*::*x* **and** *z*::*x*
assumes *atom d ♯ v*
shows *supp* ((*z ↔ d*) · *v*) ⊆ *supp v* − { *atom z* } ∪ { *atom d*}
using *assms*
proof(*nominal-induct v rule*:*v.strong-induct*)

case (*V-lit l*)
then show *?case* **using** *l.supp* **by** (*metis supp-l-empty empty-subsetI l.strong-exhaust pure-supp supp-eqvt v.supp*)
next
case (*V-var x*)
hence $d \neq x$ **using** *fresh-def* **by** *fastforce*
thus *?case* **apply**(*cases z = x*) **using** *supp-at-base V-var* ‹$d \neq x$› **by** *fastforce+*
next
case (*V-cons tyid dc v*)
show *?case* **using** *v.supp*(*4*) *pure-supp*
using *V-cons.hyps V-cons.prems fresh-def* **by** *auto*
next
case (*V-consp tyid dc b v*)
show *?case* **using** *v.supp*(*4*) *pure-supp*
using *V-consp.hyps V-consp.prems fresh-def* **by** *auto*
qed(*force+*)

Expressions

lemma *swap-e-supp*:
fixes *e*::*e* **and** *d*::*x* **and** *z*::*x*
assumes *atom d* ♯ *e*
shows *supp* $((z \leftrightarrow d) \cdot e) \subseteq$ *supp e* $-$ { *atom z* } $\cup$ { *atom d*}
using *assms*
proof(*nominal-induct e rule*:*e.strong-induct*)
case (*AE-val v*)
then show *?case* **using** *swap-v-supp* **by** *simp*
next
case (*AE-app f v*)
then show *?case* **using** *swap-v-supp* **by** (*simp add*: *pure-supp*)
next
case (*AE-appP b f v*)
hence *df*: *atom d* ♯ *v* **using** *fresh-def e.supp* **by** *force*
have *supp* $((z \leftrightarrow d) \cdot$ (*AE-appP b f v*)) = *supp* (*AE-appP b f* $((z \leftrightarrow d) \cdot v)$) **using** *e.supp*
by (*metis b.eq-iff*(*3*) *b.perm-simps*(*3*) *e.perm-simps*(*3*) *flip-b-id*)
also have ... = *supp b* $\cup$ *supp f* $\cup$ *supp* $((z \leftrightarrow d) \cdot v)$ **using** *e.supp* **by** *auto*
also have ... $\subseteq$ *supp b* $\cup$ *supp f* $\cup$ *supp v* $-$ { *atom z* } $\cup$ { *atom d*} **using** *swap-v-supp*[*OF df*] *pure-supp* **by** *auto*
finally show *?case* **using** *e.supp* **by** *auto*
next
case (*AE-op opp v1 v2*)
hence *df*: *atom d* ♯ *v1* $\wedge$ *atom d* ♯ *v2* **using** *fresh-def e.supp* **by** *force*
have $((z \leftrightarrow d) \cdot$ (*AE-op opp v1 v2*)) = *AE-op opp* $((z \leftrightarrow d) \cdot v1)$ $((z \leftrightarrow d) \cdot v2)$ **using**
e.perm-simps flip-commute opp.perm-simps AE-op opp.strong-exhaust pure-supp
by (*metis* (*full-types*))

hence *supp* $((z \leftrightarrow d) \cdot$ *AE-op opp v1 v2*) = *supp* (*AE-op opp* $((z \leftrightarrow d) \cdot v1)$ $((z \leftrightarrow d) \cdot v2)$) **by** *simp*
also have ... = *supp* $((z \leftrightarrow d) \cdot v1) \cup$ *supp* $((z \leftrightarrow d) \cdot v2)$ **using** *e.supp*
by (*metis* (*mono-tags*, *hide-lams*) *opp.strong-exhaust opp.supp sup-bot.left-neutral*)
also have ... $\subseteq$ (*supp v1* $-$ { *atom z* } $\cup$ { *atom d*}) $\cup$ (*supp v2* $-$ { *atom z* } $\cup$ { *atom d*}) **using** *swap-v-supp AE-op df* **by** *blast*
finally show *?case* **using** *e.supp opp.supp* **by** *blast*

next
case (*AE-fst v*)
then show *?case* **using** *swap-v-supp* **by** *auto*
next
case (*AE-snd v*)
then show *?case* **using** *swap-v-supp* **by** *auto*
next
case (*AE-mvar u*)
then show *?case* **using**
Diff-empty Diff-insert0 Un-upper1 atom-x-sort flip-def flip-fresh-fresh fresh-def set-eq-subset supp-eqvt swap-set-in-eq
by (*metis sort-of-atom-eq*)
next
case (*AE-len v*)
then show *?case* **using** *swap-v-supp* **by** *auto*
next
case (*AE-concat v1 v2*)
then show *?case* **using** *swap-v-supp* **by** *auto*
next
case (*AE-split v1 v2*)
then show *?case* **using** *swap-v-supp* **by** *auto*
qed

lemma *swap-ce-supp*:
fixes *e*::*ce* **and** *d*::*x* **and** *z*::*x*
assumes *atom d* ♯ *e*
shows *supp* ((*z* ↔ *d*) ∙ *e*) ⊆ *supp e* − { *atom z* } ∪ { *atom d*}
using *assms*
proof(*nominal-induct e rule*:*ce.strong-induct*)
case (*CE-val v*)
then show *?case* **using** *swap-v-supp ce.fresh ce.supp* **by** *simp*
next
case (*CE-op opp v1 v2*)
hence *df*: *atom d* ♯ *v1* ∧ *atom d* ♯ *v2* **using** *fresh-def e.supp* **by** *force*
have ((*z* ↔ *d*) ∙ (*CE-op opp v1 v2*)) = *CE-op opp* ((*z* ↔ *d*) ∙ *v1*) ((*z* ↔ *d*) ∙ *v2*) **using**
ce.perm-simps flip-commute opp.perm-simps CE-op opp.strong-exhaust x-fresh-b pure-supp
by (*metis* (*full-types*))

hence *supp* ((*z* ↔ *d*) ∙ *CE-op opp v1 v2*) = *supp* (*CE-op opp* ((*z* ↔ *d*) ∙*v1*) ((*z* ↔ *d*) ∙*v2*)) **by** *simp*
also have ... = *supp* ((*z* ↔ *d*) ∙*v1*) ∪ *supp* ((*z* ↔ *d*) ∙*v2*) **using** *ce.supp*
by (*metis* (*mono-tags*, *hide-lams*) *opp.strong-exhaust opp.supp sup-bot.left-neutral*)
also have ... ⊆ (*supp v1* − { *atom z* } ∪ { *atom d*}) ∪ (*supp v2* − { *atom z* } ∪ { *atom d*}) **using**
swap-v-supp CE-op df **by** *blast*
finally show *?case* **using** *ce.supp opp.supp* **by** *blast*

next
case (*CE-fst v*)
then show *?case* **using** *ce.supp ce.fresh swap-v-supp* **by** *auto*
next
case (*CE-snd v*)
then show *?case* **using** *ce.supp ce.fresh swap-v-supp* **by** *auto*

next
case (*CE-len v*)
then show *?case* **using** *ce.supp ce.fresh swap-v-supp* **by** *auto*
next
case (*CE-concat v1 v2*)
then show *?case* **using** *ce.supp ce.fresh swap-v-supp ce.perm-simps*
proof −
have $\forall x\ v\ xa.\ \neg\ atom\ (x::x) \sharp (v::v) \vee supp\ ((xa \leftrightarrow x) \cdot v) \subseteq supp\ v - \{atom\ xa\} \cup \{atom\ x\}$
by (*meson swap-v-supp*)
then show *?thesis*
using *CE-concat ce.supp* **by** *auto*
qed
qed

lemma *swap-c-supp*:
fixes *c*::*c* **and** *d*::*x* **and** *z*::*x*
assumes $atom\ d \sharp c$
shows $supp\ ((z \leftrightarrow d\ ) \cdot c) \subseteq supp\ c - \{\ atom\ z\ \} \cup \{\ atom\ d\}$
using *assms*
proof(*nominal-induct c rule*:*c.strong-induct*)
case (*C-eq e1 e2*)
then show *?case* **using** *swap-ce-supp* **by** *auto*
qed(*auto*+)

lemma *type-e-eq*:
assumes $atom\ z \sharp e$ **and** $atom\ z' \sharp e$
shows $\{\!|\ z : b\ \mid\ [[z]^v]^{ce} == e\ |\!\} = (\{\!|\ z' : b\ \mid\ [[z']^v]^{ce} ==\ e\ |\!\})$
by (*auto*,*metis* (*full-types*) *assms*(*1*) *assms*(*2*) *flip-fresh-fresh fresh-PairD*(*1*) *fresh-PairD*(*2*))

lemma *type-e-eq2*:
assumes $atom\ z \sharp e$ **and** $atom\ z' \sharp e$ **and** $b{=}b'$
shows $\{\!|\ z : b\ \mid\ [[z]^v]^{ce} == e\ |\!\} = (\{\!|\ z' : b'\ \mid\ [[z']^v]^{ce} == e\ |\!\})$
using *assms type-e-eq* **by** *fast*

lemma *e-flip-eq*:
fixes *e*::*e* **and** *ea*::*e*
assumes $atom\ c \sharp (e,\ ea)$ **and** $atom\ c \sharp (x,\ xa,\ e,\ ea)$ **and** $(x \leftrightarrow c) \cdot e = (xa \leftrightarrow c) \cdot ea$
shows $(e = AE\text{-}val\ w \longrightarrow (\exists w'.\ ea = AE\text{-}val\ w' \wedge (x \leftrightarrow c) \cdot w = (xa \leftrightarrow c) \cdot w')) \vee$
$(e = AE\text{-}op\ opp\ v1\ v2 \longrightarrow (\exists v1'\ v2'.\ ea = AS\text{-}op\ opp\ v1'\ v2' \wedge (x \leftrightarrow c) \cdot v1 = (xa \leftrightarrow c) \cdot v1') \wedge (x \leftrightarrow c) \cdot v2 = (xa \leftrightarrow c) \cdot v2') \vee$
$(e = AE\text{-}fst\ v \longrightarrow (\exists v'.\ ea = AE\text{-}fst\ v' \wedge (x \leftrightarrow c) \cdot v = (xa \leftrightarrow c) \cdot v')) \vee$
$(e = AE\text{-}snd\ v \longrightarrow (\exists v'.\ ea = AE\text{-}snd\ v' \wedge (x \leftrightarrow c) \cdot v = (xa \leftrightarrow c) \cdot v')) \vee$
$(e = AE\text{-}len\ v \longrightarrow (\exists v'.\ ea = AE\text{-}len\ v' \wedge (x \leftrightarrow c) \cdot v = (xa \leftrightarrow c) \cdot v')) \vee$
$(e = AE\text{-}concat\ v1\ v2 \longrightarrow (\exists v1'\ v2'.\ ea = AS\text{-}concat\ v1'\ v2' \wedge (x \leftrightarrow c) \cdot v1 = (xa \leftrightarrow c) \cdot v1') \wedge (x \leftrightarrow c) \cdot v2 = (xa \leftrightarrow c) \cdot v2') \vee$
$(e = AE\text{-}app\ f\ v \longrightarrow (\exists v'.\ ea = AE\text{-}app\ f\ \ v' \wedge (x \leftrightarrow c) \cdot v = (xa \leftrightarrow c) \cdot v'))$
by (*metis assms e.perm-simps permute-flip-cancel2*)

lemma *fresh-opp-all*:
fixes *opp*::*opp*
shows $z \sharp opp$

using *e.fresh opp.exhaust opp.fresh* **by** *metis*

lemma *fresh-e-opp-all*:
shows $(z \sharp v1 \wedge z \sharp v2) = z \sharp AE\text{-}op\ opp\ v1\ v2$
using *e.fresh opp.exhaust opp.fresh fresh-opp-all* **by** *simp*

lemma *fresh-e-opp*:
fixes $z::x$
assumes $atom\ z \sharp v1 \wedge atom\ z \sharp v2$
shows $atom\ z \sharp AE\text{-}op\ opp\ v1\ v2$
using *e.fresh opp.exhaust opp.fresh opp.supp* **by** (*metis assms*)

Statements

lemma *branch-s-flip-eq*:
fixes $v::v$ **and** $va::v$
assumes $atom\ c \sharp (v, va)$ **and** $atom\ c \sharp (x, xa, v, va)$ **and** $(x \leftrightarrow c) \cdot s = (xa \leftrightarrow c) \cdot sa$
shows $(s = AS\text{-}val\ w \longrightarrow (\exists w'.\ sa = AS\text{-}val\ w' \wedge (x \leftrightarrow c) \cdot w = (xa \leftrightarrow c) \cdot w')) \vee$
$(s = AS\text{-}seq\ s1\ s2 \longrightarrow (\exists s1'\ s2'.\ sa = AS\text{-}seq\ s1'\ s2' \wedge (x \leftrightarrow c) \cdot s1 = (xa \leftrightarrow c) \cdot s1') \wedge (x \leftrightarrow c) \cdot s2 = (xa \leftrightarrow c) \cdot s2') \vee$
$(s = AS\text{-}if\ v\ s1\ s2 \longrightarrow (\exists v'\ s1'\ s2'.\ sa = AS\text{-}if\ seq\ s1'\ s2' \wedge (x \leftrightarrow c) \cdot s1 = (xa \leftrightarrow c) \cdot s1') \wedge (x \leftrightarrow c) \cdot s2 = (xa \leftrightarrow c) \cdot s2' \wedge (x \leftrightarrow c) \cdot c = (xa \leftrightarrow c) \cdot v')$
by (*metis assms s-branch-s-branch-list.perm-simps permute-flip-cancel2*)

3.2 Context Syntax

3.2.1 Datatypes

type-synonym Φ = *fun-def list*
type-synonym Θ = *type-def list*
type-synonym $\mathcal{B}$ = *bv fset*

datatype Γ =
GNil
| *GCons x∗b∗c* Γ (**infixr** $\#_\Gamma$ *65*)

datatype Δ =
DNil ($[]_\Delta$)
| *DCons u∗τ* Δ (**infixr** $\#_\Delta$ *65*)

3.2.2 Functions and Lemmas

lemma Γ-*induct* [*case-names GNil GCons*] : $P\ GNil \Longrightarrow (\bigwedge x\ b\ c\ \Gamma'.\ P\ \Gamma' \Longrightarrow P\ ((x,b,c)\ \#_\Gamma\ \Gamma')) \Longrightarrow P\ \Gamma$
proof(*induct* Γ *rule*:Γ.*induct*)
case *GNil*
then show *?case* **by** *auto*
next
case (*GCons x1 x2*)
then obtain x **and** b **and** c **where** *x1*=(x,b,c) **using** *prod-cases3* **by** *blast*
then show *?case* **using** *GCons* **by** *presburger*
qed

instantiation Δ :: *pt*
begin

primrec *permute-*Δ
where
DNil-eqvt: $p \cdot DNil = DNil$
| *DCons-eqvt*: $p \cdot (x \#_\Delta xs) = p \cdot x \#_\Delta p \cdot (xs::\Delta)$

instance **by** *standard* (*induct-tac* [!] *x*, *simp-all*)
end

lemmas [*eqvt*] = *permute-*Δ*.simps*

lemma Δ*-induct* [*case-names DNil DCons*] : $P\ DNil \Longrightarrow (\bigwedge u\ t\ \Delta'.\ P\ \Delta' \Longrightarrow P\ ((u,t) \#_\Delta \Delta')) \Longrightarrow P\ \Delta$
proof(*induct* Δ *rule*: Δ*.induct*)
case *DNil*
then show *?case* **by** *auto*
next
case (*DCons x1 x2*)
then obtain *u* **and** *t* **where** *x1*=(*u*,*t*) **by** *fastforce*
then show *?case* **using** *DCons* **by** *presburger*
qed

lemma Φ*-induct* [*case-names PNil PConsNone PConsSome*] : $P\ [] \Longrightarrow (\bigwedge f\ x\ b\ c\ \tau\ s'\ \Phi'.\ P\ \Phi' \Longrightarrow P\ ((AF\text{-}fundef\ f\ (AF\text{-}fun\text{-}typ\text{-}none\ (AF\text{-}fun\text{-}typ\ x\ b\ c\ \tau\ s'))) \#\ \Phi')) \Longrightarrow$
$(\bigwedge f\ bv\ x\ b\ c\ \tau\ s'\ \Phi'.\ P\ \Phi' \Longrightarrow P\ ((AF\text{-}fundef\ f\ (AF\text{-}fun\text{-}typ\text{-}some\ bv\ (AF\text{-}fun\text{-}typ\ x\ b\ c\ \tau\ s'))) \#\ \Phi')) \Longrightarrow P\ \Phi$
proof(*induct* Φ *rule*: *list.induct*)
case *Nil*
then show *?case* **by** *auto*
next
case (*Cons x1 x2*)
then obtain *f* **and** *t* **where** *ft*: *x1* = (*AF-fundef f t*)
by (*meson fun-def.exhaust*)
then show *?case* **proof**(*nominal-induct t rule*:*fun-typ-q.strong-induct*)
case (*AF-fun-typ-some bv ft*)
then show *?case* **using** *Cons ft*
by (*metis fun-typ.exhaust*)
next
case (*AF-fun-typ-none ft*)
then show *?case* **using** *Cons ft*
by (*metis fun-typ.exhaust*)
qed
qed

lemma Θ*-induct* [*case-names TNil AF-typedef AF-typedef-poly*] : $P\ [] \Longrightarrow (\bigwedge tid\ dclist\ \Theta'.\ P\ \Theta' \Longrightarrow P\ ((AF\text{-}typedef\ tid\ dclist) \#\ \Theta')) \Longrightarrow$
$(\bigwedge tid\ bv\ dclist\ \Theta'.\ P\ \Theta' \Longrightarrow P\ ((AF\text{-}typedef\text{-}poly\ tid\ bv\ dclist\ ) \#\ \Theta')) \Longrightarrow P\ \Theta$

proof(*induct* Θ *rule*: *list.induct*)
case *Nil*
then show *?case* **by** *auto*
next
case (*Cons td T*)
show *?case* **by**(*cases td rule*: *type-def.exhaust*, (*simp add*: *Cons*)+)
qed

instantiation Γ :: *pt*
begin

primrec *permute-*Γ
where
GNil-eqvt: $p \cdot GNil = GNil$
| *GCons-eqvt*: $p \cdot (x \#_\Gamma xs) = p \cdot x \#_\Gamma p \cdot (xs::\Gamma)$

instance by *standard* (*induct-tac* [!] *x*, *simp-all*)
end

lemmas [*eqvt*] = *permute-*Γ*.simps*

lemma *G-cons-eqvt*[*simp*]:
fixes $\Gamma::\Gamma$
shows $p \cdot ((x,b,c) \#_\Gamma \Gamma) = ((p \cdot x,\ p \cdot b\ ,\ p \cdot c) \#_\Gamma (p \cdot \Gamma))$ (**is** *?A* = *?B*)
using *Cons-eqvt triple-eqvt supp-b-empty* **by** *simp*

lemma *G-cons-flip*[*simp*]:
fixes $x::x$ **and** $\Gamma::\Gamma$
shows $(x \leftrightarrow x') \cdot ((x'',b,c) \#_\Gamma \Gamma) = (((x \leftrightarrow x') \cdot x'',\ b\ ,\ (x \leftrightarrow x') \cdot c) \#_\Gamma ((x \leftrightarrow x') \cdot \Gamma))$
using *Cons-eqvt triple-eqvt supp-b-empty* **by** *auto*

lemma *G-cons-flip-fresh*[*simp*]:
fixes $x::x$ **and** $\Gamma::\Gamma$
assumes *atom* $x \sharp (c,\Gamma)$ **and** *atom* $x' \sharp (c,\Gamma)$
shows $(x \leftrightarrow x') \cdot ((x',b,c) \#_\Gamma \Gamma) = ((x,\ b\ ,\ c) \#_\Gamma \Gamma)$
using *G-cons-flip flip-fresh-fresh assms* **by** *force*

lemma *G-cons-flip-fresh2*[*simp*]:
fixes $x::x$ **and** $\Gamma::\Gamma$
assumes *atom* $x \sharp (c,\Gamma)$ **and** *atom* $x' \sharp (c,\Gamma)$
shows $(x \leftrightarrow x') \cdot ((x,b,c) \#_\Gamma \Gamma) = ((x',\ b\ ,\ c) \#_\Gamma \Gamma)$
using *G-cons-flip flip-fresh-fresh assms* **by** *force*

lemma *G-cons-flip-fresh3*[*simp*]:
fixes $x::x$ **and** $\Gamma::\Gamma$
assumes *atom* $x \sharp \Gamma$ **and** *atom* $x' \sharp \Gamma$
shows $(x \leftrightarrow x') \cdot ((x',b,c) \#_\Gamma \Gamma) = ((x,\ b\ ,\ (x \leftrightarrow x') \cdot c) \#_\Gamma \Gamma)$
using *G-cons-flip flip-fresh-fresh assms* **by** *force*

lemma *neq-GNil-conv*: $(xs \neq GNil) = (\exists\, y\ ys.\ xs = y\ \#_\Gamma\ ys)$
by (*induct xs*) *auto*

nominal-function *toList* :: $\Gamma \Rightarrow (x*b*c)$ *list* **where**
  toList GNil = []
| *toList* (*GCons xbc G*) = *xbc*#(*toList G*)
apply (*auto*, *simp add*: *eqvt-def toList-graph-aux-def*)
using *neq-GNil-conv surj-pair* **by** *metis*
nominal-termination (*eqvt*)
by *lexicographic-order*

nominal-function *setG* :: $\Gamma \Rightarrow (x*b*c)$ *set* **where**
  setG GNil = {}
| *setG* (*GCons xbc G*) = {*xbc*} ∪ (*setG G*)
apply (*auto*,*simp add*: *eqvt-def setG-graph-aux-def*)
using *neq-GNil-conv surj-pair* **by** *metis*
nominal-termination (*eqvt*)
by *lexicographic-order*

nominal-function *append-g* :: $\Gamma \Rightarrow \Gamma \Rightarrow \Gamma$ (**infixr** @ *65*) **where**
  append-g GNil g = *g*
| *append-g* (*xbc* $\#_\Gamma$ *g1*) *g2* = (*xbc* $\#_\Gamma$ (*g1*@*g2*))
apply (*auto*,*simp add*: *eqvt-def append-g-graph-aux-def*)
using *neq-GNil-conv surj-pair* **by** *metis*
nominal-termination (*eqvt*)
by *lexicographic-order*

nominal-function *dom* :: $\Gamma \Rightarrow x$ *set* **where**
dom Γ = (*fst*‘ (*setG* Γ))
  apply *auto*
  unfolding *eqvt-def dom-graph-aux-def lfp-eqvt setG.eqvt* **by** *simp*
nominal-termination (*eqvt*)
  by *lexicographic-order*

nominal-function *atom-dom* :: $\Gamma \Rightarrow$ *atom set* **where**
atom-dom Γ = *atom*‘(*fst*‘ (*setG* Γ))
  apply *auto*
  unfolding *eqvt-def atom-dom-graph-aux-def lfp-eqvt setG.eqvt* **by** *simp*
nominal-termination (*eqvt*)
  by *lexicographic-order*

3.2.3 Immutable Variable Context Lemmas

lemma *append-GNil*[*simp*]:
  GNil @ *G* = *G*
using *append-g.simps* **by** *auto*

lemma *append-g-setGU* [*simp*]: *setG* (*G1*@*G2*) = *setG G1* ∪ *setG G2*
  by(*induct G1*, *auto*+)

lemma *supp-GNil*:
shows *supp GNil* = {}
by (*simp add*: *supp-def*)

lemma *supp-GCons*:
fixes *xs*::Γ
shows *supp* (*x* #Γ *xs*) = *supp x* ∪ *supp xs*
by (*simp add*: *supp-def Collect-imp-eq Collect-neg-eq*)

lemma *atom-dom-eq*[*simp*]:
fixes *G*::Γ
shows *atom-dom* ((*x*, *b*, *c*) #Γ *G*) = *atom-dom* ((*x*, *b*, *c'*) #Γ *G*)
using *atom-dom.simps setG.simps* **by** *simp*

lemma *dom-append*[*simp*]:
atom-dom (Γ@Γ') = *atom-dom* Γ ∪ *atom-dom* Γ'
using *image-Un append-g-setGU atom-dom.simps* **by** *metis*

lemma *dom-cons*[*simp*]:
atom-dom ((*x*,*b*,*c*) #Γ *G*) = { *atom x* } ∪ *atom-dom G*
using *image-Un append-g-setGU atom-dom.simps* **by** *auto*

lemma *fresh-GNil*[*ms-fresh*]:
shows *a* ♯ *GNil*
by (*simp add*: *fresh-def supp-GNil*)

lemma *fresh-GCons*[*ms-fresh*]:
fixes *xs*::Γ
shows *a* ♯ (*x* #Γ *xs*) ⟷ *a* ♯ *x* ∧ *a* ♯ *xs*
by (*simp add*: *fresh-def supp-GCons*)

lemma *dom-supp-g*[*simp*]:
atom-dom G ⊆ *supp G*
apply(*induct G rule*: Γ-*induct*,*simp*)
using *supp-at-base supp-Pair atom-dom.simps supp-GCons* **by** *fastforce*

lemma *fresh-append-g*[*ms-fresh*]:
fixes *xs*::Γ
shows *a* ♯ (*xs* @ *ys*) ⟷ *a* ♯ *xs* ∧ *a* ♯ *ys*
by (*induct xs*) (*simp-all add*: *fresh-GNil fresh-GCons*)

lemma *append-g-assoc*:
fixes *xs*::Γ
shows (*xs* @ *ys*) @ *zs* = *xs* @ (*ys* @ *zs*)
by (*induct xs*) *simp-all*

lemma *append-g-inside*:
fixes *xs*::Γ

shows xs @ $(x\ \#_\Gamma\ ys) = (xs$ @ $(x\ \#_\Gamma\ GNil))$ @ ys
by(*induct xs*,*auto*+)

lemma *finite*-Γ:
finite (*setG* Γ)
by(*induct* Γ *rule*: Γ-*induct*,*auto*)

lemma *supp*-Γ:
supp Γ = *supp* (*setG* Γ)
proof(*induct* Γ *rule*: Γ-*induct*)
case *GNil*
then show *?case* **using** *supp-GNil setG.simps*
by (*simp add*: *supp-set-empty*)
next
case (*GCons x b c* Γ′)
then show *?case* **using** *supp-GCons setG.simps finite*-Γ *supp-of-finite-union*
using *supp-of-finite-insert* **by** *fastforce*
qed

lemma *supp-of-subset*:
fixes *G*::(′*a*::*fs set*)
assumes *finite G* **and** *finite G*′ **and** $G \subseteq G'$
shows *supp G* ⊆ *supp G*′
using *supp-of-finite-sets assms* **by** (*metis subset-Un-eq supp-of-finite-union*)

lemma *supp-weakening*:
assumes *setG G* ⊆ *setG G*′
shows *supp G* ⊆ *supp G*′
using *supp*-Γ *finite*-Γ **by** (*simp add*: *supp-of-subset assms*)

lemma *fresh-weakening*[*ms-fresh*]:
assumes *setG G* ⊆ *setG G*′ **and** $x \sharp G'$
shows $x \sharp G$
proof(*rule ccontr*)
assume ¬ $x \sharp G$
hence $x \in$ *supp G* **using** *fresh-def* **by** *auto*
hence $x \in$ *supp G*′ **using** *supp-weakening assms* **by** *auto*
thus *False* **using** *fresh-def assms* **by** *auto*
qed

instance Γ :: *fs*
by (*standard*, *induct-tac x*, *simp-all add*: *supp-GNil supp-GCons finite-supp*)

lemma *fresh-gamma-elem*:
fixes Γ::Γ
assumes $a \sharp \Gamma$
and $e \in$ *setG* Γ
shows $a \sharp e$

using *assms* **by**(*induct* Γ,*auto simp add*: *fresh-GCons*)

lemma *fresh-gamma-append*:
  fixes *xs*::Γ
  shows $a \sharp (xs \mathbin{@} ys) \longleftrightarrow a \sharp xs \wedge a \sharp ys$
by (*induct xs*, *simp-all add*: *fresh-GNil fresh-GCons*)

lemma *supp-triple*[*simp*]:
  shows *supp* (*x*, *y*, *z*) = *supp x* ∪ *supp y* ∪ *supp z*
proof −
  have *supp* (*x*,*y*,*z*) = *supp* (*x*,(*y*,*z*)) **by** *auto*
  hence *supp* (*x*,*y*,*z*) = *supp x* ∪ (*supp y* ∪ *supp z*) **using** *supp-Pair* **by** *metis*
  thus *?thesis* **by** *auto*
qed

lemma *supp-append-g*:
  fixes *xs*::Γ
  shows *supp* (*xs* @ *ys*) = *supp xs* ∪ *supp ys*
by(*induct xs*,*auto simp add*: *supp-GNil supp-GCons*)

lemma *fresh-in-g*[*simp*]:
  fixes Γ::Γ **and** *x'*::*x*
  shows *atom x'* ♯ Γ' @ (*x*, *b0*, *c0*) #$_\Gamma$ Γ = (*atom x'* ∉ *supp* Γ' ∪ *supp x* ∪ *supp b0* ∪ *supp c0* ∪ *supp* Γ)
proof −
   have *atom x'* ♯ Γ' @ (*x*, *b0*, *c0*) #$_\Gamma$ Γ = (*atom x'* ∉ *supp* (Γ' @((*x*,*b0*,*c0*) #$_\Gamma$ Γ)))
     using *fresh-def* **by** *auto*
   also have ... = (*atom x'* ∉ *supp* Γ' ∪ *supp* ((*x*,*b0*,*c0*) #$_\Gamma$ Γ)) **using** *supp-append-g* **by** *fast*
   also have ... = (*atom x'* ∉ *supp* Γ' ∪ *supp x* ∪ *supp b0* ∪ *supp c0* ∪ *supp* Γ) **using** *supp-GCons supp-append-g supp-triple* **by** *auto*
   finally show *?thesis* **by** *fast*
 qed

lemma *fresh-suffix*[*ms-fresh*]:
 fixes Γ::Γ
  assumes *atom x* ♯ Γ'@Γ
  shows *atom x* ♯ Γ
using *assms* **proof**(*induct* Γ' *rule*: Γ-*induct*)
  case *GNil*
  then show *?thesis* **by** *auto*
next
  case (*GCons x' b' c'* Γ')
  hence *atom x* ♯ ((*x'*, *b'*, *c'*) #$_\Gamma$ (Γ' @ Γ)) **using** *append-g.simps* **by** *auto*
  hence *atom x* ♯ (Γ' @ Γ) **using** *fresh-GCons* **by** *auto*
  then show *?thesis* **using** *GCons* **by** *auto*
qed

lemma *not-GCons-self* [*simp*]:
  fixes *xs*::Γ

shows $xs \neq x \#_\Gamma xs$
by (*induct xs*) *auto*

lemma *not-GCons-self2* [*simp*]:
fixes $xs::\Gamma$
shows $x \#_\Gamma xs \neq xs$
by (*rule not-GCons-self* [*symmetric*])

lemma *fresh-restrict*:
fixes $y::x$ **and** $\Gamma::\Gamma$
assumes *atom* $y \sharp (\Gamma' @ (x, b, c) \#_\Gamma \Gamma)$
shows *atom* $y \sharp (\Gamma'@\Gamma)$
using *assms* **proof**(*induct* Γ' *rule*: Γ-*induct*)
case *GNil*
then show *?case* **using** *fresh-GCons fresh-GNil* **by** *auto*
next
case (*GCons* x' b' c' Γ'')
then show *?case* **using** *fresh-GCons fresh-GNil* **by** *auto*
qed

lemma *fresh-dom-free*:
assumes *atom* $x \sharp \Gamma$
shows $(x,b,c) \notin setG\ \Gamma$
using *assms* **proof**(*induct* Γ *rule*: Γ-*induct*)
case *GNil*
then show *?case* **by** *auto*
next
case (*GCons* x' b' c' Γ')
hence $x \neq x'$ **using** *fresh-def fresh-GCons fresh-Pair supp-at-base* **by** *blast*
moreover have *atom* $x \sharp \Gamma'$ **using** *fresh-GCons GCons* **by** *auto*
ultimately show *?case* **using** *setG.simps GCons* **by** *auto*
qed

lemma Γ-*set-intros*: $x \in setG\ (x \#_\Gamma xs)$ **and** $y \in setG\ xs \Longrightarrow y \in setG\ (x \#_\Gamma xs)$
by *simp+*

lemma *fresh-dom-free2*:
assumes *atom* $x \notin$ *atom-dom* Γ
shows $(x,b,c) \notin setG\ \Gamma$
using *assms* **proof**(*induct* Γ *rule*: Γ-*induct*)
case *GNil*
then show *?case* **by** *auto*
next
case (*GCons* x' b' c' Γ')
hence $x \neq x'$ **using** *fresh-def fresh-GCons fresh-Pair supp-at-base* **by** *auto*
moreover have *atom* $x \notin$ *atom-dom* Γ' **using** *fresh-GCons GCons* **by** *auto*
ultimately show *?case* **using** *setG.simps GCons* **by** *auto*
qed

3.2.4 Mutable Variable Context Lemmas

lemma *supp-DNil*:

shows *supp DNil* = {}
by (*simp add*: *supp-def*)

lemma *supp-DCons*:
fixes *xs*::Δ
shows *supp* (*x* #Δ *xs*) = *supp x* ∪ *supp xs*
by (*simp add*: *supp-def Collect-imp-eq Collect-neg-eq*)

lemma *fresh-DNil*[*ms-fresh*]:
shows *a* ♯ *DNil*
by (*simp add*: *fresh-def supp-DNil*)

lemma *fresh-DCons*[*ms-fresh*]:
fixes *xs*::Δ
shows *a* ♯ (*x* #Δ *xs*) ⟷ *a* ♯ *x* ∧ *a* ♯ *xs*
by (*simp add*: *fresh-def supp-DCons*)

instance Δ :: *fs*
by (*standard*, *induct-tac x*, *simp-all add*: *supp-DNil supp-DCons finite-supp*)

3.2.5 Lookup Functions

nominal-function *lookup* :: Γ ⇒ *x* ⇒ (*b*∗*c*) *option* **where**
lookup GNil x = *None*
| *lookup* ((*x*,*b*,*c*)#Γ *G*) *y* = (*if x*=*y then Some* (*b*,*c*) *else lookup G y*)
apply(*auto*)
apply (*simp add*: *eqvt-def lookup-graph-aux-def*)
by (*metis neq-GNil-conv surj-pair*)
nominal-termination (*eqvt*)
by *lexicographic-order*

nominal-function *replace-in-g* :: Γ ⇒ *x* ⇒ *c* ⇒ Γ (-[-⟼-] [*1000*,*0*,*0*] *200*) **where**
replace-in-g GNil - - = GNil
| *replace-in-g* ((*x*,*b*,*c*)#Γ *G*) *x*′ *c*′ = (*if x*=*x*′ *then* ((*x*,*b*,*c*′)#Γ *G*) *else* (*x*,*b*,*c*)#Γ(*replace-in-g G x*′ *c*′))
apply(*auto*,*simp add*: *eqvt-def replace-in-g-graph-aux-def*)
using *surj-pair* Γ.*exhaust* **by** *metis*
nominal-termination (*eqvt*) **by** *lexicographic-order*

Functions for looking up data-constructors in the Pi context

nominal-function *lookup-fun* :: Φ ⇒ *f* ⇒ *fun-def option* **where**
lookup-fun [] *g* = *None*
| *lookup-fun* ((*AF-fundef f ft*)#Π) *g* = (*if* (*f*=*g*) *then Some* (*AF-fundef f ft*) *else lookup-fun* Π *g*)

apply(*auto*,*simp add*: *eqvt-def lookup-fun-graph-aux-def*)
by (*metis fun-def*.*exhaust neq-Nil-conv*)
nominal-termination (*eqvt*) **by** *lexicographic-order*

nominal-function *lookup-td* :: Θ ⇒ *string* ⇒ *type-def option* **where**
lookup-td [] *g* = *None*
| *lookup-td* ((*AF-typedef s lst*) # (Θ::Θ)) *g* = (*if* (*s* = *g*) *then Some* (*AF-typedef s lst*) *else lookup-td* Θ *g*)

```
| lookup-td ((AF-typedef-poly s bv lst ) # (Θ::Θ)) g = (if (s = g) then Some (AF-typedef-poly s bv lst
) else lookup-td Θ g)
  apply(auto,simp add: eqvt-def lookup-td-graph-aux-def )
  by (metis type-def.exhaust neq-Nil-conv)
nominal-termination (eqvt) by lexicographic-order

nominal-function name-of-type ::type-def ⇒ f where
  name-of-type (AF-typedef f - ) = f
| name-of-type (AF-typedef-poly f - -) = f
apply(auto,simp add: eqvt-def name-of-type-graph-aux-def )
using type-def.exhaust by blast
nominal-termination (eqvt) by lexicographic-order

nominal-function name-of-fun ::fun-def ⇒ f where
  name-of-fun (AF-fundef f ft) = f
apply(auto,simp add: eqvt-def name-of-fun-graph-aux-def )
using fun-def.exhaust by blast
nominal-termination (eqvt) by lexicographic-order


nominal-function remove2 :: 'a::pt ⇒ 'a list ⇒ 'a list where
remove2 x [] = [] |
remove2 x (y # xs) = (if x = y then xs else y # remove2 x xs)
apply (simp add: eqvt-def remove2-graph-aux-def )
apply auto+
by (meson list.exhaust)
nominal-termination (eqvt) by lexicographic-order


nominal-function base-for-lit :: l ⇒ b where
  base-for-lit (L-true) = B-bool
| base-for-lit (L-false) = B-bool
| base-for-lit (L-num n) = B-int
| base-for-lit (L-unit) = B-unit
| base-for-lit (L-bitvec v) = B-bitvec
apply (auto simp: eqvt-def base-for-lit-graph-aux-def )
using l.strong-exhaust by blast
nominal-termination (eqvt) by lexicographic-order

lemma neq-DNil-conv: (xs ≠ DNil) = (∃ y ys. xs = y #Δ ys)
  by (induct xs) auto


nominal-function setD :: Δ ⇒ (u*τ) set where
  setD DNil = {}
| setD (DCons xbc G) = {xbc} ∪ (setD G)
apply (auto,simp add: eqvt-def setD-graph-aux-def )
using neq-DNil-conv surj-pair by metis
nominal-termination (eqvt)
  by lexicographic-order
```

lemma *eqvt-triple*:
fixes $y::'a::at$ **and** $ya::'a::at$ **and** $xa::'c::at$ **and** $va::'d::fs$ **and** $s::s$ **and** $sa::s$ **and** $f::s*'c*'d \Rightarrow s$
assumes *atom* $y \sharp (xa, va)$ **and** *atom* $ya \sharp (xa, va)$ **and**
$\forall c.$ *atom* $c \sharp (s, sa) \longrightarrow$ *atom* $c \sharp (y, ya, s, sa) \longrightarrow (y \leftrightarrow c) \cdot s = (ya \leftrightarrow c) \cdot sa$
and *eqvt-at* f (s,xa,va) **and** *eqvt-at* f (sa,xa,va) **and**
atom $c \sharp (s, va, xa, sa)$ **and** *atom* $c \sharp (y, ya, f\ (s, xa, va), f\ (sa, xa, va))$
shows $(y \leftrightarrow c) \cdot f\ (s, xa, va) = (ya \leftrightarrow c) \cdot f\ (sa, xa, va)$
proof −
have $(y \leftrightarrow c) \cdot f\ (s, xa, va) = f\ (\ (y \leftrightarrow c) \cdot (s,xa,va))$ **using** *assms eqvt-at-def* **by** *metis*
also have ... $= f\ (\ (y \leftrightarrow c) \cdot s, (y \leftrightarrow c) \cdot xa\ , (y \leftrightarrow c) \cdot va)$ **by** *auto*
also have ... $= f\ (\ (ya \leftrightarrow c) \cdot sa, (ya \leftrightarrow c) \cdot xa\ , (ya \leftrightarrow c) \cdot va)$ **proof** −
have $(y \leftrightarrow c) \cdot s = (ya \leftrightarrow c) \cdot sa$ **using** *assms Abs1-eq-iff-all* **by** *auto*
moreover have $((y \leftrightarrow c) \cdot xa) = ((ya \leftrightarrow c) \cdot xa)$ **using** *assms flip-fresh-fresh fresh-prodN* **by** *metis*
moreover have $((y \leftrightarrow c) \cdot va) = ((ya \leftrightarrow c) \cdot va)$ **using** *assms flip-fresh-fresh fresh-prodN* **by** *metis*
ultimately show *?thesis* **by** *auto*
qed
also have ... $= f\ (\ (ya \leftrightarrow c) \cdot (sa,xa,va))$ **by** *auto*
finally show *?thesis* **using** *assms eqvt-at-def* **by** *metis*
qed

end

Chapter 4

Immutable Variable Substitution

4.1 Class

class *has-subst-v* = *fs* +
  fixes *subst-v* :: *'a*::*fs* ⇒ *x* ⇒ *v* ⇒ *'a*::*fs* (-[-::=-]$_v$ [*1000*,*50*,*50*] *1000*)
  assumes *fresh-subst-v-if*: *y* ♯ (*subst-v a x v*) ⟷ (*atom x* ♯ *a* ∧ *y* ♯ *a*) ∨ (*y* ♯ *v* ∧ (*y* ♯ *a* ∨ *y* = *atom x*))
   and *forget-subst-v*[*simp*]: *atom x* ♯ *a* ⟹ *subst-v a x v* = *a*
   and *subst-v-id*[*simp*]: *subst-v a x* (*V-var x*) = *a*
   and *eqvt*[*simp*,*eqvt*]: (*p*::*perm*) ∙ (*subst-v a x v*) = (*subst-v* (*p* ∙ *a*) (*p* ∙*x*) (*p* ∙*v*))
   and *flip-subst-v*[*simp*]: *atom x* ♯ *c* ⟹ ((*x* ↔ *z*) ∙ *c*) = *c*[*z*::=[*x*]^v]$_v$
   and *flip-subst-subst-v*[*simp*]: *atom x* ♯ *c* ⟹ ((*x* ↔ *z*) ∙ *c*)[*x*::=*v*]$_v$ = *c*[*z*::=*v*]$_v$
begin

lemma *subst-v-flip-eq-one*:
  fixes *z1*::*x* **and** *z2*::*x* **and** *x1*::*x* **and** *x2*::*x*
  assumes [[*atom z1*]]*lst*. *c1* = [[*atom z2*]]*lst*. *c2*
     and *atom x1* ♯ (*z1*,*z2*,*c1*,*c2*)
    shows (*c1*[*z1*::=[*x1*]^v]$_v$) = (*c2*[*z2*::=[*x1*]^v]$_v$)
proof −
  have (*c1*[*z1*::=[*x1*]^v]$_v$) = (*x1* ↔ *z1*) ∙ *c1* **using** *assms flip-subst-v* **by** *auto*
  moreover have (*c2*[*z2*::=[*x1*]^v]$_v$) = (*x1* ↔ *z2*) ∙ *c2* **using** *assms flip-subst-v* **by** *auto*
  ultimately show *?thesis* **using** *Abs1-eq-iff-all*(*3*)[*of z1 c1 z2 c2 z1*] *assms*
    by (*metis Abs1-eq-iff-fresh*(*3*) *flip-commute*)
qed

lemma *subst-v-simple-commute*[*simp*]:
  fixes *x*::*x*
  assumes *atom x* ♯ *c*
  shows (*c*[*z*::=[*x*]^v]$_v$)[*x*::=*b*]$_v$ = *c*[*z*::=*b*]$_v$
proof −
  have (*c*[*z*::=[*x*]^v]$_v$)[*x*::=*b*]$_v$ = ((*x* ↔ *z*) ∙ *c*)[*x*::=*b*]$_v$ **using** *flip-subst-v assms* **by** *simp*
  thus *?thesis* **using** *flip-subst-subst-v assms* **by** *simp*
qed

lemma *subst-v-flip-eq-two*:

fixes *z1*::*x* **and** *z2*::*x* **and** *x1*::*x* **and** *x2*::*x*
assumes [[*atom z1*]]*lst*. *c1* = [[*atom z2*]]*lst*. *c2*
shows (*c1*[*z1*::=*b*]$_v$) = (*c2*[*z2*::=*b*]$_v$)
proof −
obtain *x*::*x* **where** ∗:*atom x* ♯ (*z1*,*z2*,*c1*,*c2*) **using** *obtain-fresh* **by** *metis*
hence (*c1*[*z1*::=[*x*]^v]$_v$) = (*c2*[*z2*::=[*x*]^v]$_v$) **using** *subst-v-flip-eq-one*[*OF assms*, *of x*] **by** *metis*
hence (*c1*[*z1*::=[*x*]^v]$_v$)[*x*::=*b*]$_v$ = (*c2*[*z2*::=[*x*]^v]$_v$)[*x*::=*b*]$_v$ **by** *auto*
thus *?thesis* **using** *subst-v-simple-commute* ∗ *fresh-prod4* **by** *metis*
qed

lemma *subst-v-flip-eq-three*:
assumes [[*atom z1*]]*lst*. *c1* = [[*atom z1'*]]*lst*. *c1'* **and** *atom x* ♯ *c1* **and** *atom x'* ♯ (*x*,*z1*,*z1'*, *c1*, *c1'*)
shows (*x* ↔ *x'*) ∙ (*c1*[*z1*::=[*x*]^v]$_v$) = *c1'*[*z1'*::=[*x'*]^v]$_v$
proof −
have *atom x'* ♯ *c1*[*z1*::=[*x*]^v]$_v$ **using** *assms fresh-subst-v-if* **by** *simp*
hence (*x* ↔ *x'*) ∙ (*c1*[*z1*::=[*x*]^v]$_v$) = *c1*[*z1*::=[*x*]^v]$_v$[*x*::=[*x'*]^v]$_v$ **using** *flip-subst-v*[*of x' c1*[*z1*::=[*x*]^v]$_v$ *x*] *flip-commute* **by** *metis*
also have ... = *c1*[*z1*::=[*x'*]^v]$_v$ **using** *subst-v-simple-commute fresh-prod4 assms* **by** *auto*
also have ... = *c1'*[*z1'*::=[*x'*]^v]$_v$ **using** *subst-v-flip-eq-one*[*of z1 c1 z1' c1' x'*] **using** *assms* **by** *auto*
finally show *?thesis* **by** *auto*
qed

end

4.2 Values

nominal-function
subst-vv :: *v* ⇒ *x* ⇒ *v* ⇒ *v* **where**
subst-vv (*V-lit l*) *x v* = *V-lit l*
| *subst-vv* (*V-var y*) *x v* = (*if x* = *y then v else V-var y*)
| *subst-vv* (*V-cons tyid c v'*) *x v* = *V-cons tyid c* (*subst-vv v' x v*)
| *subst-vv* (*V-consp tyid c b v'*) *x v* = *V-consp tyid c b* (*subst-vv v' x v*)
| *subst-vv* (*V-pair v1 v2*) *x v* = *V-pair* (*subst-vv v1 x v*) (*subst-vv v2 x v*)
apply(*auto simp*: *eqvt-def subst-vv-graph-aux-def*)
by(*metis v.strong-exhaust*)
nominal-termination (*eqvt*) **by** *lexicographic-order*

abbreviation
subst-vv-abbrev :: *v* ⇒ *x* ⇒ *v* ⇒ *v* (-[-::=-]$_{vv}$ [*1000*,*50*,*50*] *1000*)
where
v[*x*::=*v'*]$_{vv}$ ≡ *subst-vv v x v'*

lemma *fresh-subst-vv-if* [*simp*]:
j ♯ *t*[*i*::=*x*]$_{vv}$ = ((*atom i* ♯ *t* ∧ *j* ♯ *t*) ∨ (*j* ♯ *x* ∧ (*j* ♯ *t* ∨ *j* = *atom i*)))
using *supp-l-empty* **apply** (*induct t rule*: *v.induct*,*auto simp add*: *subst-vv.simps fresh-def*, *auto*)
apply (*simp add*: *supp-at-base* |*metis b.supp supp-b-empty*)+
done

lemma *forget-subst-vv* [*simp*]: *atom a* ♯ *tm* ⟹ *tm*[*a*::=*x*]$_{vv}$ = *tm*
by (*induct tm rule*: *v.induct*) (*simp-all add*: *fresh-at-base*)

lemma *subst-vv-id* [*simp*]: *tm*[*a*::=*V-var a*]$_{vv}$ = *tm*
by (*induct tm rule*: *v.induct*) *simp-all*

lemma *subst-vv-commute* [*simp*]:
atom j ♯ *tm* ⟹ *subst-vv* (*subst-vv tm i t*) *j u* = *subst-vv tm i* (*subst-vv t j u*)
by (*induct tm rule*: *v.induct*) (*auto simp*: *fresh-Pair*)

lemma *subst-vv-commute2* [*simp*]:
atom j ♯ *t* ⟹ *atom i* ♯ *u* ⟹ *i* ≠ *j* ⟹ *subst-vv* (*subst-vv tm i t*) *j u* = *subst-vv* (*subst-vv tm j u*) *i t*
by (*induct tm rule*: *v.induct*) *auto*

lemma *repeat-subst-tvm* [*simp*]: *subst-vv* (*subst-vv tm i t*) *i u* = *subst-vv tm i* (*subst-vv t i u*)
by (*induct tm rule*: *v.induct*) *auto*

lemma *subst-vv-var-flip*[*simp*]:
fixes *v*::*v*
assumes *atom y* ♯ *v*
shows (*y* ↔ *x*) ∙ *v* = *v* [*x*::=*V-var y*]$_{vv}$
using *assms* **apply**(*induct v rule*:*v.induct*)
apply *auto*
using *l.fresh l.perm-simps l.strong-exhaust supp-l-empty permute-pure permute-list.simps fresh-def flip-fresh-fresh* **apply** *fastforce*
using *permute-pure* **apply** *blast+*
done

instantiation *v* :: *has-subst-v*
begin

definition
subst-v = *subst-vv*

instance proof
fix *j*::*atom* **and** *i*::*x* **and** *x*::*v* **and** *t*::*v*
show (*j* ♯ *subst-v t i x*) = ((*atom i* ♯ *t* ∧ *j* ♯ *t*) ∨ (*j* ♯ *x* ∧ (*j* ♯ *t* ∨ *j* = *atom i*)))
using *fresh-subst-vv-if*[*of j t i x*] *subst-v-v-def* **by** *metis*
fix *a*::*x* **and** *tm*::*v* **and** *x*::*v*
show *atom a* ♯ *tm* ⟹ *subst-v tm a x* = *tm*
using *forget-subst-vv subst-v-v-def* **by** *simp*

fix *a*::*x* **and** *tm*::*v*
show *subst-v tm a* (*V-var a*) = *tm* **using** *subst-vv-id subst-v-v-def* **by** *simp*

fix *p*::*perm* **and** *x1*::*x* **and** *v*::*v* **and** *t1*::*v*
show *p* ∙ *subst-v t1 x1 v* = *subst-v* (*p* ∙ *t1*) (*p* ∙ *x1*) (*p* ∙ *v*)
using *subst-vv-commute subst-v-v-def* **by** *simp*

fix *x*::*x* **and** *c*::*v* **and** *z*::*x*
show *atom x* ♯ *c* ⟹ ((*x* ↔ *z*) ∙ *c*) = *c*[*z*::=[*x*]^{v}]$_{v}$

using *subst-vv-var-flip subst-v-v-def* **by** *simp*

fix $x::x$ **and** $c::v$ **and** $z::x$
show $atom\ x \sharp c \Longrightarrow ((x \leftrightarrow z) \cdot c)[x::=v]_v = c[z::=v]_v$
using *subst-vv-var-flip subst-v-v-def* **by** *simp*
qed

end

4.3 Expressions

nominal-function *subst-ev* :: $e \Rightarrow x \Rightarrow v \Rightarrow e$ **where**
$subst\text{-}ev\ (\ (AE\text{-}val\ v')\ )\ x\ v = (\ (AE\text{-}val\ (subst\text{-}vv\ v'\ x\ v))\ )$
| $subst\text{-}ev\ (\ (AE\text{-}app\ f\ v')\ )\ x\ v = (\ (AE\text{-}app\ f\ (subst\text{-}vv\ v'\ x\ v\ ))\ )$
| $subst\text{-}ev\ (\ (AE\text{-}appP\ f\ b\ v')\ )\ x\ v = (\ (AE\text{-}appP\ f\ b\ (subst\text{-}vv\ v'\ x\ v\ ))\ )$
| $subst\text{-}ev\ (\ (AE\text{-}op\ opp\ v1\ v2)\ )\ x\ v = (\ (AE\text{-}op\ opp\ (subst\text{-}vv\ v1\ x\ v\ )\ (subst\text{-}vv\ v2\ x\ v\ ))\ )$
| $subst\text{-}ev\ [\#1\ v']^e\ x\ v = [\#1\ (subst\text{-}vv\ v'\ x\ v\ )]^e$
| $subst\text{-}ev\ [\#2\ v']^e\ x\ v = [\#2\ (subst\text{-}vv\ v'\ x\ v\ )]^e$
| $subst\text{-}ev\ (\ (AE\text{-}mvar\ u))\ x\ v = AE\text{-}mvar\ u$
| $subst\text{-}ev\ [|\ v'\ |]^e\ x\ v = [|\ (subst\text{-}vv\ \ v'\ x\ v\ )\ |]^e$
| $subst\text{-}ev\ (\ AE\text{-}concat\ v1\ v2)\ x\ v = AE\text{-}concat\ (subst\text{-}vv\ v1\ x\ v\ )\ (subst\text{-}vv\ v2\ x\ v\ )$
| $subst\text{-}ev\ (\ AE\text{-}split\ v1\ v2)\ x\ v = AE\text{-}split\ (subst\text{-}vv\ v1\ x\ v\ )\ (subst\text{-}vv\ v2\ x\ v\ )$
by(*simp add*: *eqvt-def subst-ev-graph-aux-def*,*auto*)(*meson e.strong-exhaust*)

nominal-termination (*eqvt*) **by** *lexicographic-order*

abbreviation
$subst\text{-}ev\text{-}abbrev :: e \Rightarrow x \Rightarrow v \Rightarrow e\ (\text{-}[\text{-}::=\text{-}]_{ev}\ [1000,50,50]\ 500)$
where
$e[x::=v']_{ev} \equiv subst\text{-}ev\ e\ x\ v'$

lemma *size-subst-ev* [*simp*]: *size* (*subst-ev A i x*) = *size A*
apply (*nominal-induct A avoiding*: *i x rule*: *e.strong-induct*)
apply *auto*
done

lemma *forget-subst-ev* [*simp*]: $atom\ a \sharp A \Longrightarrow subst\text{-}ev\ A\ a\ x = A$
apply (*nominal-induct A avoiding*: *a x rule*: *e.strong-induct*)
apply(*auto simp*: *fresh-at-base*)
done

lemma *subst-ev-id* [*simp*]: *subst-ev A a* (*V-var a*) = *A*
by (*nominal-induct A avoiding*: *a rule*: *e.strong-induct*) (*auto simp*: *fresh-at-base*)

lemma *fresh-subst-ev-if* [*simp*]:
$j \sharp (subst\text{-}ev\ A\ i\ x\ ) = ((atom\ i \sharp A \wedge j \sharp A) \vee (j \sharp x \wedge (j \sharp A \vee j = atom\ i)))$
apply (*induct A rule*: *e.induct*)
apply(*auto simp add*: *subst-ev.simps fresh-def fresh-subst-vv-if subst-vv.simps*)

apply (*metis* (*no-types*) *fresh-def fresh-subst-vv-if*)+
apply (*metis b.supp supp-b-empty fresh-opp-all fresh-def*)

apply (*metis* (*no-types*) *fresh-def fresh-subst-vv-if*)+
apply (*metis b.supp supp-b-empty fresh-opp-all fresh-def*)
apply (*metis* (*no-types*) *fresh-def fresh-subst-vv-if*)+
apply (*metis b.supp supp-b-empty fresh-opp-all fresh-def*)
apply (*metis* (*no-types*) *fresh-def fresh-subst-vv-if*)+
apply (*metis b.supp supp-b-empty fresh-opp-all fresh-def*)
apply (*blast* | *meson fresh-def fresh-subst-vv-if*)
apply (*metis b.supp supp-b-empty fresh-opp-all fresh-def*)
apply (*metis* (*no-types*) *fresh-def fresh-subst-vv-if*)+
apply (*metis b.supp supp-b-empty fresh-opp-all fresh-def*)
apply (*metis* (*no-types*) *fresh-def fresh-subst-vv-if*)+
apply (*simp add*: *supp-at-base x-not-in-u-atoms*)
apply (*simp add*: *supp-at-base x-not-in-u-atoms*)
apply (*metis* (*no-types*) *fresh-def fresh-subst-vv-if*)+
done

lemma *subst-ev-commute* [*simp*]:
atom j ♯ *A* ⟹ (*subst-ev* (*subst-ev A i t*)) *j u* = *subst-ev A i* (*subst-vv t j u*)
by (*nominal-induct A avoiding*: *i j t u rule*: *e.strong-induct*) (*auto simp*: *fresh-at-base*)

lemma *subst-ev-var-flip*[*simp*]:
fixes *e*::*e* **and** *y*::*x* **and** *x*::*x*
assumes *atom y* ♯ *e*
shows (*y* ↔ *x*) ∙ *e* = *e* [*x*::=*V-var y*]$_{ev}$
using *assms* **apply**(*nominal-induct e rule*:*e.strong-induct*)
apply (*simp add*: *subst-v-v-def*)
apply (*metis* (*mono-tags*, *lifting*) *b.eq-iff b.perm-simps e.fresh e.perm-simps flip-b-id subst-ev.simps subst-vv-var-flip*)
apply (*metis* (*mono-tags*, *lifting*) *b.eq-iff b.perm-simps e.fresh e.perm-simps flip-b-id subst-ev.simps subst-vv-var-flip*)
apply(*rule-tac y*=*x1a* **in** *opp.strong-exhaust*)
using *subst-vv-var-flip flip-def* **apply** (*simp add*: *flip-def permute-pure*)+
done

lemma *subst-ev-flip*:
fixes *e*::*e* **and** *ea*::*e* **and** *c*::*x*
assumes *atom c* ♯ (*e*, *ea*) **and** *atom c* ♯ (*x*, *xa*, *e*, *ea*) **and** (*x* ↔ *c*) ∙ *e* = (*xa* ↔ *c*) ∙ *ea*
shows *e*[*x*::=*v'*]$_{ev}$ = *ea*[*xa*::=*v'*]$_{ev}$
proof −
have *e*[*x*::=*v'*]$_{ev}$ = (*e*[*x*::=*V-var c*]$_{ev}$)[*c*::=*v'*]$_{ev}$ **using** *subst-ev-commute assms* **by** *simp*
also have ... = ((*c* ↔ *x*) ∙ *e*)[*c*::=*v'*]$_{ev}$ **using** *subst-ev-var-flip assms* **by** *simp*
also have ... = ((*c* ↔ *xa*) ∙ *ea*)[*c*::=*v'*]$_{ev}$ **using** *assms flip-commute* **by** *metis*
also have ... = *ea*[*xa*::=*v'*]$_{ev}$ **using** *subst-ev-var-flip assms* **by** *simp*
finally show *?thesis* **by** *auto*
qed

lemma *subst-ev-var*[*simp*]:
(*AE-val* (*V-var x*))[*x*::=[*z*]^{v}]$_{ev}$ = *AE-val* (*V-var z*)
by *auto*

instantiation e :: *has-subst-v*
begin

definition
subst-v = *subst-ev*

instance proof
fix *j*::*atom* **and** *i*::*x* **and** *x*::*v* **and** *t*::*e*
show (*j* ♯ *subst-v t i x*) = ((*atom i* ♯ *t* ∧ *j* ♯ *t*) ∨ (*j* ♯ *x* ∧ (*j* ♯ *t* ∨ *j* = *atom i*)))
using *fresh-subst-ev-if*[*of j t i x*] *subst-v-e-def* **by** *metis*

fix *a*::*x* **and** *tm*::*e* **and** *x*::*v*
show *atom a* ♯ *tm* ⟹ *subst-v tm a x* = *tm*
using *forget-subst-ev subst-v-e-def* **by** *simp*

fix *a*::*x* **and** *tm*::*e*
show *subst-v tm a* (*V-var a*) = *tm* **using** *subst-ev-id subst-v-e-def* **by** *simp*

fix *p*::*perm* **and** *x1*::*x* **and** *v*::*v* **and** *t1*::*e*
show *p* ∙ *subst-v t1 x1 v* = *subst-v* (*p* ∙ *t1*) (*p* ∙ *x1*) (*p* ∙ *v*)
using *subst-ev-commute subst-v-e-def* **by** *simp*

fix *x*::*x* **and** *c*::*e* **and** *z*::*x*
show *atom x* ♯ *c* ⟹ ((*x* ↔ *z*) ∙ *c*) = *c*[*z*::=[*x*]^v]$_v$
using *subst-ev-var subst-v-e-def* **by** *simp*

fix *x*::*x* **and** *c*::*e* **and** *z*::*x*
show *atom x* ♯ *c* ⟹ ((*x* ↔ *z*) ∙ *c*)[*x*::=*v*]$_v$ = *c*[*z*::=*v*]$_v$
using *subst-ev-var-flip subst-v-e-def* **by** *simp*
qed
end

lemma *subst-ev-commute-subst*:
fixes *e*::*e* **and** *w*::*v* **and** *v*::*v*
assumes *atom z* ♯ *v* **and** *atom x* ♯ *w* **and** *x* ≠ *z*
shows *subst-ev* (*e*[*z*::=*w*]$_{ev}$) *x v* = *subst-ev* (*e*[*x*::=*v*]$_{ev}$) *z w*
using *assms* **by**(*nominal-induct e rule*: *e.strong-induct*,*simp*+)

lemma *subst-ev-v-flip1*[*simp*]:
fixes *e*::*e*
assumes *atom z1* ♯ (*z*,*e*) **and** *atom z1'* ♯ (*z*,*e*)
shows(*z1* ↔ *z1'*) ∙ *e*[*z*::=*v*]$_{ev}$ = *e*[*z*::= ((*z1* ↔ *z1'*) ∙ *v*)]$_{ev}$
using *assms* **proof**(*nominal-induct e rule*:*e.strong-induct*)
qed (*simp add*: *flip-def fresh-Pair swap-fresh-fresh*)+

4.4 Expressions in Constraints

nominal-function *subst-cev* :: *ce* ⇒ *x* ⇒ *v* ⇒ *ce* **where**
subst-cev ((*CE-val v'*)) *x v* = ((*CE-val* (*subst-vv v' x v*)))
| *subst-cev* ((*CE-op opp v1 v2*)) *x v* = ((*CE-op opp* (*subst-cev v1 x v*) (*subst-cev v2 x v*)))
| *subst-cev* ((*CE-fst v'*)) *x v* = *CE-fst* (*subst-cev v' x v*)

| *subst-cev* ((*CE-snd v'*)) *x v* = *CE-snd* (*subst-cev v' x v*)
| *subst-cev* ((*CE-len v'*)) *x v* = *CE-len* (*subst-cev v' x v*)
| *subst-cev* (*CE-concat v1 v2*) *x v* = *CE-concat* (*subst-cev v1 x v*) (*subst-cev v2 x v*)
apply (*simp add*: *eqvt-def subst-cev-graph-aux-def*,*auto*)
by (*meson ce.strong-exhaust*)

nominal-termination (*eqvt*) **by** *lexicographic-order*

abbreviation
subst-cev-abbrev :: *ce* ⇒ *x* ⇒ *v* ⇒ *ce* (-[-::=-]$_{cev}$ [*1000*,*50*,*50*] *500*)
where
e[*x*::=*v'*]$_{cev}$ ≡ *subst-cev e x v'*

lemma *size-subst-cev* [*simp*]: *size* (*subst-cev A i x*) = *size A*
by (*nominal-induct A avoiding*: *i x rule*: *ce.strong-induct*,*auto*)

lemma *forget-subst-cev* [*simp*]: *atom a* ♯ *A* ⟹ *subst-cev A a x* = *A*
by (*nominal-induct A avoiding*: *a x rule*: *ce.strong-induct*, *auto simp*: *fresh-at-base*)

lemma *subst-cev-id* [*simp*]: *subst-cev A a* (*V-var a*) = *A*
by (*nominal-induct A avoiding*: *a rule*: *ce.strong-induct*) (*auto simp*: *fresh-at-base*)

lemma *fresh-subst-cev-if* [*simp*]:
j ♯ (*subst-cev A i x*) = ((*atom i* ♯ *A* ∧ *j* ♯ *A*) ∨ (*j* ♯ *x* ∧ (*j* ♯ *A* ∨ *j* = *atom i*)))
proof(*nominal-induct A avoiding*: *i x rule*: *ce.strong-induct*)
case (*CE-op opp v1 v2*)
then show *?case* **using** *fresh-subst-vv-if subst-ev.simps e.supp pure-fresh opp.fresh*
fresh-e-opp
using *fresh-opp-all* **by** *auto*
qed(*auto*)+

lemma *subst-cev-commute* [*simp*]:
atom j ♯ *A* ⟹ (*subst-cev* (*subst-cev A i t*) *j u*) = *subst-cev A i* (*subst-vv t j u*)
by (*nominal-induct A avoiding*: *i j t u rule*: *ce.strong-induct*) (*auto simp*: *fresh-at-base*)

lemma *subst-cev-var-flip*[*simp*]:
fixes *e*::*ce* **and** *y*::*x* **and** *x*::*x*
assumes *atom y* ♯ *e*
shows (*y* ↔ *x*) ∙ *e* = *e* [*x*::=*V-var y*]$_{cev}$
using *assms* **proof**(*nominal-induct e rule*:*ce.strong-induct*)
case (*CE-val v*)
then show *?case* **using** *subst-vv-var-flip* **by** *auto*
next
case (*CE-op opp v1 v2*)
hence *yf*: *atom y* ♯ *v1* ∧ *atom y* ♯ *v2* **using** *ce.fresh* **by** *blast*
have (*y* ↔ *x*) ∙ (*CE-op opp v1 v2*) = *CE-op* ((*y* ↔ *x*) ∙ *opp*) ((*y* ↔ *x*) ∙ *v1*) ((*y* ↔ *x*) ∙ *v2*)
using *opp.perm-simps ce.perm-simps permute-pure ce.fresh opp.strong-exhaust* **by** *presburger*
also have ... = *CE-op* ((*y* ↔ *x*) ∙ *opp*) (*v1*[*x*::=*V-var y*]$_{cev}$) (*v2* [*x*::=*V-var y*]$_{cev}$) **using** *yf*
by (*simp add*: *CE-op.hyps*(*1*) *CE-op.hyps*(*2*))
finally show *?case* **using** *subst-cev.simps opp.perm-simps opp.strong-exhaust*

by (*metis* (*full-types*))
next
case (*CE-fst v*)
then show *?case* **using** *permute-pure subst-vv-var-flip* **by** *simp*
next
case (*CE-snd v*)
then show *?case* **using** *permute-pure subst-vv-var-flip* **by** *simp*
next
case (*CE-len v*)
then show *?case* **using** *permute-pure subst-vv-var-flip* **by** *simp*
next
case (*CE-concat v1 v2*)
then show *?case* **using** *permute-pure subst-vv-var-flip* **by** *simp*
qed

lemma *subst-cev-flip*:
fixes *e*::*ce* **and** *ea*::*ce* **and** *c*::*x*
assumes *atom c* ♯ (*e*, *ea*) **and** *atom c* ♯ (*x*, *xa*, *e*, *ea*) **and** (*x* ↔ *c*) ⋅ *e* = (*xa* ↔ *c*) ⋅ *ea*
shows $e[x::=v']_{cev} = ea[xa::=v']_{cev}$
proof −
have $e[x::=v']_{cev} = (e[x::=V\text{-}var\ c]_{cev})[c::=v']_{cev}$ **using** *subst-ev-commute assms* **by** *simp*
also have ... $= ((c \leftrightarrow x) \cdot e)[c::=v']_{cev}$ **using** *subst-ev-var-flip assms* **by** *simp*
also have ... $= ((c \leftrightarrow xa) \cdot ea)[c::=v']_{cev}$ **using** *assms flip-commute* **by** *metis*
also have ... $= ea[xa::=v']_{cev}$ **using** *subst-ev-var-flip assms* **by** *simp*
finally show *?thesis* **by** *auto*
qed

lemma *subst-cev-var*[*simp*]:
fixes *z*::*x* **and** *x*::*x*
shows $[[x]^v]^{ce}\ [x::=[z]^v]_{cev} = [[z]^v]^{ce}$
by *auto*

instantiation *ce* :: *has-subst-v*
begin

definition
subst-v = *subst-cev*

instance proof
fix *j*::*atom* **and** *i*::*x* **and** *x*::*v* **and** *t*::*ce*
show (*j* ♯ *subst-v t i x*) = ((*atom i* ♯ *t* ∧ *j* ♯ *t*) ∨ (*j* ♯ *x* ∧ (*j* ♯ *t* ∨ *j* = *atom i*)))
using *fresh-subst-cev-if*[*of j t i x*] *subst-v-ce-def* **by** *metis*

fix *a*::*x* **and** *tm*::*ce* **and** *x*::*v*
show *atom a* ♯ *tm* ⟹ *subst-v tm a x* = *tm*
using *forget-subst-cev subst-v-ce-def* **by** *simp*

fix *a*::*x* **and** *tm*::*ce*
show *subst-v tm a* (*V-var a*) = *tm* **using** *subst-cev-id subst-v-ce-def* **by** *simp*

fix p::*perm* **and** $x1$::x **and** v::v **and** $t1$::*ce*
show $p \cdot$ *subst-v t1 x1 v* $=$ *subst-v* $(p \cdot t1)\ (p \cdot x1)\ (p \cdot v)$
using *subst-cev-commute subst-v-ce-def* **by** *simp*

fix x::x **and** c::*ce* **and** z::x
show *atom* $x \sharp c \Longrightarrow ((x \leftrightarrow z) \cdot c) = c\ [z::=V\text{-}var\ x]_v$
using *subst-cev-var subst-v-ce-def* **by** *simp*

fix x::x **and** c::*ce* **and** z::x
show *atom* $x \sharp c \Longrightarrow ((x \leftrightarrow z) \cdot c)[x::=v]_v = c[z::=v]_v$
using *subst-cev-var-flip subst-v-ce-def* **by** *simp*
qed

end

lemma *subst-cev-commute-subst*:
fixes e::*ce* **and** w::v **and** v::v
assumes *atom* $z \sharp v$ **and** *atom* $x \sharp w$ **and** $x \neq z$
shows *subst-cev* $(e[z::=w]_{cev})\ x\ v$ $=$ *subst-cev* $(e[x::=v]_{cev})\ z\ w$
using *assms* **by**(*nominal-induct e rule*: *ce.strong-induct*,*simp*+)

lemma *subst-cev-v-flip1*[*simp*]:
fixes e::*ce*
assumes *atom z1* $\sharp$ (*z*,*e*) **and** *atom z1'* $\sharp$ (*z*,*e*)
shows(*z1* $\leftrightarrow$ *z1'*) $\cdot$ $e[z::=v]_{cev}$ $=$ $e[z::=((z1 \leftrightarrow z1') \cdot v)]_{cev}$
using *assms* **proof**(*nominal-induct e rule*:*ce.strong-induct*)
qed (*simp add*: *flip-def fresh-Pair swap-fresh-fresh*)+

4.5 Constraints

nominal-function *subst-cv* :: $c \Rightarrow x \Rightarrow v \Rightarrow c$ **where**
subst-cv (*C-true*) *x v* $=$ *C-true*
| *subst-cv* (*C-false*) *x v* $=$ *C-false*
| *subst-cv* (*C-conj c1 c2*) *x v* $=$ *C-conj* (*subst-cv c1 x v*) (*subst-cv c2 x v*)
| *subst-cv* (*C-disj c1 c2*) *x v* $=$ *C-disj* (*subst-cv c1 x v*) (*subst-cv c2 x v*)
| *subst-cv* (*C-imp c1 c2*) *x v* $=$ *C-imp* (*subst-cv c1 x v*) (*subst-cv c2 x v*)
| *subst-cv* (*e1* == *e2*) *x v* $=$ ((*subst-cev e1 x v*) == (*subst-cev e2 x v*))
| *subst-cv* (*C-not c*) *x v* $=$ *C-not* (*subst-cv c x v*)
apply (*simp add*: *eqvt-def subst-cv-graph-aux-def*)
apply *auto*
using *c.strong-exhaust* **apply** *metis*
done
nominal-termination (*eqvt*) **by** *lexicographic-order*

abbreviation
subst-cv-abbrev :: $c \Rightarrow x \Rightarrow v \Rightarrow c$ (-[-::=-]$_{cv}$ [*1000*,*50*,*50*] *1000*)
where
$c[x::=v']_{cv}$ $\equiv$ *subst-cv c x v'*

lemma *size-subst-cv* [*simp*]: *size* (*subst-cv A i x*) $=$ *size A*

apply (*nominal-induct A avoiding*: *i x rule*: *c.strong-induct*)
apply *auto*
done

lemma *forget-subst-cv* [*simp*]: *atom a* ♯ *A* ⟹ *subst-cv A a x* = *A*
apply (*nominal-induct A avoiding*: *a x rule*: *c.strong-induct*)
apply(*auto simp*: *fresh-at-base*)
done

lemma *subst-cv-id* [*simp*]: *subst-cv A a* (*V-var a*) = *A*
by (*nominal-induct A avoiding*: *a rule*: *c.strong-induct*) (*auto simp*: *fresh-at-base*)

lemma *fresh-subst-cv-if* [*simp*]:
j ♯ (*subst-cv A i x*) ⟷ (*atom i* ♯ *A* ∧ *j* ♯ *A*) ∨ (*j* ♯ *x* ∧ (*j* ♯ *A* ∨ *j* = *atom i*))
by (*nominal-induct A avoiding*: *i x rule*: *c.strong-induct*, (*auto simp add*: *pure-fresh*)+)

lemma *subst-cv-commute* [*simp*]:
atom j ♯ *A* ⟹ (*subst-cv* (*subst-cv A i t*) *j u*) = *subst-cv A i* (*subst-vv t j u*)
by (*nominal-induct A avoiding*: *i j t u rule*: *c.strong-induct*) (*auto simp*: *fresh-at-base*)

lemma *let-s-size* [*simp*]: *size s* ≤ *size* (*AS-let x e s*)
apply (*nominal-induct s avoiding*: *e x rule*: *s-branch-s-branch-list.strong-induct*(*1*))
apply *auto*
done

lemma *subst-cv-var-flip*[*simp*]:
fixes *c*::*c*
assumes *atom y* ♯ *c*
shows (*y* ↔ *x*) ∙ *c* = *c*[*x*::=*V-var y*]$_{cv}$
using *assms* **by**(*nominal-induct c rule*:*c.strong-induct*,(*simp add*: *flip-subst-v subst-v-ce-def*)+)

instantiation *c* :: *has-subst-v*
begin

definition
subst-v = *subst-cv*

instance proof
fix *j*::*atom* **and** *i*::*x* **and** *x*::*v* **and** *t*::*c*
show (*j* ♯ *subst-v t i x*) = ((*atom i* ♯ *t* ∧ *j* ♯ *t*) ∨ (*j* ♯ *x* ∧ (*j* ♯ *t* ∨ *j* = *atom i*)))
using *fresh-subst-cv-if*[*of j t i x*] *subst-v-c-def* **by** *metis*

fix *a*::*x* **and** *tm*::*c* **and** *x*::*v*
show *atom a* ♯ *tm* ⟹ *subst-v tm a x* = *tm*
using *forget-subst-cv subst-v-c-def* **by** *simp*

fix *a*::*x* **and** *tm*::*c*
show *subst-v tm a* (*V-var a*) = *tm* **using** *subst-cv-id subst-v-c-def* **by** *simp*

fix p::*perm* **and** $x1$::x **and** v::v **and** $t1$::c
show $p \cdot$ *subst-v t1 x1 v* $=$ *subst-v* $(p \cdot t1)\ (p \cdot x1)\ (p \cdot v)$
using *subst-cv-commute subst-v-c-def* **by** *simp*

fix x::x **and** c::c **and** z::x
show *atom* $x \sharp c \Longrightarrow ((x \leftrightarrow z) \cdot c) = c[z::=[x]^v]_v$
using *subst-cv-var-flip subst-v-c-def* **by** *simp*

fix x::x **and** c::c **and** z::x
show *atom* $x \sharp c \Longrightarrow ((x \leftrightarrow z) \cdot c)[x::=v]_v = c[z::=v]_v$
using *subst-cv-var-flip subst-v-c-def* **by** *simp*
qed

end

lemma *subst-cv-var-flip1*[*simp*]:
fixes c::c
assumes *atom* $y \sharp c$
shows $(x \leftrightarrow y) \cdot c = c[x::=V\text{-}var\ y]_{cv}$
using *subst-cv-var-flip flip-commute*
by (*metis assms*)

lemma *subst-cv-v-flip1*[*simp*]:
fixes c::c
assumes *atom z1* $\sharp$ (*z*,*c*) **and** *atom z1'* $\sharp$ (*z*,*c*)
shows$(z1 \leftrightarrow z1') \cdot c[z::=v]_{cv} = c[z::= ((z1 \leftrightarrow z1') \cdot v)]_{cv}$
using *assms* **proof**(*nominal-induct c rule*:*c.strong-induct*)
case (*C-conj c1 c2*)
then show *?case*
by (*metis flip-fresh-fresh fresh-PairD*(*1*) *fresh-PairD*(*2*) *subst-cv.eqvt*)
next
case (*C-disj c1 c2*)
then show *?case* **by** (*metis flip-fresh-fresh fresh-PairD*(*1*) *fresh-PairD*(*2*) *subst-cv.eqvt*)
next
case (*C-not c*)
then show *?case* **by** (*metis flip-fresh-fresh fresh-PairD*(*1*) *fresh-PairD*(*2*) *subst-cv.eqvt*)
next
case (*C-imp c1 c2*)
then show *?case* **by** (*metis flip-fresh-fresh fresh-PairD*(*1*) *fresh-PairD*(*2*) *subst-cv.eqvt*)
next
case (*C-eq e1 e2*)
then show *?case* **using** *subst-ev-v-flip1 flip-def fresh-Pair swap-fresh-fresh*
by (*simp add*: *fresh-Pair*)
qed(*force*+)

lemma *subst-cv-v-flip2*[*simp*]:
fixes c::c
assumes *atom z1* $\sharp$ (*z*,*c*) **and** *atom z1'* $\sharp$ (*z*,*c*)
shows$(z1 \leftrightarrow z1') \cdot c[z::=[z1]^v]_{cv} = c[z::=[z1']^v]_{cv}$
using *subst-cv-v-flip1 assms* **by** *simp*

lemma *subst-cv-v-flip3*[*simp*]:
  fixes *c*::*c*
  assumes *atom z1* ♯ *c* **and** *atom z1'* ♯ *c*
  shows(*z1* ↔ *z1'*) ∙ *c*[*z*::=[*z1*]^v]$_{cv}$ = *c*[*z*::=[*z1'*]^v]$_{cv}$
proof −
  consider *z1'* = *z* | *z1* = *z* | *atom z1* ♯ *z* ∧ *atom z1'* ♯ *z* **by** *force*
  then show *?thesis* **proof**(*cases*)
    case *1*
    then show *?thesis* **using** *1 assms* **by** *auto*
  next
    case *2*
     then show *?thesis* **using** *2 assms* **by** *auto*
  next
    case *3*
    then show *?thesis* **using** *subst-cv-v-flip2 assms* **by** *auto*
  qed
qed

lemma *subst-cv-v-flip*[*simp*]:
  fixes *c*::*c*
  assumes *atom x* ♯ *c*
  shows ((*x* ↔ *z*) ∙ *c*)[*x*::=*v*]$_{cv}$ = *c* [*z*::=*v*]$_{cv}$
  using *assms subst-v-c-def* **by** *auto*

lemma *subst-cv-commute-subst*:
  fixes *c*::*c*
  assumes *atom z* ♯ *v* **and** *atom x* ♯ *w* **and** *x*≠*z*
  shows (*c*[*z*::=*w*]$_{cv}$)[*x*::=*v*]$_{cv}$ = (*c*[*x*::=*v*]$_{cv}$)[*z*::=*w*]$_{cv}$
 using *assms* **proof**(*nominal-induct c rule*: *c.strong-induct*)
  case (*C-eq e1 e2*)
  then show *?case* **using** *subst-cev-commute-subst* **by** *simp*
qed(*force*+)

lemma *subst-cv-eq*[*simp*]:
  assumes *atom z1* ♯ *e1*
  shows (*CE-val* (*V-var z1*) == *e1*)[*z1*::=[*x*]^v]$_{cv}$ = (*CE-val* (*V-var x*) == *e1*) (**is** *?A* = *?B*)
proof −
  have *?A* = (((*CE-val* (*V-var z1*))[*z1*::=[*x*]^v]$_{cev}$) == *e1*) **using** *subst-cv.simps assms* **by** *simp*
  thus *?thesis* **by** *simp*
qed

4.6 Variable Context

nominal-function *subst-gv* :: Γ ⇒ *x* ⇒ *v* ⇒ Γ **where**
  subst-gv GNil x v = *GNil*
| *subst-gv* ((*y*,*b*,*c*) #$_\Gamma$ Γ) *x v* = (*if x* = *y then* Γ *else* ((*y*,*b*,*c*[*x*::=*v*]$_{cv}$)#$_\Gamma$ (*subst-gv* Γ *x v*)))
proof(*goal-cases*)
  case *1*
  then show *?case* **by**(*simp add*: *eqvt-def subst-gv-graph-aux-def*)
next

case (*3 P x*)
then show *?case* **by** (*metis neq-GNil-conv prod-cases3*)
qed(*fast+*)
nominal-termination (*eqvt*) **by** *lexicographic-order*

abbreviation
subst-gv-abbrev :: Γ ⇒ *x* ⇒ *v* ⇒ Γ (-[-::=-]$_{\Gamma v}$ [*1000*,*50*,*50*] *1000*)
where
$g[x::=v]_{\Gamma v}$ ≡ *subst-gv g x v*

lemma *size-subst-gv* [*simp*]: *size* (*subst-gv G i x*) ≤ *size G*
by (*induct G*,*auto*)

lemma *forget-subst-gv* [*simp*]: *atom a* ♯ *G* ⟹ *subst-gv G a x* = *G*
apply (*induct G* ,*auto*)
using *fresh-GCons fresh-PairD*(*1*) *not-self-fresh* **apply** *blast*
apply (*simp add*: *fresh-GCons*)+
done

lemma *fresh-subst-gv*: *atom a* ♯ *G* ⟹ *atom a* ♯ *v* ⟹ *atom a* ♯ *subst-gv G x v*
proof(*induct G*)
case *GNil*
then show *?case* **by** *auto*
next
case (*GCons xbc G*)
obtain *x′* **and** *b′* **and** *c′* **where** *xbc*: *xbc* = (*x′*,*b′*,*c′*) **using** *prod-cases3* **by** *blast*
show *?case* **proof**(*cases x*=*x′*)
case *True*
have *atom a* ♯ *G* **using** *GCons fresh-GCons* **by** *blast*
thus *?thesis* **using** *subst-gv.simps*(*2*)[*of x′ b′ c′ G*] *GCons xbc True* **by** *presburger*
next
case *False*
then show *?thesis* **using** *subst-gv.simps*(*2*)[*of x′ b′ c′ G*] *GCons xbc False fresh-GCons* **by** *simp*
qed
qed

lemma *subst-gv-flip*:
fixes *x*::*x* **and** *xa*::*x* **and** *z*::*x* **and** *c*::*c* **and** *b*::*b* **and** Γ::Γ
assumes *atom xa* ♯ ((*x*, *b*, *c*[*z*::=[*x*]^v]$_{cv}$) #$_\Gamma$ Γ) **and** *atom xa* ♯ Γ **and** *atom x* ♯ Γ **and** *atom x* ♯ (*z*, *c*) **and** *atom xa* ♯ (*z*, *c*)
shows (*x* ↔ *xa*) ∙ ((*x*, *b*, *c*[*z*::=[*x*]^v]$_{cv}$) #$_\Gamma$ Γ) = (*xa*, *b*, *c*[*z*::=*V-var xa*]$_{cv}$) #$_\Gamma$ Γ
proof −
have (*x* ↔ *xa*) ∙ ((*x*, *b*, *c*[*z*::=[*x*]^v]$_{cv}$) #$_\Gamma$ Γ) = (((*x* ↔ *xa*) ∙ *x*, *b*, (*x* ↔ *xa*) ∙ *c*[*z*::=[*x*]^v]$_{cv}$) #$_\Gamma$ ((*x* ↔ *xa*) ∙ Γ))
using *subst Cons-eqvt flip-fresh-fresh* **using** *G-cons-flip* **by** *simp*
also have ... = ((*xa*, *b*, (*x* ↔ *xa*) ∙ *c*[*z*::=[*x*]^v]$_{cv}$) #$_\Gamma$ ((*x* ↔ *xa*) ∙ Γ)) **using** *assms* **by** *fastforce*
also have ... = ((*xa*, *b*, *c*[*z*::=*V-var xa*]$_{cv}$) #$_\Gamma$ ((*x* ↔ *xa*) ∙ Γ)) **using** *assms subst-cv-v-flip1*[*of x z c xa V-var x*] **by** *fastforce*
also have ... = ((*xa*, *b*, *c*[*z*::=*V-var xa*]$_{cv}$) #$_\Gamma$ Γ) **using** *assms flip-fresh-fresh* **by** *blast*
finally show *?thesis* **by** *simp*
qed

4.7 Types

nominal-function *subst-tv* :: $\tau \Rightarrow x \Rightarrow v \Rightarrow \tau$ **where**
atom z $\sharp$ *(x,v)* $\Longrightarrow$ *subst-tv* $\{\!|\ z : b \mid c\ |\!\}$ *x v* $=$ $\{\!|\ z : b \mid c[x::=v]_{cv}\ |\!\}$
apply (*simp add*: *eqvt-def subst-tv-graph-aux-def*)
apply *auto*
apply(*rule-tac y=a* **and** *c=(aa,b)* **in** τ*.strong-exhaust*)
apply (*auto simp*: *eqvt-at-def fresh-star-def fresh-Pair fresh-at-base*)
apply *blast*
proof −
fix *z* :: *x* **and** *c* :: *c* **and** *za* :: *x* **and** *xa* :: *x* **and** *va* :: *v* **and** *ca* :: *c* **and** *cb* :: *x*
assume *a1*: *atom za* $\sharp$ *va* **and** *a2*: *atom z* $\sharp$ *va* **and** *a3*: $\forall$ *cb. atom cb* $\sharp$ *c* $\wedge$ *atom cb* $\sharp$ *ca* $\longrightarrow$ *cb* $\neq$ *z* $\wedge$ *cb* $\neq$ *za* $\longrightarrow$ *c*[*z*::=*V-var cb*]$_{cv}$ = *ca*[*za*::=*V-var cb*]$_{cv}$
assume *a4*: *atom cb* $\sharp$ *c* **and** *a5*: *atom cb* $\sharp$ *ca* **and** *a6*: *cb* $\neq$ *z* **and** *a7*: *cb* $\neq$ *za* **and** *atom cb* $\sharp$ *va* **and** *a8*: *za* $\neq$ *xa* **and** *a9*: *z* $\neq$ *xa*
assume *a10*:*cb* $\neq$ *xa*
note *assms* = *a10 a9 a8 a7 a6 a5 a4 a3 a2 a1*

have *c*[*z*::=*V-var cb*]$_{cv}$ = *ca*[*za*::=*V-var cb*]$_{cv}$ **using** *assms* **by** *auto*
hence *c*[*z*::=*V-var cb*]$_{cv}$[*xa*::=*va*]$_{cv}$ = *ca*[*za*::=*V-var cb*]$_{cv}$[*xa*::=*va*]$_{cv}$ **by** *simp*
moreover have *c*[*z*::=*V-var cb*]$_{cv}$[*xa*::=*va*]$_{cv}$ = *c*[*xa*::=*va*]$_{cv}$[*z*::=*V-var cb*]$_{cv}$ **using** *subst-cv-commute-subst*[*of z va xa V-var cb*] *assms fresh-def v.supp* **by** *fastforce*
moreover have *ca*[*za*::=*V-var cb*]$_{cv}$[*xa*::=*va*]$_{cv}$ = *ca*[*xa*::=*va*]$_{cv}$[*za*::=*V-var cb*]$_{cv}$ **using** *subst-cv-commute-subst*[*of za va xa V-var cb*] *assms fresh-def v.supp* **by** *fastforce*

ultimately show *c*[*xa*::=*va*]$_{cv}$[*z*::=*V-var cb*]$_{cv}$ = *ca*[*xa*::=*va*]$_{cv}$[*za*::=*V-var cb*]$_{cv}$ **by** *simp*
qed
nominal-termination (*eqvt*) **by** *lexicographic-order*

abbreviation
subst-tv-abbrev :: $\tau \Rightarrow x \Rightarrow v \Rightarrow \tau$ (-[-::=-]$_{\tau v}$ [*1000,50,50*] *1000*)
where
t[*x*::=*v*]$_{\tau v}$ $\equiv$ *subst-tv t x v*

lemma *size-subst-tv* [*simp*]: *size* (*subst-tv A i x*) = *size A*
proof (*nominal-induct A avoiding*: *i x rule*: τ*.strong-induct*)
case (*T-refined-type x' b' c'*)
then show *?case* **by** *auto*
qed

lemma *forget-subst-tv* [*simp*]: *atom a* $\sharp$ *A* $\Longrightarrow$ *subst-tv A a x* = *A*
apply (*nominal-induct A avoiding*: *a x rule*: τ*.strong-induct*)
apply(*auto simp*: *fresh-at-base*)
done

lemma *subst-tv-id* [*simp*]: *subst-tv A a* (*V-var a*) = *A*
by (*nominal-induct A avoiding*: *a rule*: τ*.strong-induct*) (*auto simp*: *fresh-at-base*)

lemma *fresh-subst-tv-if* [*simp*]:
j $\sharp$ (*subst-tv A i x*) $\longleftrightarrow$ (*atom i* $\sharp$ *A* $\wedge$ *j* $\sharp$ *A*) $\vee$ (*j* $\sharp$ *x* $\wedge$ (*j* $\sharp$ *A* $\vee$ *j* = *atom i*))
apply (*nominal-induct A avoiding*: *i x rule*: τ*.strong-induct*)
using *fresh-def supp-b-empty x-fresh-b* **by** *auto*

lemma *subst-tv-commute* [*simp*]:
atom y ♯ *τ* ⟹ (*τ*[*x*::= *t*]$_{\tau v}$)[*y*::=*v*]$_{\tau v}$ = *τ*[*x*::= *t*[*y*::=*v*]$_{vv}$]$_{\tau v}$
by (*nominal-induct τ avoiding*: *x y t v rule*: *τ.strong-induct*) (*auto simp*: *fresh-at-base*)

lemma *subst-tv-var-flip* [*simp*]:
fixes *x*::*x* **and** *xa*::*x* **and** *τ*::*τ*
assumes *atom xa* ♯ *τ*
shows (*x* ↔ *xa*) ∙ *τ* = *τ*[*x*::=*V-var xa*]$_{\tau v}$
proof −
obtain *z*::*x* **and** *b* **and** *c* **where** *zbc*: *atom z* ♯ (*x*,*xa*, *V-var xa*) ∧ *τ* = {‖ *z* : *b* | *c* ‖}
using *obtain-fresh-z* **by** (*metis prod.inject subst-tv.cases*)
hence *atom xa* ∉ *supp c* − { *atom z* } **using** *τ.supp*[*of z b c*] *fresh-def supp-b-empty assms*
by *auto*
moreover have *xa* ≠ *z* **using** *zbc fresh-prod3* **by** *force*
ultimately have *xaf*: *atom xa* ♯ *c* **using** *fresh-def* **by** *auto*
have (*x* ↔ *xa*) ∙ *τ* = {‖ *z* : *b* | (*x* ↔ *xa*) ∙ *c* ‖}
by (*metis τ.perm-simps empty-iff flip-at-base-simps*(*3*) *flip-fresh-fresh fresh-PairD*(*1*) *fresh-PairD*(*2*) *fresh-def not-self-fresh supp-b-empty v.fresh*(*2*) *zbc*)
also have ... = {‖ *z* : *b* | *c*[*x*::=*V-var xa*]$_{cv}$ ‖} **using** *subst-cv-var-flip1 xaf* **by** *presburger*
finally show *?thesis* **using** *subst-tv.simps zbc*
using *fresh-PairD*(*1*) *not-self-fresh* **by** *force*
qed

instantiation *τ* :: *has-subst-v*
begin

definition
subst-v = *subst-tv*

instance proof
fix *j*::*atom* **and** *i*::*x* **and** *x*::*v* **and** *t*::*τ*
show (*j* ♯ *subst-v t i x*) = ((*atom i* ♯ *t* ∧ *j* ♯ *t*) ∨ (*j* ♯ *x* ∧ (*j* ♯ *t* ∨ *j* = *atom i*)))

proof(*nominal-induct t avoiding*: *i x rule*:*τ.strong-induct*)
case (*T-refined-type z b c*)
hence *j* ♯ {‖ *z* : *b* | *c* ‖}[*i*::=*x*]$_v$ = *j* ♯ {‖ *z* : *b* | *c*[*i*::=*x*]$_{cv}$ ‖} **using** *subst-tv.simps subst-v-τ-def fresh-Pair* **by** *simp*
also have ... = (*atom i* ♯ {‖ *z* : *b* | *c* ‖} ∧ *j* ♯ {‖ *z* : *b* | *c* ‖} ∨ *j* ♯ *x* ∧ (*j* ♯ {‖ *z* : *b* | *c* ‖} ∨ *j* = *atom i*))
unfolding *τ.fresh* **using** *subst-v-c-def fresh-subst-v-if*
using *T-refined-type.hyps*(*1*) *T-refined-type.hyps*(*2*) *x-fresh-b* **by** *auto*
finally show *?case* **by** *auto*
qed

fix *a*::*x* **and** *tm*::*τ* **and** *x*::*v*
show *atom a* ♯ *tm* ⟹ *subst-v tm a x* = *tm*
apply(*nominal-induct tm avoiding*: *a x rule*:*τ.strong-induct*)
using *subst-v-c-def forget-subst-v subst-tv.simps subst-v-τ-def fresh-Pair* **by** *simp*

fix *a*::*x* **and** *tm*::*τ*
show *subst-v tm a* (*V-var a*) = *tm*
apply(*nominal-induct tm avoiding*: *a rule*:*τ.strong-induct*)

using *subst-v-c-def forget-subst-v subst-tv.simps subst-v-τ-def fresh-Pair* **by** *simp*

fix *p*::*perm* **and** *x1*::*x* **and** *v*::*v* **and** *t1*::*τ*
show *p* ∙ *subst-v t1 x1 v* = *subst-v* (*p* ∙ *t1*) (*p* ∙ *x1*) (*p* ∙ *v*)
apply(*nominal-induct tm avoiding*: *a x rule*:*τ.strong-induct*)
using *subst-v-c-def forget-subst-v subst-tv.simps subst-v-τ-def fresh-Pair* **by** *simp*

fix *x*::*x* **and** *c*::*τ* **and** *z*::*x*
show *atom x* ♯ *c* ⟹ ((*x* ↔ *z*) ∙ *c*) = *c*[*z*::=[*x*]^*v*]_*v*
apply(*nominal-induct c avoiding*: *z x rule*:*τ.strong-induct*)
using *subst-v-c-def flip-subst-v subst-tv.simps subst-v-τ-def fresh-Pair* **by** *auto*

fix *x*::*x* **and** *c*::*τ* **and** *z*::*x*
show *atom x* ♯ *c* ⟹ ((*x* ↔ *z*) ∙ *c*)[*x*::=*v*]_*v* = *c*[*z*::=*v*]_*v*
apply(*nominal-induct c avoiding*: *x v z rule*:*τ.strong-induct*)
using *subst-v-c-def subst-tv.simps subst-v-τ-def fresh-Pair*
by (*metis flip-commute subst-tv-commute subst-tv-var-flip subst-v-τ-def subst-vv.simps*(*2*))
qed

end

lemma *subst-tv-commute-subst*:
fixes *c*::*τ*
assumes *atom z* ♯ *v* **and** *atom x* ♯ *w* **and** *x*≠*z*
shows (*c*[*z*::=*w*]_*τv*)[*x*::=*v*]_*τv* = (*c*[*x*::=*v*]_*τv*)[*z*::=*w*]_*τv*
using *assms* **proof**(*nominal-induct c avoiding*: *x v z w rule*: *τ.strong-induct*)
case (*T-refined-type x1a x2a x3a*)
then show *?case* **using** *subst-cv-commute-subst* **by** *simp*
qed

lemma *type-eq-subst-eq*:
fixes *v*::*v* **and** *c1*::*c*
assumes {| *z1* : *b1* | *c1* |} = {| *z2* : *b2* | *c2* |}
shows *c1*[*z1*::=*v*]_*cv* = *c2*[*z2*::=*v*]_*cv*
using *subst-v-flip-eq-two*[*of z1 c1 z2 c2 v*] *τ.eq-iff assms subst-v-c-def* **by** *simp*

nominal-function *c-of* :: *τ* ⇒ *x* ⇒ *c* **where**
atom z ♯ *x* ⟹ *c-of* (*T-refined-type z b c*) *x* = *c*[*z*::=[*x*]^*v*]_*cv*
proof(*goal-cases*)
case *1*
then show *?case* **using** *eqvt-def c-of-graph-aux-def* **by** *force*
next
case (*2 x y*)
then show *?case* **using** *eqvt-def c-of-graph-aux-def* **by** *force*
next
case (*3 P x*)
then obtain *x1*::*τ* **and** *x2*::*x* **where** ∗:*x* = (*x1*,*x2*) **by** *force*
obtain *z'* **and** *b'* **and** *c'* **where** *x1* = {| *z'* : *b'* | *c'* |} ∧ *atom z'* ♯ *x2* **using** *obtain-fresh-z* **by** *metis*

then show *?case* **using** *3* $*$ **by** *auto*
next
case (*4 z1 x1 b1 c1 z2 x2 b2 c2*)
then show *?case* **using** *subst-v-flip-eq-two τ.eq-iff* **by** (*metis prod.inject type-eq-subst-eq*)
qed

nominal-termination (*eqvt*) **by** *lexicographic-order*

lemma *c-of-eq*:
shows *c-of* {| *x* : *b* | *c* |} *x* = *c*
proof(*nominal-induct* {| *x* : *b* | *c* |} *avoiding*: *x rule*: *τ.strong-induct*)
case (*T-refined-type x' c'*)

moreover hence *c-of* {| *x'* : *b* | *c'* |} *x* = *c'*[*x'*::=*V-var x*]$_{cv}$ **using** *c-of.simps* **by** *auto*
moreover have {| *x'* : *b* | *c'* |} = {| *x* : *b* | *c* |} **using** *T-refined-type τ.eq-iff* **by** *metis*
moreover have *c'*[*x'*::=*V-var x*]$_{cv}$ = *c* **using** *T-refined-type Abs1-eq-iff flip-subst-v subst-v-c-def*
by (*metis subst-cv-id*)
ultimately show *?case* **by** *auto*
qed

lemma *obtain-fresh-z-c-of*:
fixes *t*::'*b*::*fs*
obtains *z* **where** *atom z* ♯ *t* ∧ *τ* = {| *z* : *b-of τ* | *c-of τ z* |}
proof −
obtain *z* **and** *c* **where** *atom z* ♯ *t* ∧ *τ* = {| *z* : *b-of τ* | *c* |} **using** *obtain-fresh-z2* **by** *metis*
moreover hence *c* = *c-of τ z* **using** *c-of.simps* **using** *c-of-eq* **by** *metis*
ultimately show *?thesis*
using *that* **by** *auto*
qed

lemma *c-of-fresh*:
fixes *x*::*x*
assumes *atom x* ♯ (*t*,*z*)
shows *atom x* ♯ *c-of t z*
proof −
obtain *z'* **and** *c'* **where** *z*:*t* = {| *z'* : *b-of t* | *c'* |} ∧ *atom z'* ♯ (*x*,*z*) **using** *obtain-fresh-z-c-of* **by** *metis*
hence $*$:*c-of t z* = *c'*[*z'*::=*V-var z*]$_{cv}$ **using** *c-of.simps fresh-Pair* **by** *metis*
have (*atom x* ♯ *c'* ∨ *atom x* ∈ *set* [*atom z'*]) ∧ *atom x* ♯ *b-of t* **using** *τ.fresh assms z fresh-Pair* **by** *metis*
hence *atom x* ♯ *c'* **using** *fresh-Pair z fresh-at-base*(*2*) **by** *fastforce*
moreover have *atom x* ♯ *V-var z* **using** *assms fresh-Pair v.fresh* **by** *metis*
ultimately show *?thesis* **using** *assms fresh-subst-v-if*[*of atom x c' z' V-var z*] *subst-v-c-def* $*$ **by** *metis*
qed

lemma *c-of-switch*:
fixes *z*::*x*
assumes *atom z* ♯ *t*
shows (*c-of t z*)[*z*::=*V-var x*]$_{cv}$ = *c-of t x*
proof −

obtain z' **and** c' **where** $z{:}t = \{\!|\ z' : b\text{-}of\ t \mid c'\ |\!\} \wedge atom\ z' \sharp (x,z)$ **using** *obtain-fresh-z-c-of* **by** *metis*
hence $(atom\ z \sharp c' \vee atom\ z \in set\ [atom\ z'])$ $\wedge$ $atom\ z \sharp b\text{-}of\ t$ **using** τ*.fresh*[*of atom z z' b-of t c'*] *assms* **by** *metis*
moreover have $atom\ z \notin set\ [atom\ z']$ **using** *z fresh-Pair* **by** *force*
ultimately have $**{:}atom\ z \sharp c'$ **using** *fresh-Pair z fresh-at-base*(2) **by** *metis*

have $(c\text{-}of\ t\ z)[z::=V\text{-}var\ x]_{cv} = c'[z'::=V\text{-}var\ z]_{cv}[z::=V\text{-}var\ x]_{cv}$ **using** *c-of.simps fresh-Pair z* **by** *metis*
also have $... = c'[z'::=V\text{-}var\ x]_{cv}$ **using** *subst-v-simple-commute subst-v-c-def assms c-of.simps z* $**$ **by** *metis*
finally show *?thesis* **using** *c-of.simps*[*of z' x b-of t c'*] *fresh-Pair z* **by** *metis*
qed

lemma *type-eq-subst-eq1*:
fixes *v*::*v* **and** *c1*::*c*
assumes $\{\!|\ z1 : b1\ \mid\ c1\ |\!\} = (\{\!|\ z2 : b2\ \mid\ c2\ |\!\})$ **and** $atom\ z1 \sharp c2$
shows $c1[z1::=v]_{cv} = c2[z2::=v]_{cv}$ **and** *b1*=*b2* **and** $c1 = (z1 \leftrightarrow z2) \cdot c2$
proof −
show $c1[z1::=v]_{cv} = c2[z2::=v]_{cv}$ **using** *type-eq-subst-eq assms* **by** *blast*
show *b1*=*b2* **using** τ*.eq-iff assms* **by** *blast*
have $z1 = z2 \wedge c1 = c2 \vee z1 \neq z2 \wedge c1 = (z1 \leftrightarrow z2) \cdot c2 \wedge atom\ z1 \sharp c2$
using τ*.eq-iff Abs1-eq-iff*[*of z1 c1 z2 c2*] *assms* **by** *blast*
thus $c1 = (z1 \leftrightarrow z2) \cdot c2$ **by** *auto*
qed

lemma *type-eq-subst-eq2*:
fixes *v*::*v* **and** *c1*::*c*
assumes $\{\!|\ z1 : b1\ \mid\ c1\ |\!\} = (\{\!|\ z2 : b2\ \mid\ c2\ |\!\})$
shows $c1[z1::=v]_{cv} = c2[z2::=v]_{cv}$ **and** *b1*=*b2* **and** $[[atom\ z1]]lst.\ c1 = [[atom\ z2]]lst.\ c2$
proof −
show $c1[z1::=v]_{cv} = c2[z2::=v]_{cv}$ **using** *type-eq-subst-eq assms* **by** *blast*
show *b1*=*b2* **using** τ*.eq-iff assms* **by** *blast*
show $[[atom\ z1]]lst.\ c1 = [[atom\ z2]]lst.\ c2$
using τ*.eq-iff assms* **by** *auto*
qed

lemma *type-eq-subst-eq3*:
fixes *v*::*v* **and** *c1*::*c*
assumes $\{\!|\ z1 : b1\ \mid\ c1\ |\!\} = (\{\!|\ z2 : b2\ \mid\ c2\ |\!\})$ **and** $atom\ z1 \sharp c2$
shows $c1 = c2[z2::=V\text{-}var\ z1]_{cv}$ **and** *b1*=*b2*
using *type-eq-subst-eq1 assms subst-v-c-def*
by (*metis subst-cv-var-flip*)+

lemma *type-eq-flip*:
assumes $atom\ x \sharp c$
shows $\{\!|\ z : b\ \mid c\ |\!\} = \{\!|\ x : b \mid (x \leftrightarrow z\ ) \cdot c\ |\!\}$
using τ*.eq-iff Abs1-eq-iff assms*
by (*metis* (*no-types*, *lifting*) *flip-fresh-fresh*)

lemma *c-of-true*:
c-of ⦃ *z'* : *B-bool* | *TRUE* ⦄ *x* = *C-true*
proof(*nominal-induct* ⦃ *z'* : *B-bool* | *TRUE* ⦄ *avoiding*: *x rule*:τ.*strong-induct*)
case (*T-refined-type x1a x3a*)
hence ⦃ *z'* : *B-bool* | *TRUE* ⦄ = ⦃ *x1a* : *B-bool* | *x3a* ⦄ **using** τ.*eq-iff* **by** *metis*
then show *?case* **using** *subst-cv.simps c-of.simps T-refined-type*
type-eq-subst-eq3
by (*metis type-eq-subst-eq*)
qed

lemma *type-eq-subst*:
assumes *atom x* ♯ *c*
shows ⦃ *z* : *b* | *c* ⦄ = ⦃ *x* : *b* | *c*[*z*::=[*x*]^v]$_{cv}$ ⦄
using τ.*eq-iff Abs1-eq-iff assms* **by** *auto*

lemma *type-e-subst-fresh*:
fixes *x*::*x* **and** *z*::*x*
assumes *atom z* ♯ (*x*,*v*) **and** *atom x* ♯ *e*
shows ⦃ *z* : *b* | *CE-val* (*V-var z*) == *e* ⦄[*x*::=*v*]$_{\tau v}$ = ⦃ *z* : *b* | *CE-val* (*V-var z*) == *e* ⦄
using *assms subst-tv.simps subst-cv.simps forget-subst-cev* **by** *simp*

lemma *type-v-subst-fresh*:
fixes *x*::*x* **and** *z*::*x*
assumes *atom z* ♯ (*x*,*v*) **and** *atom x* ♯ *v'*
shows ⦃ *z* : *b* | *CE-val* (*V-var z*) == *CE-val v'* ⦄[*x*::=*v*]$_{\tau v}$ = ⦃ *z* : *b* | *CE-val* (*V-var z*) == *CE-val v'* ⦄
using *assms subst-tv.simps subst-cv.simps* **by** *simp*

lemma *subst-tbase-eq*:
b-of τ = *b-of* τ[*x*::=*v*]$_{\tau v}$
proof −
obtain *z* **and** *b* **and** *c* **where** *zbc*: τ = ⦃ *z*:*b*|*c*⦄ ∧ *atom z* ♯ (*x*,*v*) **using** τ.*exhaust*
by (*metis prod.inject subst-tv.cases*)
hence *b-of* ⦃ *z*:*b*|*c*⦄ = *b-of* ⦃ *z*:*b*|*c*⦄[*x*::=*v*]$_{\tau v}$ **using** *subst-tv.simps* **by** *simp*
thus *?thesis* **using** *zbc* **by** *blast*
qed

lemma *subst-tv-if*:
assumes *atom z1* ♯ (*x*,*v*) **and** *atom z'* ♯ (*x*,*v*)
shows ⦃ *z1* : *b* | *CE-val* (*v'*[*x*::=*v*]$_{vv}$) == *CE-val* (*V-lit l*) *IMP* (*c'*[*x*::=*v*]$_{cv}$)[*z'*::=[*z1*]^v]$_{cv}$ ⦄ =
⦃ *z1* : *b* | *CE-val v'* == *CE-val* (*V-lit l*) *IMP* *c'*[*z'*::=[*z1*]^v]$_{cv}$ ⦄[*x*::=*v*]$_{\tau v}$
using *subst-cv-commute-subst*[*of z' v x V-var z1 c'*] *subst-tv.simps subst-vv.simps*(1) *subst-ev.simps subst-cv.simps assms*
by *simp*

lemma *subst-tv-tid*:
assumes *atom za* ♯ (*x*,*v*)
shows ⦃ *za* : *B-id tid* | *TRUE* ⦄ = ⦃ *za* : *B-id tid* | *TRUE* ⦄[*x*::=*v*]$_{\tau v}$

using *assms subst-tv.simps subst-cv.simps* **by** *presburger*

lemma *b-of-subst*:
b-of $(\tau[x::=v]_{\tau v}) = b\text{-}of\ \tau$
proof −
obtain *z b c* **where** $*{:}\tau = \{\!|\ z : b \mid c\ |\!\} \wedge atom\ z \sharp (x,v)$ **using** *obtain-fresh-z* **by** *metis*
thus *?thesis* **using** *subst-tv.simps* $*$ **by** *auto*
qed

lemma *subst-tv-flip*:
assumes $\tau'[x::=v]_{\tau v} = \tau$ **and** $atom\ x \sharp (v,\tau)$ **and** $atom\ x' \sharp (v,\tau)$
shows $((x' \leftrightarrow x) \cdot \tau')[x'::=v]_{\tau v} = \tau$
proof −
have $(x' \leftrightarrow x) \cdot v = v \wedge (x' \leftrightarrow x) \cdot \tau = \tau$ **using** *assms flip-fresh-fresh* **by** *auto*
thus *?thesis* **using** *subst-tv.eqvt*[*of* $(x' \leftrightarrow x)$ $\tau'\ x\ v$] *assms* **by** *auto*
qed

lemma *subst-cv-true*:
$\{\!|\ z : B\text{-}id\ tid\ \mid TRUE\ |\!\} = \{\!|\ z : B\text{-}id\ tid\ \mid TRUE\ |\!\}[x::=v]_{\tau v}$
proof −
obtain *za*::*x* **where** $atom\ za \sharp (x,v)$ **using** *obtain-fresh* **by** *auto*
hence $\{\!|\ z : B\text{-}id\ tid\ \mid TRUE\ |\!\} = \{\!|\ za{:}\ B\text{-}id\ tid \mid TRUE\ |\!\}$ **using** $\tau.eq\text{-}iff\ Abs1\text{-}eq\text{-}iff$ **by** *fastforce*
moreover have $\{\!|\ za : B\text{-}id\ tid\ \mid TRUE\ |\!\} = \{\!|\ za : B\text{-}id\ tid\ \mid TRUE\ |\!\}[x::=v]_{\tau v}$
using *subst-cv.simps subst-tv.simps* **by** (*simp add*: ‹$atom\ za \sharp (x,\ v)$›)
ultimately show *?thesis* **by** *argo*
qed

lemma *t-eq-supp*:
assumes $(\{\!|\ z : b \mid c\ |\!\}) = (\{\!|\ z1 : b1 \mid c1\ |\!\})$
shows $supp\ c - \{\ atom\ z\ \} = supp\ c1 - \{\ atom\ z1\ \}$
proof −
have $supp\ c - \{\ atom\ z\ \} \cup supp\ b = supp\ c1 - \{\ atom\ z1\ \} \cup supp\ b1$ **using** $\tau.supp\ assms$
by (*metis list.set*(1) *list.simps*(15) *sup-bot.right-neutral supp-b-empty*)
moreover have $supp\ b = supp\ b1$ **using** *assms* $\tau.eq\text{-}iff$ **by** *simp*
moreover have $atom\ z1 \notin supp\ b1 \wedge atom\ z \notin supp\ b$ **using** *supp-b-empty* **by** *simp*
ultimately show *?thesis*
by (*metis* $\tau.eq\text{-}iff\ \tau.supp\ assms$ *b.supp*(1) *list.set*(1) *list.set*(2) *sup-bot.right-neutral*)
qed

lemma *fresh-t-eq*:
fixes *x*::*x*
assumes $(\{\!|\ z : b\ \mid c\ |\!\}) = (\{\!|\ zz : b \mid cc\ |\!\})$ **and** $atom\ x \sharp c$ **and** $x \neq zz$
shows $atom\ x \sharp cc$
proof −
thm $\tau.supp$
have $supp\ c - \{\ atom\ z\ \} \cup supp\ b = supp\ cc - \{\ atom\ zz\ \} \cup supp\ b$ **using** $\tau.supp\ assms$
by (*metis list.set*(1) *list.simps*(15) *sup-bot.right-neutral supp-b-empty*)
moreover have $atom\ x \notin supp\ c$ **using** *assms fresh-def* **by** *blast*
ultimately have $atom\ x \notin supp\ cc - \{\ atom\ zz\ \} \cup supp\ b$ **by** *force*

hence *atom x* $\notin$ *supp cc* **using** *assms* **by** *simp*
thus *?thesis* **using** *fresh-def* **by** *auto*
qed

4.8 Mutable Variable Context

nominal-function *subst-dv* :: $\Delta \Rightarrow x \Rightarrow v \Rightarrow \Delta$ **where**
subst-dv DNil x v = DNil
| *subst-dv* $((u,t)\ \#_\Delta\ \Delta)\ x\ v\ = ((u,t[x::=v]_{\tau v})\ \#_\Delta\ (subst\text{-}dv\ \Delta\ x\ v\ ))$
apply (*simp add*: *eqvt-def subst-dv-graph-aux-def*,*auto*)
using *delete-aux.elims* **by** (*metis* Δ.*exhaust surj-pair*)
nominal-termination (*eqvt*) **by** *lexicographic-order*

abbreviation
subst-dv-abbrev :: $\Delta \Rightarrow x \Rightarrow v \Rightarrow \Delta$ (-[-::=-]$_{\Delta v}$ [*1000*,*50*,*50*] *1000*)
where
$\Delta[x::=v]_{\Delta v}\ \equiv$ *subst-dv* Δ *x v*

nominal-function *dmap* :: $(u*\tau \Rightarrow u*\tau) \Rightarrow \Delta \Rightarrow \Delta$ **where**
dmap f DNil = DNil
| *dmap f* $((u,t)\#_\Delta\Delta)\ = (f\ (u,t)\ \#_\Delta\ (dmap\ f\ \Delta\ ))$
apply (*simp add*: *eqvt-def dmap-graph-aux-def*,*auto*)
using *delete-aux.elims* **by** (*metis* Δ.*exhaust surj-pair*)
nominal-termination (*eqvt*) **by** *lexicographic-order*

lemma *subst-dv-iff*:
$\Delta[x::=v]_{\Delta v} = dmap\ (\lambda(u,t).\ (u,\ t[x::=v]_{\tau v}))\ \Delta$
by(*induct* Δ, *auto*)

lemma *size-subst-dv* [*simp*]: *size* (*subst-dv G i x*) $\leq$ *size G*
by (*induct G*,*auto*)

lemma *forget-subst-dv* [*simp*]: *atom a* $\sharp$ *G* $\Longrightarrow$ *subst-dv G a x = G*
apply (*induct G* ,*auto*)
using *fresh-DCons fresh-PairD*(*1*) *not-self-fresh* **apply** *fastforce*
apply (*simp add*: *fresh-DCons*)+
done

lemma *subst-dv-member*:
assumes $(u,\tau) \in setD\ \Delta$
shows $(u,\ \tau[x::=v]_{\tau v}) \in setD\ (\Delta[x::=v]_{\Delta v})$
using *assms* **by**(*induct* Δ *rule*: Δ-*induct*,*auto*)

lemma *fresh-subst-dv*:
fixes *x*::*x*
assumes *atom xa* $\sharp$ Δ **and** *atom xa* $\sharp$ *v*
shows *atom xa* $\sharp\Delta[x::=v]_{\Delta v}$
using *assms* **proof**(*induct* Δ *rule*:Δ-*induct*)
case *DNil*
then show *?case* **by** *auto*

next
case (*DCons u t* Δ)
then show *?case* **using** *subst-dv.simps subst-v-τ-def fresh-DCons fresh-Pair* **by** *simp*
qed

lemma *fresh-subst-dv-if*:
fixes *j*::*atom* **and** *i*::*x* **and** *x*::*v* **and** *t*::Δ
assumes *j* ♯ *t* ∧ *j* ♯ *x*
shows (*j* ♯ *subst-dv t i x*)
using *assms* **proof**(*induct t rule*: Δ-*induct*)
case *DNil*
then show *?case* **using** *subst-gv.simps fresh-GNil* **by** *auto*
next
case (*DCons u′ t′ D′*)
then show *?case* **unfolding** *subst-dv.simps* **using** *fresh-DCons fresh-subst-tv-if fresh-Pair* **by** *metis*
qed

4.9 Statements

Using ideas from proof at top of AFP/Launchbury/Substitution.thy. Chunks borrowed from there; hence the apply style proofs.

nominal-function (*default case-sum* (λ*x*. *Inl undefined*) (*case-sum* (λ*x*. *Inl undefined*) (λ*x*. *Inr undefined*)))
subst-sv :: *s* ⇒ *x* ⇒ *v* ⇒ *s*
and *subst-branchv* :: *branch-s* ⇒ *x* ⇒ *v* ⇒ *branch-s*
and *subst-branchlv* :: *branch-list* ⇒ *x* ⇒ *v* ⇒ *branch-list* **where**
subst-sv ((*AS-val v′*)) *x v* = (*AS-val* (*subst-vv v′ x v*))
| *atom y* ♯ (*x*,*v*) ⟹ *subst-sv* (*AS-let y e s*) *x v* = (*AS-let y* (*e*[*x*::=*v*]$_{ev}$) (*subst-sv s x v*))
| *atom y* ♯ (*x*,*v*) ⟹ *subst-sv* (*AS-let2 y t s1 s2*) *x v* = (*AS-let2 y* (*t*[*x*::=*v*]$_{\tau v}$) (*subst-sv s1 x v*) (*subst-sv s2 x v*))
| *subst-sv* (*AS-match v′ cs*) *x v* = *AS-match* (*v′*[*x*::=*v*]$_{vv}$) (*subst-branchlv cs x v*)
| *subst-sv* (*AS-assign y v′*) *x v* = *AS-assign y* (*subst-vv v′ x v*)
| *subst-sv* ((*AS-if v′ s1 s2*)) *x v* = (*AS-if* (*subst-vv v′ x v*) (*subst-sv s1 x v*) (*subst-sv s2 x v*))
| *atom u* ♯ (*x*,*v*) ⟹ *subst-sv* (*AS-var u τ v′ s*) *x v* = *AS-var u* (*subst-tv τ x v*) (*subst-vv v′ x v*) (*subst-sv s x v*)
| *subst-sv* (*AS-while s1 s2*) *x v* = *AS-while* (*subst-sv s1 x v*) (*subst-sv s2 x v*)
| *subst-sv* (*AS-seq s1 s2*) *x v* = *AS-seq* (*subst-sv s1 x v*) (*subst-sv s2 x v*)
| *subst-sv* (*AS-assert c s*) *x v* = *AS-assert* (*subst-cv c x v*) (*subst-sv s x v*)
| *atom x1* ♯ (*x*,*v*) ⟹ *subst-branchv* (*AS-branch dc x1 s1*) *x v* = *AS-branch dc x1* (*subst-sv s1 x v*)

| *subst-branchlv* (*AS-final cs*) *x v* = *AS-final* (*subst-branchv cs x v*)
| *subst-branchlv* (*AS-cons cs css*) *x v* = *AS-cons* (*subst-branchv cs x v*) (*subst-branchlv css x v*)
apply (*auto*,*simp add*: *eqvt-def subst-sv-subst-branchv-subst-branchlv-graph-aux-def*)
proof(*goal-cases*)

have *eqvt-at-proj*: ⋀ *s xa va* . *eqvt-at subst-sv-subst-branchv-subst-branchlv-sumC* (*Inl* (*s*, *xa*, *va*)) ⟹

eqvt-at (λ*a*. *projl* (*subst-sv-subst-branchv-subst-branchlv-sumC* (*Inl a*))) (*s*, *xa*, *va*)
apply(*simp add*: *eqvt-at-def*)
apply(*rule*)

apply(*subst Projl-permute*)
apply(*thin-tac* -)+
apply (*simp add*: *subst-sv-subst-branchv-subst-branchlv-sumC-def*)
apply (*simp add*: *THE-default-def*)
apply (*case-tac Ex1* (*subst-sv-subst-branchv-subst-branchlv-graph* (*Inl* (*s*,*xa*,*va*))))
apply *simp*
apply(*auto*)[*1*]
apply (*erule-tac x*=*x* **in** *allE*)
apply *simp*
apply(*cases rule*: *subst-sv-subst-branchv-subst-branchlv-graph.cases*)
apply(*assumption*)
apply(*rule-tac x*=*Sum-Type.projl x* **in** *exI*,*clarify*,*rule the1-equality*,*blast*,*simp* (*no-asm*) *only*: *sum.sel*)+
apply *blast* +

apply(*simp*)+
done

{

case (*1 P x'*)
then show *?case* **proof**(*cases x'*)
case (*Inl a*) **thus** *P*
proof(*cases a*)
case (*fields aa bb cc*)
thus *P* **using** *Inl 1 s-branch-s-branch-list.strong-exhaust fresh-star-insert* **by** *metis*
qed
next
case (*Inr b*) **thus** *P*
proof(*cases b*)
case (*Inl a*) **thus** *P* **proof**(*cases a*)
case (*fields aa bb cc*)
then show *?thesis* **using** *Inr Inl 1 s-branch-s-branch-list.strong-exhaust fresh-star-insert* **by** *metis*
qed
next
case *Inr2*: (*Inr b*) **thus** *P* **proof**(*cases b*)
case (*fields aa bb cc*)
then show *?thesis* **using** *Inr Inr2 1 s-branch-s-branch-list.strong-exhaust fresh-star-insert* **by** *metis*
qed
qed
qed
next
case (*2 y s ya xa va sa c*)
thus *?case* **using** *eqvt-triple eqvt-at-proj* **by** *blast*
next
case (*3 y s2 ya xa va s1a s2a c*)
thus *?case* **using** *eqvt-triple eqvt-at-proj* **by** *blast*
next
case (*4 u s ua xa va sa c*)
moreover have *atom u* ♯ (*xa*, *va*) ∧ *atom ua* ♯ (*xa*, *va*) **using** *fresh-Pair u-fresh-xv* **by** *auto*
ultimately show *?case* **using** *eqvt-triple*[*of u xa va ua s sa*] *subst-sv-def eqvt-at-proj* **by** *metis*

next
case (*5 x1 s1 x1a xa va s1a c*)
thus *?case* **using** *eqvt-triple eqvt-at-proj* **by** *blast*
}
qed
nominal-termination (*eqvt*) **by** *lexicographic-order*

abbreviation
subst-sv-abbrev :: $s \Rightarrow x \Rightarrow v \Rightarrow s$ (-[-::=-]$_{sv}$ [*1000,50,50*] *1000*)
where
$s[x::=v]_{sv} \equiv$ *subst-sv s x v*

abbreviation
subst-branchv-abbrev :: *branch-s* $\Rightarrow x \Rightarrow v \Rightarrow$ *branch-s* (-[-::=-]$_{sv}$ [*1000,50,50*] *1000*)
where
$s[x::=v]_{sv} \equiv$ *subst-branchv s x v*

lemma *size-subst-sv* [*simp*]: *size* (*subst-sv A i x*) = *size A* **and** *size* (*subst-branchv B i x*) = *size B* **and** *size* (*subst-branchlv C i x*) = *size C*
by(*nominal-induct A* **and** *B* **and** *C avoiding*: *i x rule*: *s-branch-s-branch-list.strong-induct*,*auto*)

lemma *forget-subst-sv* [*simp*]: **shows** *atom a* $\sharp$ *A* $\Longrightarrow$ *subst-sv A a x* = *A* **and** *atom a* $\sharp$ *B* $\Longrightarrow$ *subst-branchv B a x* = *B* **and** *atom a* $\sharp$ *C* $\Longrightarrow$ *subst-branchlv C a x* = *C*
by (*nominal-induct A* **and** *B* **and** *C avoiding*: *a x rule*: *s-branch-s-branch-list.strong-induct*,*auto simp*: *fresh-at-base*)

lemma *subst-sv-id* [*simp*]: *subst-sv A a* (*V-var a*) = *A* **and** *subst-branchv B a* (*V-var a*) = *B* **and** *subst-branchlv C a* (*V-var a*) = *C*
proof(*nominal-induct A* **and** *B* **and** *C avoiding*: *a rule*: *s-branch-s-branch-list.strong-induct*)
case (*AS-let x option e s*)
then show *?case*
by (*metis* (*no-types*, *lifting*) *fresh-Pair not-None-eq subst-ev-id subst-sv.simps*(*2*) *subst-sv.simps*(*3*) *subst-tv-id v.fresh*(*2*))
next
case (*AS-match v branch-s*)
then show *?case* **using** *fresh-Pair not-None-eq subst-ev-id subst-sv.simps subst-sv.simps subst-tv-id v.fresh subst-vv-id*
by *metis*
qed(*auto*)+

lemma *fresh-subst-sv-if-rl*:
shows
(*atom x* $\sharp$ *s* $\wedge$ *j* $\sharp$ *s*) $\vee$ (*j* $\sharp$ *v* $\wedge$ (*j* $\sharp$ *s* $\vee$ *j* = *atom x*)) $\Longrightarrow$ *j* $\sharp$ (*subst-sv s x v*) **and**
(*atom x* $\sharp$ *cs* $\wedge$ *j* $\sharp$ *cs*) $\vee$ (*j* $\sharp$ *v* $\wedge$ (*j* $\sharp$ *cs* $\vee$ *j* = *atom x*)) $\Longrightarrow$ *j* $\sharp$ (*subst-branchv cs x v*) **and**
(*atom x* $\sharp$ *css* $\wedge$ *j* $\sharp$ *css*) $\vee$ (*j* $\sharp$ *v* $\wedge$ (*j* $\sharp$ *css* $\vee$ *j* = *atom x*)) $\Longrightarrow$ *j* $\sharp$ (*subst-branchlv css x v*)
apply(*nominal-induct s* **and** *cs* **and** *css avoiding*: *v x rule*: *s-branch-s-branch-list.strong-induct*)
using *pure-fresh* **by** *force*+

lemma *fresh-subst-sv-if-lr*:
shows *j* $\sharp$ (*subst-sv s x v*) $\Longrightarrow$ (*atom x* $\sharp$ *s* $\wedge$ *j* $\sharp$ *s*) $\vee$ (*j* $\sharp$ *v* $\wedge$ (*j* $\sharp$ *s* $\vee$ *j* = *atom x*)) **and**
j $\sharp$ (*subst-branchv cs x v*) $\Longrightarrow$ (*atom x* $\sharp$ *cs* $\wedge$ *j* $\sharp$ *cs*) $\vee$ (*j* $\sharp$ *v* $\wedge$ (*j* $\sharp$ *cs* $\vee$ *j* = *atom x*)) **and**
j $\sharp$ (*subst-branchlv css x v*) $\Longrightarrow$ (*atom x* $\sharp$ *css* $\wedge$ *j* $\sharp$ *css*) $\vee$ (*j* $\sharp$ *v* $\wedge$ (*j* $\sharp$ *css* $\vee$ *j* = *atom x*))

proof(*nominal-induct s* **and** *cs* **and** *css avoiding*: *v x rule*: *s-branch-s-branch-list.strong-induct*)

case (*AS-branch list x s*)
then show *?case* **using** *s-branch-s-branch-list.fresh fresh-Pair list.distinct*(*1*) *list.set-cases pure-fresh set-ConsD subst-branchv.simps* **by** *metis*
next
case (*AS-let y e s'*)
thus *?case* **proof**(*cases atom x* ♯ (*AS-let y e s'*))
case *True*
hence *subst-sv* (*AS-let y e s'*) *x v* = (*AS-let y e s'*) **using** *forget-subst-sv* **by** *simp*
hence *j* ♯ (*AS-let y e s'*) **using** *AS-let* **by** *argo*
then show *?thesis* **using** *True* **by** *blast*
next
case *False*

have *subst-sv* (*AS-let y e s'*) *x v* = *AS-let y* (*e*[*x*::=*v*]$_{ev}$) (*s'*[*x*::=*v*]$_{sv}$) **using** *subst-sv.simps*(*2*) *AS-let* **by** *force*
hence ((*j* ♯ *s'*[*x*::=*v*]$_{sv}$ ∨ *j* ∈ *set* [*atom y*]) ∧ *j* ♯ *None* ∧ *j* ♯ *e*[*x*::=*v*]$_{ev}$) **using** *s-branch-s-branch-list.fresh AS-let*
by (*simp add*: *fresh-None*)
then show *?thesis* **using** *AS-let fresh-None fresh-subst-ev-if list.discI list.set-cases s-branch-s-branch-list.fresh set-ConsD*
by *metis*
qed
next
case (*AS-let2 y τ s1 s2*)
thus *?case* **proof**(*cases atom x* ♯ (*AS-let2 y τ s1 s2*))
case *True*
hence *subst-sv* (*AS-let2 y τ s1 s2*) *x v* = (*AS-let2 y τ s1 s2*) **using** *forget-subst-sv* **by** *simp*
hence *j* ♯ (*AS-let2 y τ s1 s2*) **using** *AS-let2* **by** *argo*
then show *?thesis* **using** *True* **by** *blast*
next
case *False*
have *subst-sv* (*AS-let2 y τ s1 s2*) *x v* = *AS-let2 y* (*τ*[*x*::=*v*]$_{τv}$) (*s1*[*x*::=*v*]$_{sv}$) (*s2*[*x*::=*v*]$_{sv}$) **using** *subst-sv.simps AS-let2* **by** *force*
then show *?thesis* **using** *AS-let2*
fresh-subst-tv-if list.discI list.set-cases s-branch-s-branch-list.fresh(*4*) *set-ConsD* **by** *auto*
qed
qed(*auto*)+

lemma *fresh-subst-sv-if*[*simp*]:
fixes *x*::*x* **and** *v*::*v*
shows *j* ♯ (*subst-sv s x v*) ⟷ (*atom x* ♯ *s* ∧ *j* ♯ *s*) ∨ (*j* ♯ *v* ∧ (*j* ♯ *s* ∨ *j* = *atom x*)) **and**
j ♯ (*subst-branchv cs x v*) ⟷ (*atom x* ♯ *cs* ∧ *j* ♯ *cs*) ∨ (*j* ♯ *v* ∧ (*j* ♯ *cs* ∨ *j* = *atom x*))
using *fresh-subst-sv-if-lr fresh-subst-sv-if-rl* **by** *metis*+

lemma *subst-sv-commute* [*simp*]:
fixes *A*::*s* **and** *t*::*v* **and** *j*::*x* **and** *i*::*x*
shows *atom j* ♯ *A* ⟹ (*subst-sv* (*subst-sv A i t*) *j u*) = *subst-sv A i* (*subst-vv t j u*) **and**
atom j ♯ *B* ⟹ (*subst-branchv* (*subst-branchv B i t*) *j u*) = *subst-branchv B i* (*subst-vv t j u*)
and
atom j ♯ *C* ⟹ (*subst-branchlv* (*subst-branchlv C i t*) *j u*) = *subst-branchlv C i* (*subst-vv t j u*

)
apply(*nominal-induct A* **and** *B* **and** *C avoiding*: *i j t u rule*: *s-branch-s-branch-list.strong-induct*)
apply(*auto simp*: *fresh-at-base*)
done

lemma *c-eq-perm*:
assumes ((*atom z*) $\rightleftharpoons$ (*atom z'*)) $\cdot$ *c* = *c'* **and** *atom z'* $\sharp$ *c*
shows $\{\!|$ *z* : *b* | *c* $|\!\}$ = $\{\!|$ *z'* : *b* | *c'* $|\!\}$
using τ*.eq-iff Abs1-eq-iff*(*3*)
by (*metis Nominal2-Base.swap-commute assms*(*1*) *assms*(*2*) *flip-def swap-fresh-fresh*)

lemma *subst-sv-flip*:
fixes *s*::*s* **and** *sa*::*s* **and** *v'*::*v*
assumes *atom c* $\sharp$ (*s*, *sa*) **and** *atom c* $\sharp$ (*v'*,*x*, *xa*, *s*, *sa*) *atom x* $\sharp$ *v'* **and** *atom xa* $\sharp$ *v'* **and** (*x* $\leftrightarrow$ *c*) $\cdot$ *s* = (*xa* $\leftrightarrow$ *c*) $\cdot$ *sa*
shows *s*[*x*::=*v'*]$_{sv}$ = *sa*[*xa*::=*v'*]$_{sv}$
proof −
have *atom x* $\sharp$ (*s*[*x*::=*v'*]$_{sv}$) **and** *xafr*: *atom xa* $\sharp$ (*sa*[*xa*::=*v'*]$_{sv}$)
and *atom c* $\sharp$ (*s*[*x*::=*v'*]$_{sv}$, *sa*[*xa*::=*v'*]$_{sv}$) **using** *assms* **using** *fresh-subst-sv-if assms* **by**(*blast*+ ,*force*)

hence *s*[*x*::=*v'*]$_{sv}$ = (*x* $\leftrightarrow$ *c*) $\cdot$ (*s*[*x*::=*v'*]$_{sv}$) **by** (*simp add*: *flip-fresh-fresh fresh-Pair*)
also have ... = ((*x* $\leftrightarrow$ *c*) $\cdot$ *s*)[((*x* $\leftrightarrow$ *c*) $\cdot$ *x*) ::= ((*x* $\leftrightarrow$ *c*) $\cdot$ *v'*)]$_{sv}$ **using** *subst-sv-subst-branchv-subst-branchlv.eqvt* **by** *blast*
also have ... = ((*xa* $\leftrightarrow$ *c*) $\cdot$ *sa*)[((*x* $\leftrightarrow$ *c*) $\cdot$ *x*) ::= ((*x* $\leftrightarrow$ *c*) $\cdot$ *v'*)]$_{sv}$ **using** *assms* **by** *presburger*
also have ... = ((*xa* $\leftrightarrow$ *c*) $\cdot$ *sa*)[((*xa* $\leftrightarrow$ *c*) $\cdot$ *xa*) ::= ((*xa* $\leftrightarrow$ *c*) $\cdot$ *v'*)]$_{sv}$ **using** *assms*
by (*metis flip-at-simps*(*1*) *flip-fresh-fresh fresh-PairD*(*1*))
also have ... = (*xa* $\leftrightarrow$ *c*) $\cdot$ (*sa*[*xa*::=*v'*]$_{sv}$) **using** *subst-sv-subst-branchv-subst-branchlv.eqvt* **by** *presburger*
also have ... = *sa*[*xa*::=*v'*]$_{sv}$ **using** *xafr assms* **by** (*simp add*: *flip-fresh-fresh fresh-Pair*)
finally show *?thesis* **by** *simp*
qed

lemma *if-type-eq*:
fixes Γ::Γ **and** *v*::*v* **and** *z1*::*x*
assumes *atom z1'* $\sharp$ (*v*, *ca*, (*x*, *b*, *c*) $\#_{\Gamma}$ Γ, (*CE-val v* == *CE-val* (*V-lit ll*) *IMP* *ca*[*za*::=[*z1*]^{v}]$_{cv}$)) **and** *atom z1* $\sharp$ *v*
and *atom z1* $\sharp$ (*za*,*ca*) **and** *atom z1'* $\sharp$ (*za*,*ca*)
shows ($\{\!|$ *z1'* : *ba* | *CE-val v* == *CE-val* (*V-lit ll*) *IMP* *ca*[*za*::=[*z1'*]^{v}]$_{cv}$ $|\!\}$) = $\{\!|$ *z1* : *ba* | *CE-val v* == *CE-val* (*V-lit ll*) *IMP* *ca*[*za*::=[*z1*]^{v}]$_{cv}$ $|\!\}$
proof −
have *atom z1'* $\sharp$ (*CE-val v* == *CE-val* (*V-lit ll*) *IMP* *ca*[*za*::=[*z1*]^{v}]$_{cv}$) **using** *assms fresh-prod4* **by** *blast*
moreover hence (*CE-val v* == *CE-val* (*V-lit ll*) *IMP* *ca*[*za*::=[*z1'*]^{v}]$_{cv}$) = (*z1'* $\leftrightarrow$ *z1*) $\cdot$ (*CE-val v* == *CE-val* (*V-lit ll*) *IMP* *ca*[*za*::=[*z1*]^{v}]$_{cv}$)
proof −
have (*z1'* $\leftrightarrow$ *z1*) $\cdot$ (*CE-val v* == *CE-val* (*V-lit ll*) *IMP* *ca*[*za*::=[*z1*]^{v}]$_{cv}$) = ((*z1'* $\leftrightarrow$ *z1*) $\cdot$ (*CE-val v* == *CE-val* (*V-lit ll*)) *IMP* ((*z1'* $\leftrightarrow$ *z1*) $\cdot$ *ca*[*za*::=[*z1*]^{v}]$_{cv}$))
by *auto*
also have ... = ((*CE-val v* == *CE-val* (*V-lit ll*)) *IMP* ((*z1'* $\leftrightarrow$ *z1*) $\cdot$ *ca*[*za*::=[*z1*]^{v}]$_{cv}$))
using ‹*atom z1* $\sharp$ *v*› *assms*

by (*metis* (*mono-tags*) ‹*atom z1'* ♯ (*CE-val v* == *CE-val* (*V-lit ll*) *IMP ca*[*za*::=[*z1*]^v]$_{cv}$)› *c.fresh*(*6*) *c.fresh*(*7*) *ce.fresh*(*1*) *flip-at-simps*(*2*) *flip-fresh-fresh fresh-at-base-permute-iff fresh-def supp-l-empty v.fresh*(*1*))
also have ... = ((*CE-val v* == *CE-val* (*V-lit ll*)) *IMP* (*ca*[*za*::=[*z1'*]^v]$_{cv}$))
using *assms subst-cv-v-flip2* **by** *fastforce*
finally show *?thesis* **by** *auto*
qed
ultimately show *?thesis*
using *τ.eq-iff Abs1-eq-iff*(*3*)[*of z1' CE-val v* == *CE-val* (*V-lit ll*) *IMP ca*[*za*::=[*z1'*]^v]$_{cv}$ *z1 CE-val v* == *CE-val* (*V-lit ll*) *IMP ca*[*za*::=[*z1*]^v]$_{cv}$] **by** *blast*
qed

lemma *subst-sv-var-flip*:
fixes *x*::*x* **and** *s*::*s* **and** *z*::*x*
shows *atom x* ♯ *s* ⟹ ((*x* ↔ *z*) ∙ *s*) = *s*[*z*::=[*x*]^v]$_{sv}$ **and**
atom x ♯ *cs* ⟹ ((*x* ↔ *z*) ∙ *cs*) = *subst-branchv cs z* [*x*]^v **and**
atom x ♯ *css* ⟹ ((*x* ↔ *z*) ∙ *css*) = *subst-branchlv css z* [*x*]^v
apply(*nominal-induct s* **and** *cs* **and** *css avoiding*: *z x rule*: *s-branch-s-branch-list.strong-induct*)
using [[*simproc del*: *alpha-lst*]]
apply (*auto*)
using *subst-tv-var-flip flip-fresh-fresh v.fresh s-branch-s-branch-list.fresh*
subst-v-τ-def subst-v-v-def subst-vv-var-flip subst-v-e-def subst-ev-var-flip pure-fresh **apply** *auto*
defer *1*
using *x-fresh-u* **apply** *blast*
defer *1*
using *x-fresh-u* **apply** *blast*
defer *1*
using *x-fresh-u Abs1-eq-iff'*(*3*) *flip-fresh-fresh*
apply (*simp add*: *subst-v-c-def*)
using *x-fresh-u Abs1-eq-iff'*(*3*) *flip-fresh-fresh*
by (*simp add*: *flip-fresh-fresh*)

instantiation *s* :: *has-subst-v*
begin

definition
subst-v = *subst-sv*

instance proof
fix *j*::*atom* **and** *i*::*x* **and** *x*::*v* **and** *t*::*s*
show (*j* ♯ *subst-v t i x*) = ((*atom i* ♯ *t* ∧ *j* ♯ *t*) ∨ (*j* ♯ *x* ∧ (*j* ♯ *t* ∨ *j* = *atom i*)))
using *fresh-subst-sv-if subst-v-s-def* **by** *auto*

fix *a*::*x* **and** *tm*::*s* **and** *x*::*v*
show *atom a* ♯ *tm* ⟹ *subst-v tm a x* = *tm*
using *forget-subst-sv subst-v-s-def* **by** *simp*

fix *a*::*x* **and** *tm*::*s*
show *subst-v tm a* (*V-var a*) = *tm* **using** *subst-sv-id subst-v-s-def* **by** *simp*

fix *p*::*perm* **and** *x1*::*x* **and** *v*::*v* **and** *t1*::*s*
show *p* ∙ *subst-v t1 x1 v* = *subst-v* (*p* ∙ *t1*) (*p* ∙ *x1*) (*p* ∙ *v*)
using *subst-sv-commute subst-v-s-def* **by** *simp*

fix *x*::*x* **and** *c*::*s* **and** *z*::*x*
show *atom x* ♯ *c* ⟹ ((*x* ↔ *z*) ∙ *c*) = *c*[*z*::=[*x*]^v]$_v$
using *subst-sv-var-flip subst-v-s-def* **by** *simp*

fix *x*::*x* **and** *c*::*s* **and** *z*::*x*
show *atom x* ♯ *c* ⟹ ((*x* ↔ *z*) ∙ *c*)[*x*::=*v*]$_v$ = *c*[*z*::=*v*]$_v$
using *subst-sv-var-flip subst-v-s-def* **by** *simp*
qed
end

4.10 Type Definition

nominal-function *subst-ft-v* :: *fun-typ* ⇒ *x* ⇒ *v* ⇒ *fun-typ* **where**
atom z ♯ (*x*,*v*) ⟹ *subst-ft-v* (*AF-fun-typ z b c t* (*s*::*s*)) *x v* = *AF-fun-typ z b c*[*x*::=*v*]$_{cv}$ *t*[*x*::=*v*]$_{\tau v}$ *s*[*x*::=*v*]$_{sv}$
apply(*simp add*: *eqvt-def subst-ft-v-graph-aux-def*)
apply(*simp add*:*fun-typ.strong-exhaust*)
apply(*auto*)
apply(*rule-tac y*=*a* **and** *c*=(*aa*,*b*) **in** *fun-typ.strong-exhaust*)
apply (*auto simp*: *eqvt-at-def fresh-star-def fresh-Pair fresh-at-base*)
apply *blast*
proof(*goal-cases*)
case (*1 z c t s za xa va ca ta sa cb*)
hence *c*[*z*::=[*cb*]^v]$_{cv}$ = *ca*[*za*::=[*cb*]^v]$_{cv}$ **by** *metis*
hence *c*[*z*::=[*cb*]^v]$_{cv}$[*xa*::=*va*]$_{cv}$ = *ca*[*za*::=[*cb*]^v]$_{cv}$[*xa*::=*va*]$_{cv}$ **by** *auto*
then show *?case* **using** *subst-cv-commute atom-eq-iff fresh-atom fresh-atom-at-base subst-cv-commute-subst v.fresh*
using *1*(*14*) *1*(*2*) *1*(*3*) *1*(*4*) *1*(*5*) **by** *auto*
next
case (*2 z c t s za xa va ca ta sa cb*)
hence *t*[*z*::=[*cb*]^v]$_{\tau v}$ = *ta*[*za*::=[*cb*]^v]$_{\tau v}$ **by** *metis*
hence *t*[*z*::=[*cb*]^v]$_{\tau v}$[*xa*::=*va*]$_{\tau v}$ = *ta*[*za*::=[*cb*]^v]$_{\tau v}$[*xa*::=*va*]$_{\tau v}$ **by** *auto*
then show *?case* **using** *subst-tv-commute-subst 2*
by (*metis atom-eq-iff fresh-atom fresh-atom-at-base v.fresh*(*2*))

qed

nominal-termination (*eqvt*) **by** *lexicographic-order*

nominal-function *subst-ftq-v* :: *fun-typ-q* ⇒ *x* ⇒ *v* ⇒ *fun-typ-q* **where**
atom bv ♯ (*x*,*v*) ⟹ *subst-ftq-v* (*AF-fun-typ-some bv ft*) *x v* = (*AF-fun-typ-some bv* (*subst-ft-v ft x v*))

| *subst-ftq-v* (*AF-fun-typ-none ft*) *x v* = (*AF-fun-typ-none* (*subst-ft-v ft x v*))
apply(*simp add*: *eqvt-def subst-ftq-v-graph-aux-def*)
apply(*simp add*:*fun-typ-q.strong-exhaust*)
apply(*auto*)
apply(*rule-tac y*=*a* **and** *c*=(*aa*,*b*) **in** *fun-typ-q.strong-exhaust*)

```
    apply (auto simp: eqvt-at-def fresh-star-def fresh-Pair fresh-at-base)
proof(goal-cases)
  case (1 bv ft bva fta xa va c)
  then show ?case using subst-ft-v.simps  by (simp add: flip-fresh-fresh)
qed
nominal-termination (eqvt) by lexicographic-order



lemma size-subst-ft[simp]:  size (subst-ft-v A x v) = size A
  by(nominal-induct A  avoiding: x v rule: fun-typ.strong-induct,auto)

lemma forget-subst-ft [simp]: shows  atom x ♯ A ⟹ subst-ft-v A x a = A
  by (nominal-induct A  avoiding: a x rule: fun-typ.strong-induct,auto simp: fresh-at-base)

lemma subst-ft-id [simp]: subst-ft-v A a (V-var a)  = A
by(nominal-induct A  avoiding: a  rule: fun-typ.strong-induct,auto)


instantiation fun-typ :: has-subst-v
begin

definition
  subst-v = subst-ft-v

instance proof

  fix j::atom and i::x and  x::v and t::fun-typ
  show  (j ♯ subst-v t i x) = ((atom i ♯ t ∧ j ♯ t) ∨ (j ♯ x ∧ (j ♯ t ∨ j = atom i)))
  apply(nominal-induct t avoiding: i x rule:fun-typ.strong-induct)
    apply(simp only: subst-v-fun-typ-def subst-ft-v.simps )
    using fun-typ.fresh fresh-subst-v-if apply simp
      by auto

  fix a::x and tm::fun-typ and x::v
  show atom a ♯ tm ⟹ subst-v tm a x  = tm
  proof(nominal-induct tm avoiding: a x rule:fun-typ.strong-induct)
    case (AF-fun-typ x1a x2a x3a x4a x5a)
    then show ?case unfolding subst-ft-v.simps subst-v-fun-typ-def fun-typ.fresh  using forget-subst-v
subst-ft-v.simps subst-v-c-def forget-subst-sv subst-v-τ-def  by fastforce
  qed

  fix a::x and tm::fun-typ
  show subst-v tm a (V-var a) = tm
  proof(nominal-induct tm avoiding: a x rule:fun-typ.strong-induct)
    case (AF-fun-typ x1a x2a x3a x4a x5a)
    then show ?case unfolding subst-ft-v.simps subst-v-fun-typ-def fun-typ.fresh  using forget-subst-v
subst-ft-v.simps subst-v-c-def forget-subst-sv subst-v-τ-def  by fastforce
  qed

  fix p::perm and x1::x and v::v and t1::fun-typ
  show p • subst-v t1 x1 v  = subst-v  (p • t1) (p • x1) (p • v)
```

proof(*nominal-induct t1 avoiding*: *x1 v rule*:*fun-typ.strong-induct*)
case (*AF-fun-typ x1a x2a x3a x4a x5a*)
then show *?case* **unfolding** *subst-ft-v.simps subst-v-fun-typ-def fun-typ.fresh* **using** *forget-subst-v subst-ft-v.simps subst-v-c-def forget-subst-sv subst-v-τ-def* **by** *fastforce*
qed

fix *x*::*x* **and** *c*::*fun-typ* **and** *z*::*x*
show *atom x* ♯ *c* ⟹ ((*x* ↔ *z*) ∙ *c*) = *c*[*z*::=[*x*]^*v*]_*v*
apply(*nominal-induct c avoiding*: *x z rule*:*fun-typ.strong-induct*)
by (*auto simp add*: *subst-v-c-def subst-v-s-def subst-v-τ-def subst-v-fun-typ-def*)

fix *x*::*x* **and** *c*::*fun-typ* **and** *z*::*x*
show *atom x* ♯ *c* ⟹ ((*x* ↔ *z*) ∙ *c*)[*x*::=*v*]_*v* = *c*[*z*::=*v*]_*v*
apply(*nominal-induct c avoiding*: *z x v rule*:*fun-typ.strong-induct*)
apply *auto*
by (*auto simp add*: *subst-v-c-def subst-v-s-def subst-v-τ-def subst-v-fun-typ-def*)
qed
end

instantiation *fun-typ-q* :: *has-subst-v*
begin

definition
subst-v = *subst-ftq-v*

instance proof
fix *j*::*atom* **and** *i*::*x* **and** *x*::*v* **and** *t*::*fun-typ-q*
show (*j* ♯ *subst-v t i x*) = ((*atom i* ♯ *t* ∧ *j* ♯ *t*) ∨ (*j* ♯ *x* ∧ (*j* ♯ *t* ∨ *j* = *atom i*)))
apply(*nominal-induct t avoiding*: *i x rule*:*fun-typ-q.strong-induct*,*auto*)
apply(*auto simp add*: *subst-v-fun-typ-def subst-v-s-def subst-v-τ-def subst-v-fun-typ-q-def fresh-subst-v-if*)
by (*metis* (*no-types*) *fresh-subst-v-if subst-v-fun-typ-def*)+

fix *i*::*x* **and** *t*::*fun-typ-q* **and** *x*::*v*
show *atom i* ♯ *t* ⟹ *subst-v t i x* = *t*
apply(*nominal-induct t avoiding*: *i x rule*:*fun-typ-q.strong-induct*,*auto*)
by(*auto simp add*: *subst-v-fun-typ-def subst-v-s-def subst-v-τ-def subst-v-fun-typ-q-def fresh-subst-v-if*)

fix *i*::*x* **and** *t*::*fun-typ-q*
show *subst-v t i* (*V-var i*) = *t* **using** *subst-cv-id subst-v-fun-typ-def*
apply(*nominal-induct t avoiding*: *i x rule*:*fun-typ-q.strong-induct*,*auto*)
by(*auto simp add*: *subst-v-fun-typ-def subst-v-s-def subst-v-τ-def subst-v-fun-typ-q-def fresh-subst-v-if*)

fix *p*::*perm* **and** *x1*::*x* **and** *v*::*v* **and** *t1*::*fun-typ-q*
show *p* ∙ *subst-v t1 x1 v* = *subst-v* (*p* ∙ *t1*) (*p* ∙ *x1*) (*p* ∙ *v*)
apply(*nominal-induct t1 avoiding*: *v x1 rule*:*fun-typ-q.strong-induct*,*auto*)
by(*auto simp add*: *subst-v-fun-typ-def subst-v-s-def subst-v-τ-def subst-v-fun-typ-q-def fresh-subst-v-if*)

fix *x*::*x* **and** *c*::*fun-typ-q* **and** *z*::*x*

show *atom x* ♯ *c* ⟹ ((*x* ↔ *z*) ∙ *c*) = *c*[*z*::=[*x*]^v]$_v$
apply(*nominal-induct c avoiding*: *x z rule*:*fun-typ-q.strong-induct*,*auto*)
by(*auto simp add*: *subst-v-fun-typ-def subst-v-s-def subst-v-τ-def subst-v-fun-typ-q-def fresh-subst-v-if*
)

fix *x*::*x* **and** *c*::*fun-typ-q* **and** *z*::*x*
show *atom x* ♯ *c* ⟹ ((*x* ↔ *z*) ∙ *c*)[*x*::=*v*]$_v$ = *c*[*z*::=*v*]$_v$
apply(*nominal-induct c avoiding*: *z x v rule*:*fun-typ-q.strong-induct*,*auto*)
apply(*auto simp add*: *subst-v-fun-typ-def subst-v-s-def subst-v-τ-def subst-v-fun-typ-q-def fresh-subst-v-if*
)
by (*metis subst-v-fun-typ-def flip-bv-x-cancel subst-ft-v.eqvt subst-v-simple-commute v.perm-simps*
)+
qed

end

4.11 Variable Context

lemma *subst-dv-fst-eq*:
fst ‘ *setD* (Δ[*x*::=*v*]$_{\Delta v}$) = *fst* ‘ *setD* Δ
by(*induct* Δ *rule*: Δ-*induct*,*simp*,*force*)

lemma *subst-gv-member-iff*:
fixes *x'*::*x* **and** *x*::*x* **and** *v*::*v* **and** *c'*::*c*
assumes (*x'*,*b'*,*c'*) ∈ *setG* Γ **and** *atom x* ∉ *atom-dom* Γ
shows (*x'*,*b'*,*c'*[*x*::=*v*]$_{cv}$) ∈ *setG* Γ[*x*::=*v*]$_{\Gamma v}$
proof −
have *x'* ≠ *x* **using** *assms fresh-dom-free2* **by** *auto*
then show *?thesis* **using** *assms* **proof**(*induct* Γ *rule*: Γ-*induct*)
case *GNil*
then show *?case* **by** *auto*
next
case (*GCons x1 b1 c1* Γ')
show *?case* **proof**(*cases* (*x'*,*b'*,*c'*) = (*x1*,*b1*,*c1*))
case *True*
hence ((*x1*, *b1*, *c1*) #$_\Gamma$ Γ')[*x*::=*v*]$_{\Gamma v}$ = ((*x1*, *b1*, *c1*[*x*::=*v*]$_{cv}$) #$_\Gamma$ (Γ'[*x*::=*v*]$_{\Gamma v}$)) **using** *subst-gv.simps* ‹*x'*≠*x*› **by** *auto*
then show *?thesis* **using** *True* **by** *auto*
next
case *False*
have *x1*≠*x* **using** *fresh-def fresh-GCons fresh-Pair supp-at-base GCons fresh-dom-free2* **by** *auto*
hence (*x'*, *b'*, *c'*) ∈ *setG* Γ' **using** *GCons False setG.simps* **by** *auto*
moreover have *atom x* ∉ *atom-dom* Γ' **using** *fresh-GCons GCons dom.simps setG.simps* **by** *simp*
ultimately have (*x'*, *b'*, *c'*[*x*::=*v*]$_{cv}$) ∈ *setG* Γ'[*x*::=*v*]$_{\Gamma v}$ **using** *GCons* **by** *auto*
hence (*x'*, *b'*, *c'*[*x*::=*v*]$_{cv}$) ∈ *setG* ((*x1*, *b1*, *c1*[*x*::=*v*]$_{cv}$) #$_\Gamma$ (Γ'[*x*::=*v*]$_{\Gamma v}$)) **by** *auto*
then show *?thesis* **using** *subst-gv.simps* ‹*x1*≠*x*› **by** *auto*
qed
qed
qed

lemma *fresh-subst-gv-if*:
fixes *j*::*atom* **and** *i*::*x* **and** *x*::*v* **and** *t*::Γ

assumes $j \sharp t \wedge j \sharp x$
shows $(j \sharp \mathit{subst\text{-}gv}\ t\ i\ x)$
using *assms* **proof**(*induct t rule*: Γ-*induct*)
case *GNil*
then show *?case* **using** *subst-gv.simps fresh-GNil* **by** *auto*
next
case $(\mathit{GCons}\ x'\ b'\ c'\ \Gamma')$
then show *?case* **unfolding** *subst-gv.simps* **using** *fresh-GCons fresh-subst-cv-if* **by** *auto*
qed

4.12 Lookup

lemma *set-GConsD*: $y \in \mathit{setG}\ (x\ \#_\Gamma\ xs) \Longrightarrow y{=}x \vee y \in \mathit{setG}\ xs$
by *auto*

lemma *subst-g-assoc-cons*:
assumes $x \neq x'$
shows $(((x', b', c')\ \#_\Gamma\ \Gamma')[x::=v]_{\Gamma v}\ @\ G) = ((x', b', c'[x::=v]_{cv})\ \#_\Gamma\ ((\Gamma'[x::=v]_{\Gamma v})\ @\ G))$
using *subst-gv.simps append-g.simps assms* **by** *auto*

end

Chapter 5

Base Type Variable Substitution

5.1 Class

class *has-subst-b* = *fs* +
  fixes *subst-b* :: *'a*::*fs* ⇒ *bv* ⇒ *b* ⇒ *'a*::*fs* (-[-::=-]$_b$ [*1000*,*50*,*50*] *1000*)

  assumes *fresh-subst-if*: *j* ♯ (*t*[*i*::=*x*]$_b$) ⟷ (*atom i* ♯ *t* ∧ *j* ♯ *t*) ∨ (*j* ♯ *x* ∧ (*j* ♯ *t* ∨ *j* = *atom i*))
   and *forget-subst*[*simp*]: *atom a* ♯ *tm* ⟹ *tm*[*a*::=*x*]$_b$ = *tm*
   and *subst-id*[*simp*]: *tm*[*a*::=(*B-var a*)]$_b$ = *tm*
   and *eqvt*[*simp*,*eqvt*]: (*p*::*perm*) ∙ (*subst-b t1 x1 v*) = (*subst-b* (*p* ∙*t1*) (*p* ∙*x1*) (*p* ∙*v*))
   and *flip-subst*[*simp*]: *atom bv* ♯ *c* ⟹ ((*bv* ↔ *z*) ∙ *c*) = *c*[*z*::=*B-var bv*]$_b$
   and *flip-subst-subst*[*simp*]: *atom bv* ♯ *c* ⟹ ((*bv* ↔ *z*) ∙ *c*)[*bv*::=*v*]$_b$ = *c*[*z*::=*v*]$_b$
begin

lemmas *flip-subst-b* = *flip-subst-subst*

lemma *subst-b-simple-commute*:
  fixes *x*::*bv*
  assumes *atom x* ♯ *c*
  shows (*c*[*z*::=*B-var x*]$_b$)[*x*::=*b*]$_b$ = *c*[*z*::=*b*]$_b$
proof −
  have (*c*[*z*::=*B-var x*]$_b$)[*x*::=*b*]$_b$ = ((*x* ↔ *z*) ∙ *c*)[*x*::=*b*]$_b$ **using** *flip-subst assms* **by** *simp*
  thus *?thesis* **using** *flip-subst-subst assms* **by** *simp*
qed

lemma *subst-b-flip-eq-one*:
  fixes *z1*::*bv* **and** *z2*::*bv* **and** *x1*::*bv* **and** *x2*::*bv*
  assumes [[*atom z1*]]*lst*. *c1* = [[*atom z2*]]*lst*. *c2*
     and *atom x1* ♯ (*z1*,*z2*,*c1*,*c2*)
    shows (*c1*[*z1*::=*B-var x1*]$_b$) = (*c2*[*z2*::=*B-var x1*]$_b$)
proof −
  have (*c1*[*z1*::=*B-var x1*]$_b$) = (*x1* ↔ *z1*) ∙ *c1* **using** *assms flip-subst* **by** *auto*
  moreover have (*c2*[*z2*::=*B-var x1*]$_b$) = (*x1* ↔ *z2*) ∙ *c2* **using** *assms flip-subst* **by** *auto*
  ultimately show *?thesis* **using** *Abs1-eq-iff-all*(*3*)[*of z1 c1 z2 c2 z1*] *assms*
    by (*metis Abs1-eq-iff-fresh*(*3*) *flip-commute*)
qed

lemma *subst-b-flip-eq-two*:
fixes *z1*::*bv* **and** *z2*::*bv* **and** *x1*::*bv* **and** *x2*::*bv*
assumes [[*atom z1*]]*lst*. *c1* = [[*atom z2*]]*lst*. *c2*
shows (*c1*[*z1*::=*b*]$_b$) = (*c2*[*z2*::=*b*]$_b$)
proof −
obtain *x*::*bv* **where** ∗:*atom x* ♯ (*z1*,*z2*,*c1*,*c2*) **using** *obtain-fresh* **by** *metis*
hence (*c1*[*z1*::=*B-var x*]$_b$) = (*c2*[*z2*::=*B-var x*]$_b$) **using** *subst-b-flip-eq-one*[*OF assms*, *of x*] **by** *metis*
hence (*c1*[*z1*::=*B-var x*]$_b$)[*x*::=*b*]$_b$ = (*c2*[*z2*::=*B-var x*]$_b$)[*x*::=*b*]$_b$ **by** *auto*
thus *?thesis* **using** *subst-b-simple-commute* ∗ *fresh-prod4* **by** *metis*
qed

lemma *subst-b-fresh-x*:
fixes *tm*::′*a*::*fs* **and** *x*::*x*
shows *atom x* ♯ *tm* = *atom x* ♯ *tm*[*bv*::=*b*′]$_b$
using *fresh-subst-if*[*of atom x tm bv b*′] **using** *x-fresh-b* **by** *auto*

lemma *subst-b-x-flip*[*simp*]:
fixes *x*′::*x* **and** *x*::*x* **and** *bv*::*bv*
shows ((*x*′ ↔ *x*) · *tm*)[*bv*::=*b*′]$_b$ = (*x*′ ↔ *x*) · (*tm*[*bv*::=*b*′]$_b$)
proof −
have (*x*′ ↔ *x*) · *bv* = *bv* **using** *pure-supp flip-fresh-fresh* **by** *force*
moreover have (*x*′ ↔ *x*) · *b*′ = *b*′ **using** *x-fresh-b flip-fresh-fresh* **by** *auto*
ultimately show *?thesis* **using** *eqvt* **by** *simp*
qed

end

5.2 Base Type

nominal-function *subst-bb* :: *b* ⇒ *bv* ⇒ *b* ⇒ *b* **where**
subst-bb (*B-var bv2*) *bv1 b* = (*if bv1* = *bv2 then b else* (*B-var bv2*))
| *subst-bb B-int bv1 b* = *B-int*
| *subst-bb B-bool bv1 b* = *B-bool*
| *subst-bb* (*B-id s*) *bv1 b* = *B-id s*
| *subst-bb* (*B-pair b1 b2*) *bv1 b* = *B-pair* (*subst-bb b1 bv1 b*) (*subst-bb b2 bv1 b*)
| *subst-bb B-unit bv1 b* = *B-unit*
| *subst-bb B-bitvec bv1 b* = *B-bitvec*
| *subst-bb* (*B-app s b2*) *bv1 b* = *B-app s* (*subst-bb b2 bv1 b*)
apply (*simp add*: *eqvt-def subst-bb-graph-aux-def*)
apply (*simp add*: *eqvt-def subst-bb-graph-aux-def*)
apply *auto*
apply (*meson b.strong-exhaust*)
done
nominal-termination (*eqvt*) **by** *lexicographic-order*

abbreviation
subst-bb-abbrev :: *b* ⇒ *bv* ⇒ *b* ⇒ *b* (-[-::=-]$_{bb}$ [*1000*,*50*,*50*] *1000*)
where

$b[bv::=b']_{bb} \equiv$ *subst-bb b bv b'*

instantiation *b* :: *has-subst-b*
begin
definition *subst-b* = *subst-bb*

instance proof
fix *j*::*atom* **and** *i*::*bv* **and** *x*::*b* **and** *t*::*b*
show *j* ♯ *subst-b t i x* = (*atom i* ♯ *t* ∧ *j* ♯ *t* ∨ *j* ♯ *x* ∧ (*j* ♯ *t* ∨ *j* = *atom i*))
proof (*induct t rule*: *b.induct*)
case (*B-id x*)
then show *?case* **using** *subst-bb.simps fresh-def pure-fresh subst-b-b-def* **by** *auto*
next
case (*B-var x*)
then show *?case* **using** *subst-bb.simps fresh-def pure-fresh subst-b-b-def* **by** *auto*
next
case (*B-app x1 x2*)
then show *?case* **using** *subst-bb.simps fresh-def pure-fresh subst-b-b-def* **by** *auto*
qed(*auto simp add*: *subst-bb.simps fresh-def pure-fresh subst-b-b-def*)+

fix *a*::*bv* **and** *tm*::*b* **and** *x*::*b*
show *atom a* ♯ *tm* ⟹ *tm*[*a*::=*x*]$_b$ = *tm*
by (*induct tm rule*: *b.induct*, *auto simp add*: *fresh-at-base subst-bb.simps subst-b-b-def*)

fix *a*::*bv* **and** *tm*::*b*
show *subst-b tm a* (*B-var a*) = *tm* **using** *subst-bb.simps subst-b-b-def*
by (*induct tm rule*: *b.induct*, *auto simp add*: *fresh-at-base subst-bb.simps subst-b-b-def*)

fix *p*::*perm* **and** *x1*::*bv* **and** *v*::*b* **and** *t1*::*b*
show *p* ∙ *subst-b t1 x1 v* = *subst-b* (*p* ∙ *t1*) (*p* ∙ *x1*) (*p* ∙ *v*)
by (*induct tm rule*: *b.induct*, *auto simp add*: *fresh-at-base subst-bb.simps subst-b-b-def*)

fix *bv*::*bv* **and** *c*::*b* **and** *z*::*bv*
show *atom bv* ♯ *c* ⟹ ((*bv* ↔ *z*) ∙ *c*) = *c*[*z*::=*B-var bv*]$_b$
by (*induct c rule*: *b.induct*, (*auto simp add*: *fresh-at-base subst-bb.simps subst-b-b-def permute-pure pure-supp*)+)

fix *bv*::*bv* **and** *c*::*b* **and** *z*::*bv* **and** *v*::*b*
show *atom bv* ♯ *c* ⟹ ((*bv* ↔ *z*) ∙ *c*)[*bv*::=*v*]$_b$ = *c*[*z*::=*v*]$_b$
by (*induct c rule*: *b.induct*, (*auto simp add*: *fresh-at-base subst-bb.simps subst-b-b-def permute-pure pure-supp*)+)

qed
end

lemma *subst-bb-inject*:
assumes *b1* = *b2*[*bv*::=*b*]$_{bb}$ **and** *b2* ≠ *B-var bv*
shows
b1 = *B-int* ⟹ *b2* = *B-int* **and**
b1 = *B-bool* ⟹ *b2* = *B-bool* **and**

$b1 = B\text{-}unit \Longrightarrow b2 = B\text{-}unit$ **and**
$b1 = B\text{-}bitvec \Longrightarrow b2 = B\text{-}bitvec$ **and**
$b1 = B\text{-}pair\ b11\ b12 \Longrightarrow (\exists\, b11'\ b12'.\ b11 = b11'[bv::=b]_{bb} \wedge b12 = b12'[bv::=b]_{bb} \wedge b2 = B\text{-}pair\ b11'\ b12')$ **and**
$b1 = B\text{-}var\ bv' \Longrightarrow b2 = B\text{-}var\ bv'$ **and**
$b1 = B\text{-}id\ tyid \Longrightarrow b2 = B\text{-}id\ tyid$ **and**
$b1 = B\text{-}app\ tyid\ b11 \Longrightarrow (\exists\, b11'.\ b11 = b11'[bv::=b]_{bb} \wedge b2 = B\text{-}app\ tyid\ b11')$
using *assms* **by**(*nominal-induct b2 rule*:*b.strong-induct*,*auto*+)

lemma *flip-b-subst4*:
fixes *b1*::*b* **and** *bv1*::*bv* **and** *c*::*bv* **and** *b*::*b*
assumes *atom c* ♯ (*b1*,*bv1*)
shows $b1[bv1::=b]_{bb} = ((bv1 \leftrightarrow c) \cdot b1)[c ::= b]_{bb}$
using *assms* **proof**(*nominal-induct b1 rule*: *b.strong-induct*)
case *B-int*
then show *?case* **using** *subst-bb.simps b.perm-simps* **by** *auto*
next
case *B-bool*
then show *?case* **using** *subst-bb.simps b.perm-simps* **by** *auto*
next
case (*B-id x*)
hence *atom bv1* ♯ *x* ∧ *atom c* ♯ *x* **using** *fresh-def pure-supp* **by** *auto*
hence $((bv1 \leftrightarrow c) \cdot B\text{-}id\ x) = B\text{-}id\ x$ **using** *fresh-Pair b.fresh*(*3*) *flip-fresh-fresh b.perm-simps fresh-def pure-supp* **by** *metis*
then show *?case* **using** *subst-bb.simps* **by** *simp*
next
case (*B-pair x1 x2*)
hence $x1[bv1::=b]_{bb} = ((bv1 \leftrightarrow c) \cdot x1)[c::=b]_{bb}$ **using** *b.perm-simps*(*4*) *b.fresh*(*4*) *fresh-Pair* **by** *metis*
moreover have $x2[bv1::=b]_{bb} = ((bv1 \leftrightarrow c) \cdot x2)[c::=b]_{bb}$ **using** *b.perm-simps*(*4*) *b.fresh*(*4*) *fresh-Pair B-pair* **by** *metis*
ultimately show *?case* **using** *subst-bb.simps*(*5*) *b.perm-simps*(*4*) *b.fresh*(*4*) *fresh-Pair* **by** *auto*
next
case *B-unit*
then show *?case* **using** *subst-bb.simps b.perm-simps* **by** *auto*
next
case *B-bitvec*
then show *?case* **using** *subst-bb.simps b.perm-simps* **by** *auto*
next
case (*B-var x*)
then show *?case* **proof**(*cases x=bv1*)
case *True*
then show *?thesis* **using** *B-var subst-bb.simps b.perm-simps* **by** *simp*
next
case *False*
moreover have $x \neq c$ **using** *B-var b.fresh fresh-def supp-at-base fresh-Pair* **by** *fastforce*
ultimately show *?thesis* **using** *B-var subst-bb.simps*(*1*) *b.perm-simps*(*7*) **by** *simp*
qed
next
case (*B-app x1 x2*)
hence $x2[bv1::=b]_{bb} = ((bv1 \leftrightarrow c) \cdot x2)[c::=b]_{bb}$ **using** *b.perm-simps b.fresh fresh-Pair* **by** *metis*
thus *?case* **using** *subst-bb.simps b.perm-simps b.fresh fresh-Pair B-app*

by (*simp add*: *permute-pure*)
qed

lemma *subst-bb-flip-sym*:
fixes *b1*::*b* **and** *b2*::*b*
assumes *atom c* ♯ *b* **and** *atom c* ♯ (*bv1*,*bv2*, *b1*, *b2*) **and** (*bv1* ↔ *c*) ∙ *b1* = (*bv2* ↔ *c*) ∙ *b2*
shows *b1*[*bv1*::=*b*]$_{bb}$ = *b2*[*bv2*::=*b*]$_{bb}$
using *assms* *flip-b-subst4*[*of c b1 bv1 b*] *flip-b-subst4*[*of c b2 bv2 b*] *fresh-prod4 fresh-Pair* **by** *simp*

5.3 Value

nominal-function *subst-vb* :: *v* ⇒ *bv* ⇒ *b* ⇒ *v* **where**
subst-vb (*V-lit l*) *x v* = *V-lit l*
| *subst-vb* (*V-var y*) *x v* = *V-var y*
| *subst-vb* (*V-cons tyid c v'*) *x v* = *V-cons tyid c* (*subst-vb v' x v*)
| *subst-vb* (*V-consp tyid c b v'*) *x v* = *V-consp tyid c* (*subst-bb b x v*) (*subst-vb v' x v*)
| *subst-vb* (*V-pair v1 v2*) *x v* = *V-pair* (*subst-vb v1 x v*) (*subst-vb v2 x v*)
apply (*simp add*: *eqvt-def subst-vb-graph-aux-def*)
apply *auto*
using *v.strong-exhaust* **by** *meson*
nominal-termination (*eqvt*) **by** *lexicographic-order*

abbreviation
subst-vb-abbrev :: *v* ⇒ *bv* ⇒ *b* ⇒ *v* (-[-::=-]$_{vb}$ [*1000*,*50*,*50*] *500*)
where
e[*bv*::=*b*]$_{vb}$ ≡ *subst-vb e bv b*

instantiation *v* :: *has-subst-b*
begin
definition *subst-b* = *subst-vb*

instance proof
fix *j*::*atom* **and** *i*::*bv* **and** *x*::*b* **and** *t*::*v*
show *j* ♯ *subst-b t i x* = (*atom i* ♯ *t* ∧ *j* ♯ *t* ∨ *j* ♯ *x* ∧ (*j* ♯ *t* ∨ *j* = *atom i*))
proof (*induct t rule*: *v.induct*)
case (*V-lit l*)
have *j* ♯ *subst-b* (*V-lit l*) *i x* = *j* ♯ (*V-lit l*) **using** *subst-vb.simps fresh-def pure-fresh*
subst-b-v-def v.supp *v.fresh has-subst-b-class.fresh-subst-if subst-b-b-def subst-b-v-def* **by** *auto*
also have ... = *True* **using** *fresh-at-base v.fresh l.fresh supp-l-empty fresh-def* **by** *metis*
moreover have (*atom i* ♯ (*V-lit l*) ∧ *j* ♯ (*V-lit l*) ∨ *j* ♯ *x* ∧ (*j* ♯ (*V-lit l*) ∨ *j* = *atom i*)) = *True*
using *fresh-at-base v.fresh l.fresh supp-l-empty fresh-def* **by** *metis*
ultimately show *?case* **by** *simp*
next
case (*V-var y*)
then show *?case* **using** *subst-b-v-def subst-vb.simps pure-fresh* **by** *force*
next
case (*V-pair x1a x2a*)
then show *?case* **using** *subst-b-v-def subst-vb.simps* **by** *auto*
next

```
    case (V-cons x1a x2a x3)
    then show ?case using V-cons subst-b-v-def subst-vb.simps pure-fresh by force
  next
    case (V-consp x1a x2a x3 x4)
     then show ?case using  subst-b-v-def subst-vb.simps pure-fresh  has-subst-b-class.fresh-subst-if
subst-b-b-def subst-b-v-def by fastforce
  qed

  fix a::bv and tm::v and x::b
  show atom a ♯ tm ⟹ subst-b tm a x = tm
    apply(induct tm rule: v.induct)
    apply( auto simp add: fresh-at-base subst-vb.simps subst-b-v-def)
    using has-subst-b-class.fresh-subst-if  subst-b-b-def e.fresh
    using has-subst-b-class.forget-subst by fastforce

  fix a::bv and tm::v
  show subst-b tm a (B-var a) = tm using subst-bb.simps  subst-b-b-def
    apply (induct tm rule: v.induct)
          apply(auto simp add: fresh-at-base subst-vb.simps subst-b-v-def)
 using has-subst-b-class.fresh-subst-if  subst-b-b-def e.fresh
    using has-subst-b-class.subst-id by metis

  fix p::perm and x1::bv and v::b and t1::v
  show p ∙ subst-b t1 x1 v  = subst-b  (p ∙ t1) (p ∙ x1) (p ∙ v)
    apply(induct tm rule: v.induct)
          apply( auto simp add: fresh-at-base subst-bb.simps subst-b-b-def )
   using has-subst-b-class.eqvt  subst-b-b-def e.fresh
    using has-subst-b-class.eqvt
    by (simp add: subst-b-v-def)+

  fix  bv::bv and c::v and z::bv
  show atom bv ♯ c ⟹ ((bv ↔ z) ∙ c) = c[z::=B-var bv]_b
   apply (induct c rule: v.induct, (auto simp add: fresh-at-base  subst-vb.simps subst-b-v-def permute-pure
pure-supp )+)

      apply (metis flip-fresh-fresh flip-l-eq permute-flip-cancel2)
    using  fresh-at-base flip-fresh-fresh[of bv  x z]
     apply (simp add: flip-fresh-fresh)
    using subst-b-b-def by argo

  fix bv::bv and c::v and z::bv and v::b
  show atom bv ♯ c ⟹ ((bv ↔ z) ∙ c)[bv::=v]_b = c[z::=v]_b
   apply (induct c rule: v.induct, (auto simp add: fresh-at-base  subst-vb.simps subst-b-v-def permute-pure
pure-supp )+)
      apply (metis flip-fresh-fresh flip-l-eq permute-flip-cancel2)
    using  fresh-at-base flip-fresh-fresh[of bv  x z]
     apply (simp add: flip-fresh-fresh)
    using     subst-b-b-def  flip-subst-subst by fastforce

qed
end
```

5.4 Constraints Expressions

nominal-function *subst-ceb* :: *ce* ⇒ *bv* ⇒ *b* ⇒ *ce* **where**
subst-ceb ((*CE-val v'*)) *bv b* = (*CE-val* (*subst-vb v' bv b*))
| *subst-ceb* ((*CE-op opp v1 v2*)) *bv b* = ((*CE-op opp* (*subst-ceb v1 bv b*)(*subst-ceb v2 bv b*)))
| *subst-ceb* ((*CE-fst v'*)) *bv b* = *CE-fst* (*subst-ceb v' bv b*)
| *subst-ceb* ((*CE-snd v'*)) *bv b* = *CE-snd* (*subst-ceb v' bv b*)
| *subst-ceb* ((*CE-len v'*)) *bv b* = *CE-len* (*subst-ceb v' bv b*)
| *subst-ceb* (*CE-concat v1 v2*) *bv b* = *CE-concat* (*subst-ceb v1 bv b*) (*subst-ceb v2 bv b*)
apply (*simp add*: *eqvt-def subst-ceb-graph-aux-def*)
apply *auto*
by (*meson ce.strong-exhaust*)
nominal-termination (*eqvt*) **by** *lexicographic-order*

abbreviation
subst-ceb-abbrev :: *ce* ⇒ *bv* ⇒ *b* ⇒ *ce* (-[-::=-]$_{ceb}$ [*1000*,*50*,*50*] *500*)
where
ce[*bv*::=*b*]$_{ceb}$ ≡ *subst-ceb ce bv b*

instantiation *ce* :: *has-subst-b*
begin
definition *subst-b* = *subst-ceb*

instance proof
fix *j*::*atom* **and** *i*::*bv* **and** *x*::*b* **and** *t*::*ce*
show *j* ♯ *subst-b t i x* = (*atom i* ♯ *t* ∧ *j* ♯ *t* ∨ *j* ♯ *x* ∧ (*j* ♯ *t* ∨ *j* = *atom i*))
proof (*induct t rule*: *ce.induct*)
case (*CE-val v*)
then show *?case* **using** *subst-ceb.simps fresh-def pure-fresh subst-b-ce-def ce.supp v.supp ce.fresh has-subst-b-class.fresh-subst-if subst-b-b-def subst-b-v-def*
by *metis*
next
case (*CE-op opp v1 v2*)

have (*j* ♯ *v1*[*i*::=*x*]$_{ceb}$ ∧ *j* ♯ *v2*[*i*::=*x*]$_{ceb}$) = ((*atom i* ♯ *v1* ∧ *atom i* ♯ *v2*) ∧ *j* ♯ *v1* ∧ *j* ♯ *v2* ∨ *j* ♯ *x* ∧ (*j* ♯ *v1* ∧ *j* ♯ *v2* ∨ *j* = *atom i*))
using *has-subst-b-class.fresh-subst-if subst-b-v-def*
using *CE-op.hyps*(*1*) *CE-op.hyps*(*2*) *subst-b-ce-def* **by** *auto*

thus *?case* **unfolding** *subst-ceb.simps subst-b-ce-def ce.fresh*
using *fresh-def pure-fresh opp.fresh subst-b-v-def opp.exhaust fresh-e-opp-all*
by (*metis* (*full-types*))
next
case (*CE-concat x1a x2*)
then show *?case* **using** *subst-ceb.simps subst-b-ce-def e.supp v.supp has-subst-b-class.fresh-subst-if subst-b-v-def ce.fresh* **by** *force*
next
case (*CE-fst x*)
then show *?case* **using** *subst-ceb.simps subst-b-ce-def e.supp v.supp has-subst-b-class.fresh-subst-if subst-b-v-def ce.fresh* **by** *metis*

next
case (*CE-snd x*)

```
   then show ?case using subst-ceb.simps  subst-b-ce-def e.supp v.supp  has-subst-b-class.fresh-subst-if
subst-b-v-def  ce.fresh by metis
  next
   case (CE-len x)
   then show ?case using subst-ceb.simps  subst-b-ce-def e.supp v.supp  has-subst-b-class.fresh-subst-if
subst-b-v-def  ce.fresh by metis
  qed

  fix a::bv and tm::ce and x::b
  show atom a ♯ tm ⟹ subst-b tm a x = tm
   apply(induct tm rule: ce.induct)
   apply( auto simp add: fresh-at-base subst-ceb.simps subst-b-ce-def)
   using has-subst-b-class.fresh-subst-if  subst-b-b-def e.fresh
    using has-subst-b-class.forget-subst subst-b-v-def apply metis+
   done

  fix a::bv and tm::ce
  show subst-b tm a (B-var a) = tm using subst-bb.simps  subst-b-b-def
   apply (induct tm rule: ce.induct)
   apply(auto simp add: fresh-at-base subst-ceb.simps subst-b-ce-def)
   using has-subst-b-class.fresh-subst-if  subst-b-b-def e.fresh
     using has-subst-b-class.subst-id subst-b-v-def apply metis+
  done

  fix p::perm and x1::bv and v::b and t1::ce
  show p ∙ subst-b t1 x1 v  = subst-b  (p ∙ t1) (p ∙ x1) (p ∙ v)
   apply(induct tm rule: ce.induct)
   apply( auto simp add: fresh-at-base subst-bb.simps subst-b-b-def  )
   using has-subst-b-class.eqvt  subst-b-b-def ce.fresh
    using has-subst-b-class.eqvt
    by (simp add: subst-b-ce-def)+


  fix  bv::bv and c::ce and z::bv
  show atom bv ♯ c ⟹ ((bv ↔ z) ∙ c) = c[z::=B-var bv]_b
     apply (induct c rule: ce.induct,  (auto simp add: fresh-at-base  subst-ceb.simps subst-b-ce-def
permute-pure pure-supp )+)

    using  flip-fresh-fresh flip-l-eq permute-flip-cancel2 has-subst-b-class.flip-subst subst-b-v-def apply
metis
using  flip-fresh-fresh flip-l-eq permute-flip-cancel2 has-subst-b-class.flip-subst subst-b-v-def
      apply (metis opp.perm-simps(2) opp.strong-exhaust)+
  done

  fix bv::bv and c::ce and z::bv and v::b
  show atom bv ♯ c ⟹ ((bv ↔ z) ∙ c)[bv::=v]_b = c[z::=v]_b
proof (induct c rule: ce.induct)
  case (CE-val x)
 then show ?case using flip-subst-subst subst-b-v-def subst-ceb.simps  using subst-b-ce-def by fastforce
next
  case (CE-op x1a x2 x3)
 then show ?case unfolding subst-ceb.simps subst-b-ce-def ce.perm-simps using flip-subst-subst subst-b-v-def
```

opp.perm-simps opp.strong-exhaust
  by (*metis* (*full-types*) *ce.fresh*(*2*))
next
 case (*CE-concat x1a x2*)
 then show *?case* **using** *flip-subst-subst subst-b-v-def subst-ceb.simps* **using** *subst-b-ce-def* **by** *fastforce*
next
 case (*CE-fst x*)
 then show *?case* **using** *flip-subst-subst subst-b-v-def subst-ceb.simps* **using** *subst-b-ce-def* **by** *fastforce*
next
 case (*CE-snd x*)
 then show *?case* **using** *flip-subst-subst subst-b-v-def subst-ceb.simps* **using** *subst-b-ce-def* **by** *fastforce*
next
 case (*CE-len x*)
 then show *?case* **using** *flip-subst-subst subst-b-v-def subst-ceb.simps* **using** *subst-b-ce-def* **by** *fastforce*
qed

qed
end

5.5 Constraints

nominal-function *subst-cb* :: $c \Rightarrow bv \Rightarrow b \Rightarrow c$ **where**
  subst-cb (*C-true*) *x v* = *C-true*
| *subst-cb* (*C-false*) *x v* = *C-false*
| *subst-cb* (*C-conj c1 c2*) *x v* = *C-conj* (*subst-cb c1 x v*) (*subst-cb c2 x v*)
| *subst-cb* (*C-disj c1 c2*) *x v* = *C-disj* (*subst-cb c1 x v*) (*subst-cb c2 x v*)
| *subst-cb* (*C-imp c1 c2*) *x v* = *C-imp* (*subst-cb c1 x v*) (*subst-cb c2 x v*)
| *subst-cb* (*C-eq e1 e2*) *x v* = *C-eq* (*subst-ceb e1 x v*) (*subst-ceb e2 x v*)
| *subst-cb* (*C-not c*) *x v* = *C-not* (*subst-cb c x v*)
apply (*simp add*: *eqvt-def subst-cb-graph-aux-def*)
apply *auto*
using *c.strong-exhaust* **apply** *metis*
done
nominal-termination (*eqvt*) **by** *lexicographic-order*

abbreviation
  subst-cb-abbrev :: $c \Rightarrow bv \Rightarrow b \Rightarrow c$ (-[-::=-]$_{cb}$ [*1000*,*50*,*50*] *500*)
where
  $c[bv::=b]_{cb} \equiv$ *subst-cb c bv b*

instantiation *c* :: *has-subst-b*
begin
definition *subst-b* = *subst-cb*

instance proof
  fix *j*::*atom* **and** *i*::*bv* **and** *x*::*b* **and** *t*::*c*
  show $j \sharp$ *subst-b t i x* = (*atom i* $\sharp$ *t* $\wedge$ *j* $\sharp$ *t* $\vee$ *j* $\sharp$ *x* $\wedge$ (*j* $\sharp$ *t* $\vee$ *j* = *atom i*))
    by (*induct t rule*: *c.induct*, *unfold subst-cb.simps subst-b-c-def c.fresh*,
      (*metis has-subst-b-class.fresh-subst-if subst-b-ce-def c.fresh*)+
)

fix *a*::*bv* **and** *tm*::*c* **and** *x*::*b*
show *atom a* ♯ *tm* ⟹ *subst-b tm a x* = *tm*
by(*induct tm rule*: *c.induct*, *unfold subst-cb.simps subst-b-c-def c.fresh*,
(*metis has-subst-b-class.forget-subst subst-b-ce-def*)+)

fix *a*::*bv* **and** *tm*::*c*
show *subst-b tm a* (*B-var a*) = *tm* **using** *subst-bb.simps subst-b-c-def*
by(*induct tm rule*: *c.induct*, *unfold subst-cb.simps subst-b-c-def c.fresh*,
(*metis has-subst-b-class.subst-id subst-b-ce-def*)+)

fix *p*::*perm* **and** *x1*::*bv* **and** *v*::*b* **and** *t1*::*c*
show *p* ∙ *subst-b t1 x1 v* = *subst-b* (*p* ∙ *t1*) (*p* ∙ *x1*) (*p* ∙ *v*)
apply(*induct tm rule*: *c.induct*,*unfold subst-cb.simps subst-b-c-def c.fresh*)
by(*auto simp add*: *fresh-at-base subst-bb.simps subst-b-b-def*)

fix *bv*::*bv* **and** *c*::*c* **and** *z*::*bv*
show *atom bv* ♯ *c* ⟹ ((*bv* ↔ *z*) ∙ *c*) = *c*[*z*::=*B-var bv*]$_b$
apply (*induct c rule*: *c.induct*, (*auto simp add*: *fresh-at-base subst-cb.simps subst-b-c-def permute-pure pure-supp*)+)
using *flip-fresh-fresh flip-l-eq permute-flip-cancel2 has-subst-b-class.flip-subst subst-b-ce-def* **apply** *metis*
using *flip-fresh-fresh flip-l-eq permute-flip-cancel2 has-subst-b-class.flip-subst subst-b-ce-def*
apply (*metis opp.perm-simps*(2) *opp.strong-exhaust*)+
done

fix *bv*::*bv* **and** *c*::*c* **and** *z*::*bv* **and** *v*::*b*
show *atom bv* ♯ *c* ⟹ ((*bv* ↔ *z*) ∙ *c*)[*bv*::=*v*]$_b$ = *c*[*z*::=*v*]$_b$
apply (*induct c rule*: *c.induct*, (*auto simp add*: *fresh-at-base subst-cb.simps subst-b-c-def permute-pure pure-supp*)+)

using *flip-fresh-fresh flip-l-eq permute-flip-cancel2 has-subst-b-class.flip-subst subst-b-ce-def*
using *flip-subst-subst* **apply** *fastforce*
using *flip-fresh-fresh flip-l-eq permute-flip-cancel2 has-subst-b-class.flip-subst subst-b-ce-def*
opp.perm-simps(2) *opp.strong-exhaust*
proof −
fix *x1a* :: *ce* **and** *x2* :: *ce*
assume *a1*: *atom bv* ♯ *x2*
then have ((*bv* ↔ *z*) ∙ *x2*)[*bv*::=*v*]$_b$ = *x2*[*z*::=*v*]$_b$
by (*metis flip-subst-subst*)
then show *x2*[*z*::=*B-var bv*]$_b$[*bv*::=*v*]$_{ceb}$ = *x2*[*z*::=*v*]$_{ceb}$
using *a1* **by** (*simp add*: *subst-b-ce-def*)
qed

qed
end

5.6 Types

nominal-function *subst-tb* :: *τ* ⇒ *bv* ⇒ *b* ⇒ *τ* **where**
subst-tb (⦃ *z* : *b2* | *c* ⦄) *bv1 b1* = ⦃ *z* : *b2*[*bv1*::=*b1*]$_{bb}$ | *c*[*bv1*::=*b1*]$_{cb}$ ⦄
proof(*goal-cases*)
case *1*

then show *?case* **using** *eqvt-def subst-tb-graph-aux-def* **by** *force*
next
case (*2 x y*)
then show *?case* **by** *auto*
next
case (*3 P x*)
then show *?case* **using** *eqvt-def subst-tb-graph-aux-def τ.strong-exhaust*
by (*metis b-of.cases prod-cases3*)
next
case (*4 z' b2' c' bv1' b1' z b2 c bv1 b1*)
show *?case* **unfolding** *τ.eq-iff* **proof**
have ∗:[[*atom z'*]]*lst. c'* = [[*atom z*]]*lst. c* **using** *τ.eq-iff 4* **by** *auto*
show [[*atom z'*]]*lst. c'*[*bv1'*::=*b1'*]$_{cb}$ = [[*atom z*]]*lst. c*[*bv1*::=*b1*]$_{cb}$ **proof**(*subst Abs1-eq-iff-all*(*3*),*rule*,*rule*,*rule*)
fix *ca*::*x*
assume *atom ca* ♯ *z* **and** *1*:*atom ca* ♯ (*z'*, *z*, *c'*[*bv1'*::=*b1'*]$_{cb}$, *c*[*bv1*::=*b1*]$_{cb}$)
hence *2*:*atom ca* ♯ (*c'*,*c*) **using** *fresh-subst-if subst-b-c-def fresh-Pair fresh-prod4 fresh-at-base subst-b-fresh-x* **by** *metis*
hence (*z'* ↔ *ca*) ∙ *c'* = (*z* ↔ *ca*) ∙ *c* **using** *1 2* ∗ *Abs1-eq-iff-all*(*3*) **by** *auto*
hence ((*z'* ↔ *ca*) ∙ *c'*)[*bv1'*::=*b1'*]$_{cb}$ = ((*z* ↔ *ca*) ∙ *c*)[*bv1'*::=*b1'*]$_{cb}$ **by** *auto*
hence (*z'* ↔ *ca*) ∙ *c'*[(*z'* ↔ *ca*) ∙ *bv1'*::=(*z'* ↔ *ca*) ∙ *b1'*]$_{cb}$ = (*z* ↔ *ca*) ∙ *c*[(*z* ↔ *ca*) ∙ *bv1'*::=(*z* ↔ *ca*) ∙ *b1'*]$_{cb}$ **by** *auto*
thus (*z'* ↔ *ca*) ∙ *c'*[*bv1'*::=*b1'*]$_{cb}$ = (*z* ↔ *ca*) ∙ *c*[*bv1*::=*b1*]$_{cb}$ **using** *4 flip-x-b-cancel* **by** *simp*
qed
show *b2'*[*bv1'*::=*b1'*]$_{bb}$ = *b2*[*bv1*::=*b1*]$_{bb}$ **using** *4* **by** *simp*
qed
qed

nominal-termination (*eqvt*) **by** *lexicographic-order*

abbreviation
subst-tb-abbrev :: *τ* ⇒ *bv* ⇒ *b* ⇒ *τ* (-[-::=-]$_{\tau b}$ [*1000*,*50*,*50*] *1000*)
where
t[*bv*::=*b'*]$_{\tau b}$ ≡ *subst-tb t bv b'*

instantiation *τ* :: *has-subst-b*
begin
definition *subst-b* = *subst-tb*

instance proof
fix *j*::*atom* **and** *i*::*bv* **and** *x*::*b* **and** *t*::*τ*
show *j* ♯ *subst-b t i x* = (*atom i* ♯ *t* ∧ *j* ♯ *t* ∨ *j* ♯ *x* ∧ (*j* ♯ *t* ∨ *j* = *atom i*))
proof (*nominal-induct t avoiding*: *i x j rule*: *τ.strong-induct*)
case (*T-refined-type z b c*)
then show *?case*
unfolding *subst-b-τ-def subst-tb.simps τ.fresh*
using *fresh-subst-if*[*of j b i x*] *subst-b-b-def subst-b-c-def*
by (*metis has-subst-b-class.fresh-subst-if list.distinct*(*1*) *list.set-cases not-self-fresh set-ConsD*)
qed

fix *a*::*bv* **and** *tm*::*τ* **and** *x*::*b*

show *atom a ♯ tm ⟹ subst-b tm a x = tm*
proof (*nominal-induct tm avoiding*: *a x rule*: *τ.strong-induct*)
case (*T-refined-type xx bb cc*)
moreover hence *atom a ♯ bb ∧ atom a ♯ cc* **using** *τ.fresh* **by** *auto*
ultimately show *?case*
unfolding *subst-b-τ-def subst-tb.simps*
using *forget-subst subst-b-b-def subst-b-c-def forget-subst τ.fresh* **by** *metis*
qed

fix *a*::*bv* **and** *tm*::*τ*
show *subst-b tm a* (*B-var a*) = *tm*
proof (*nominal-induct tm rule*: *τ.strong-induct*)
case (*T-refined-type xx bb cc*)
thus *?case*
unfolding *subst-b-τ-def subst-tb.simps*
using *subst-id subst-b-b-def subst-b-c-def* **by** *metis*
qed

fix *p*::*perm* **and** *x1*::*bv* **and** *v*::*b* **and** *t1*::*τ*
show *p* ∙ *subst-b t1 x1 v* = *subst-b* (*p* ∙ *t1*) (*p* ∙ *x1*) (*p* ∙ *v*)
by (*induct tm rule*: *τ.induct*, *auto simp add*: *fresh-at-base subst-tb.simps subst-b-τ-def subst-bb.simps subst-b-b-def*)

fix *bv*::*bv* **and** *c*::*τ* **and** *z*::*bv*
show *atom bv ♯ c ⟹* ((*bv* ↔ *z*) ∙ *c*) = *c*[*z*::=*B-var bv*]$_b$
apply (*induct c rule*: *τ.induct*, (*auto simp add*: *fresh-at-base subst-ceb.simps subst-b-ce-def permute-pure pure-supp*)+)
using *flip-fresh-fresh permute-flip-cancel2 has-subst-b-class.flip-subst subst-b-c-def subst-b-b-def*
by (*simp add*: *flip-fresh-fresh subst-b-τ-def*)

fix *bv*::*bv* **and** *c*::*τ* **and** *z*::*bv* **and** *v*::*b*
show *atom bv ♯ c ⟹* ((*bv* ↔ *z*) ∙ *c*)[*bv*::=*v*]$_b$ = *c*[*z*::=*v*]$_b$
proof (*induct c rule*: *τ.induct*)
case (*T-refined-type x1a x2a x3a*)
hence *atom bv ♯ x2a ∧ atom bv ♯ x3a ∧ atom bv ♯ x1a* **using** *fresh-at-base τ.fresh* **by** *simp*
then show *?case*
unfolding *subst-tb.simps subst-b-τ-def τ.perm-simps*
using *fresh-at-base flip-fresh-fresh*[*of bv x1a z*] *flip-subst-subst subst-b-b-def subst-b-c-def T-refined-type*

proof −
have *atom z ♯ x1a*
by (*metis b.fresh*(7) *fresh-at-base*(2) *x-fresh-b*)
then show {‖ (*bv* ↔ *z*) ∙ *x1a* : ((*bv* ↔ *z*) ∙ *x2a*)[*bv*::=*v*]$_{bb}$ | ((*bv* ↔ *z*) ∙ *x3a*)[*bv*::=*v*]$_{cb}$ ‖} = {‖ *x1a* : *x2a*[*z*::=*v*]$_{bb}$ | *x3a*[*z*::=*v*]$_{cb}$ ‖}
by (*metis* ‹⟦*atom bv ♯ x1a*; *atom z ♯ x1a*⟧ ⟹ (*bv* ↔ *z*) ∙ *x1a* = *x1a*› ‹*atom bv ♯ x2a ∧ atom bv ♯ x3a ∧ atom bv ♯ x1a*› *flip-subst-subst subst-b-b-def subst-b-c-def*)
qed
qed

qed
end

lemma *subst-bb-commute* [*simp*]:
atom j ♯ *A* ⟹ (*subst-bb* (*subst-bb A i t*) *j u*) = *subst-bb A i* (*subst-bb t j u*)
by (*nominal-induct A avoiding*: *i j t u rule*: *b.strong-induct*) (*auto simp*: *fresh-at-base*)

lemma *subst-vb-commute* [*simp*]:
atom j ♯ *A* ⟹ (*subst-vb* (*subst-vb A i t*)) *j u* = *subst-vb A i* (*subst-bb t j u*)
by (*nominal-induct A avoiding*: *i j t u rule*: *v.strong-induct*) (*auto simp*: *fresh-at-base*)

lemma *subst-ceb-commute* [*simp*]:
atom j ♯ *A* ⟹ (*subst-ceb* (*subst-ceb A i t*)) *j u* = *subst-ceb A i* (*subst-bb t j u*)
by (*nominal-induct A avoiding*: *i j t u rule*: *ce.strong-induct*) (*auto simp*: *fresh-at-base*)

lemma *subst-cb-commute* [*simp*]:
atom j ♯ *A* ⟹ (*subst-cb* (*subst-cb A i t*)) *j u* = *subst-cb A i* (*subst-bb t j u*)
by (*nominal-induct A avoiding*: *i j t u rule*: *c.strong-induct*) (*auto simp*: *fresh-at-base*)

lemma *subst-tb-commute* [*simp*]:
atom j ♯ *A* ⟹ (*subst-tb* (*subst-tb A i t*)) *j u* = *subst-tb A i* (*subst-bb t j u*)
proof (*nominal-induct A avoiding*: *i j t u rule*: *τ.strong-induct*)
case (*T-refined-type z b c*)
then show *?case* **using** *subst-tb.simps subst-bb-commute subst-cb-commute* **by** *simp*
qed

5.7 Expressions

nominal-function *subst-eb* :: *e* ⇒ *bv* ⇒ *b* ⇒ *e* **where**
subst-eb ((*AE-val v'*)) *bv b* = (*AE-val* (*subst-vb v' bv b*))
| *subst-eb* ((*AE-app f v'*)) *bv b* = ((*AE-app f* (*subst-vb v' bv b*)))
| *subst-eb* ((*AE-appP f b' v'*)) *bv b* = ((*AE-appP f* (*b'*[*bv*::=*b*]$_{bb}$) (*subst-vb v' bv b*)))
| *subst-eb* ((*AE-op opp v1 v2*)) *bv b* = ((*AE-op opp* (*subst-vb v1 bv b*) (*subst-vb v2 bv b*)))
| *subst-eb* ((*AE-fst v'*)) *bv b* = *AE-fst* (*subst-vb v' bv b*)
| *subst-eb* ((*AE-snd v'*)) *bv b* = *AE-snd* (*subst-vb v' bv b*)
| *subst-eb* ((*AE-mvar u*)) *bv b* = *AE-mvar u*
| *subst-eb* ((*AE-len v'*)) *bv b* = *AE-len* (*subst-vb v' bv b*)
| *subst-eb* (*AE-concat v1 v2*) *bv b* = *AE-concat* (*subst-vb v1 bv b*) (*subst-vb v2 bv b*)
| *subst-eb* (*AE-split v1 v2*) *bv b* = *AE-split* (*subst-vb v1 bv b*) (*subst-vb v2 bv b*)
apply (*simp add*: *eqvt-def subst-eb-graph-aux-def*)
apply *auto*
by (*meson e.strong-exhaust*)
nominal-termination (*eqvt*) **by** *lexicographic-order*

abbreviation
subst-eb-abbrev :: *e* ⇒ *bv* ⇒ *b* ⇒ *e* (-[-::=-]$_{eb}$ [*1000*,*50*,*50*] *500*)
where
e[*bv*::=*b*]$_{eb}$ ≡ *subst-eb e bv b*

instantiation *e* :: *has-subst-b*
begin

definition *subst-b* = *subst-eb*

instance proof
fix *j*::*atom* **and** *i*::*bv* **and** *x*::*b* **and** *t*::*e*
show *j* ♯ *subst-b t i x* = (*atom i* ♯ *t* ∧ *j* ♯ *t* ∨ *j* ♯ *x* ∧ (*j* ♯ *t* ∨ *j* = *atom i*))
proof (*induct t rule*: *e.induct*)
case (*AE-val v*)
then show *?case* **using** *subst-eb.simps fresh-def pure-fresh subst-b-e-def e.supp v.supp*
e.fresh has-subst-b-class.fresh-subst-if subst-b-e-def subst-b-v-def
by *metis*
next
case (*AE-app f v*)
then show *?case* **using** *subst-eb.simps fresh-def pure-fresh subst-b-e-def*
e.supp v.supp has-subst-b-class.fresh-subst-if subst-b-v-def
by (*metis* (*mono-tags*, *hide-lams*) *e.fresh*(*2*))
next
case (*AE-appP f b′ v*)
then show *?case* **unfolding** *subst-eb.simps subst-b-e-def e.fresh* **using**
fresh-def pure-fresh subst-b-e-def e.supp v.supp
e.fresh has-subst-b-class.fresh-subst-if subst-b-b-def subst-vb-def **by** (*metis subst-b-v-def*)
next
case (*AE-op opp v1 v2*)
then show *?case* **unfolding** *subst-eb.simps subst-b-e-def e.fresh* **using**
fresh-def pure-fresh subst-b-e-def e.supp v.supp fresh-e-opp-all
e.fresh has-subst-b-class.fresh-subst-if subst-b-b-def subst-vb-def **by** (*metis subst-b-v-def*)

next
case (*AE-concat x1a x2*)
then show *?case* **using** *subst-eb.simps fresh-def pure-fresh subst-b-e-def e.supp v.supp*
has-subst-b-class.fresh-subst-if subst-b-v-def
by (*metis subst-vb.simps*(*5*))
next
case (*AE-split x1a x2*)
then show *?case* **using** *subst-eb.simps fresh-def pure-fresh subst-b-e-def e.supp v.supp*
has-subst-b-class.fresh-subst-if subst-b-v-def
by (*metis subst-vb.simps*(*5*))
next
case (*AE-fst x*)
then show *?case* **using** *subst-eb.simps fresh-def pure-fresh subst-b-e-def e.supp v.supp has-subst-b-class.fresh-subst-if*
subst-b-v-def **by** *metis*
next
case (*AE-snd x*)
then show *?case* **using** *subst-eb.simps fresh-def pure-fresh subst-b-e-def e.supp v.supp* **using** *has-subst-b-class.fresh-sub*
subst-b-v-def **by** *metis*
next
case (*AE-mvar x*)
then show *?case* **using** *subst-eb.simps fresh-def pure-fresh subst-b-e-def e.supp v.supp* **by** *auto*
next
case (*AE-len x*)
then show *?case* **using** *subst-eb.simps fresh-def pure-fresh subst-b-e-def e.supp v.supp* **using** *has-subst-b-class.fresh-su*
subst-b-v-def **by** *metis*
qed

```
  fix a::bv and tm::e and x::b
  show atom a ♯ tm ⟹ subst-b tm a x = tm
    apply(induct tm rule: e.induct)
    apply( auto simp add: fresh-at-base subst-eb.simps subst-b-e-def)
    using has-subst-b-class.fresh-subst-if  subst-b-b-def e.fresh
    using has-subst-b-class.forget-subst subst-b-v-def apply metis+
    done

  fix a::bv and tm::e
  show subst-b tm a (B-var a) = tm using subst-bb.simps  subst-b-b-def
    apply (induct tm rule: e.induct)
    apply(auto simp add: fresh-at-base subst-eb.simps subst-b-e-def)
    using has-subst-b-class.fresh-subst-if  subst-b-b-def e.fresh
    using has-subst-b-class.subst-id subst-b-v-def apply metis+
  done

  fix p::perm and x1::bv and v::b and t1::e
  show p • subst-b t1 x1 v  = subst-b  (p • t1) (p • x1) (p • v)
    apply(induct tm rule: e.induct)
   apply( auto simp add: fresh-at-base subst-bb.simps subst-b-b-def  )
   using has-subst-b-class.eqvt  subst-b-b-def e.fresh
    using has-subst-b-class.eqvt
    by (simp add: subst-b-e-def)+

  fix  bv::bv and c::e and z::bv
  show atom bv ♯ c ⟹ ((bv ↔ z) • c) = c[z::=B-var bv]_b
    apply (induct c rule: e.induct)
   apply(auto simp add: fresh-at-base  subst-eb.simps subst-b-e-def subst-b-v-def  permute-pure pure-supp
)
    using  flip-fresh-fresh permute-flip-cancel2 has-subst-b-class.flip-subst subst-b-v-def  subst-b-b-def
     flip-fresh-fresh subst-b-τ-def apply metis
    apply (metis (full-types) opp.perm-simps(1) opp.perm-simps(2) opp.strong-exhaust)
    done

  fix bv::bv and c::e and z::bv and v::b
  show atom bv ♯ c ⟹ ((bv ↔ z) • c)[bv::=v]_b = c[z::=v]_b
    apply (induct c rule: e.induct)
   apply(auto simp add: fresh-at-base  subst-eb.simps subst-b-e-def subst-b-v-def  permute-pure pure-supp
)
    using  flip-fresh-fresh permute-flip-cancel2 has-subst-b-class.flip-subst subst-b-v-def  subst-b-b-def
     flip-fresh-fresh subst-b-τ-def apply simp

    apply (metis opp.perm-simps(1) opp.perm-simps(2) opp.strong-exhaust)
    done
qed
end
```

5.8 Statements

nominal-function (*default case-sum* (λ*x*. *Inl undefined*) (*case-sum* (λ*x*. *Inl undefined*) (λ*x*. *Inr undefined*)))

subst-sb :: $s \Rightarrow bv \Rightarrow b \Rightarrow s$
and *subst-branchb* :: *branch-s* $\Rightarrow bv \Rightarrow b \Rightarrow$ *branch-s*
and *subst-branchlb* :: *branch-list* $\Rightarrow bv \Rightarrow b \Rightarrow$ *branch-list*
where
  subst-sb (*AS-val v'*) *bv b* = (*AS-val* (*subst-vb v' bv b*))
| *subst-sb* (*AS-let y e s*) *bv b* = (*AS-let y* ($e[bv::=b]_{eb}$) (*subst-sb s bv b*))
| *subst-sb* (*AS-let2 y t s1 s2*) *bv b* = (*AS-let2 y* (*subst-tb t bv b*) (*subst-sb s1 bv b*) (*subst-sb s2 bv b*))
| *subst-sb* (*AS-match v' cs*) *bv b* = *AS-match* (*subst-vb v' bv b*) (*subst-branchlb cs bv b*)
| *subst-sb* (*AS-assign y v'*) *bv b* = *AS-assign y* (*subst-vb v' bv b*)
| *subst-sb* (*AS-if v' s1 s2*) *bv b* = (*AS-if* (*subst-vb v' bv b*) (*subst-sb s1 bv b*) (*subst-sb s2 bv b*))
| *subst-sb* (*AS-var u* τ *v' s*) *bv b* = *AS-var u* (*subst-tb* τ *bv b*) (*subst-vb v' bv b*) (*subst-sb s bv b*)
| *subst-sb* (*AS-while s1 s2*) *bv b* = *AS-while* (*subst-sb s1 bv b*) (*subst-sb s2 bv b*)
| *subst-sb* (*AS-seq s1 s2*) *bv b* = *AS-seq* (*subst-sb s1 bv b*) (*subst-sb s2 bv b*)
| *subst-sb* (*AS-assert c s*) *bv b* = *AS-assert* (*subst-cb c bv b*) (*subst-sb s bv b*)

| *subst-branchb* (*AS-branch dc x1 s'*) *bv b* = *AS-branch dc x1* (*subst-sb s' bv b*)

| *subst-branchlb* (*AS-final sb*) *bv b* = *AS-final* (*subst-branchb sb bv b*)
| *subst-branchlb* (*AS-cons sb ssb*) *bv b* = *AS-cons* (*subst-branchb sb bv b*) (*subst-branchlb ssb bv b*)

apply (*simp add*: *eqvt-def subst-sb-subst-branchb-subst-branchlb-graph-aux-def*)

apply (*auto*,*metis s-branch-s-branch-list.exhaust s-branch-s-branch-list.exhaust*(*2*) *old.sum.exhaust surj-pair*)

proof(*goal-cases*)

have *eqvt-at-proj*: $\bigwedge$ *s xa va . eqvt-at subst-sb-subst-branchb-subst-branchlb-sumC* (*Inl* (*s*, *xa*, *va*)) $\Longrightarrow$
  eqvt-at (λ*a. projl* (*subst-sb-subst-branchb-subst-branchlb-sumC* (*Inl a*))) (*s*, *xa*, *va*)
 apply(*simp only*: *eqvt-at-def*)
 apply(*rule*)
 apply(*subst Projl-permute*)
  apply(*thin-tac* -)+
  apply(*simp add*: *subst-sb-subst-branchb-subst-branchlb-sumC-def*)
  apply(*simp add*: *THE-default-def*)
  apply(*case-tac Ex1* (*subst-sb-subst-branchb-subst-branchlb-graph* (*Inl* (*s*,*xa*,*va*))))
  apply *simp*
  apply(*auto*)[*1*]
  apply(*erule-tac x*=*x* **in** *allE*)
  apply *simp*
  apply(*cases rule*: *subst-sb-subst-branchb-subst-branchlb-graph.cases*)
  apply(*assumption*)
 apply(*rule-tac x*=*Sum-Type.projl x* **in** *exI*,*clarify*,*rule the1-equality*,*blast*,*simp* (*no-asm*) *only*: *sum.sel*)+
  apply *blast* +

  apply(*simp*)+
 done
{
 case (*1 y s ya sa bva ba c*)
 moreover have *atom y* $\sharp$ (*bva*, *ba*) $\wedge$ *atom ya* $\sharp$ (*bva*, *ba*) **using** *x-fresh-b x-fresh-bv fresh-Pair* **by** *simp*
 ultimately show *?case*

using *eqvt-triple eqvt-at-proj* **by** *metis*
next
case (*2 y s2 ya s1a s2a bva ba c*)
moreover have *atom y* ♯ (*bva*, *ba*) ∧ *atom ya* ♯ (*bva*, *ba*) **using** *x-fresh-b x-fresh-bv fresh-Pair* **by** *simp*
ultimately show *?case*
using *eqvt-triple eqvt-at-proj* **by** *metis*
next
case (*3 u s ua sa bva ba c*)
moreover have *atom u* ♯ (*bva*, *ba*) ∧ *atom ua* ♯ (*bva*, *ba*) **using** *x-fresh-b x-fresh-bv fresh-Pair* **by** *simp*
ultimately show *?case* **using** *eqvt-triple eqvt-at-proj* **by** *metis*
next
case (*4 x1 s′ x1a s′a bva ba c*)
moreover have *atom x1* ♯ (*bva*, *ba*) ∧ *atom x1a* ♯ (*bva*, *ba*) **using** *x-fresh-b x-fresh-bv fresh-Pair* **by** *simp*
ultimately show *?case* **using** *eqvt-triple eqvt-at-proj* **by** *metis*
}
qed

nominal-termination (*eqvt*) **by** *lexicographic-order*

abbreviation
subst-sb-abbrev :: *s* ⇒ *bv* ⇒ *b* ⇒ *s* (-[-::=-]$_{sb}$ [*1000*,*50*,*50*] *1000*)
where
b[*bv*::=*b′*]$_{sb}$ ≡ *subst-sb b bv b′*

lemma *fresh-subst-sb-if* [*simp*]:
(*j* ♯ (*subst-sb A i x*)) = ((*atom i* ♯ *A* ∧ *j* ♯ *A*) ∨ (*j* ♯ *x* ∧ (*j* ♯ *A* ∨ *j* = *atom i*))) **and**
(*j* ♯ (*subst-branchb B i x*)) = ((*atom i* ♯ *B* ∧ *j* ♯ *B*) ∨ (*j* ♯ *x* ∧ (*j* ♯ *B* ∨ *j* = *atom i*))) **and**
(*j* ♯ (*subst-branchlb C i x*)) = ((*atom i* ♯ *C* ∧ *j* ♯ *C*) ∨ (*j* ♯ *x* ∧ (*j* ♯ *C* ∨ *j* = *atom i*)))
proof (*nominal-induct A* **and** *B* **and** *C avoiding*: *i x rule*: *s-branch-s-branch-list.strong-induct*)
case (*AS-branch x1 x2 x3*)
have (*j* ♯ *subst-branchb* (*AS-branch x1 x2 x3*) *i x*) = (*j* ♯ (*AS-branch x1 x2* (*subst-sb x3 i x*))) **by** *auto*
also have ... = ((*j* ♯ *x3*[*i*::=*x*]$_{sb}$ ∨ *j* ∈ *set* [*atom x2*]) ∧ *j* ♯ *x1*) **using** *s-branch-s-branch-list.fresh* **by** *auto*
also have ... = ((*atom i* ♯ *AS-branch x1 x2 x3* ∧ *j* ♯ *AS-branch x1 x2 x3*) ∨ *j* ♯ *x* ∧ (*j* ♯ *AS-branch x1 x2 x3* ∨ *j* = *atom i*))
using *subst-branchb.simps*(*1*) *s-branch-s-branch-list.fresh*(*1*) *fresh-at-base has-subst-b-class.fresh-subst-if list.distinct list.set-cases set-ConsD subst-b-τ-def*
v.fresh AS-branch
proof −
have *f1*: ∀ *cs b*. *atom* (*b*::*bv*) ♯ (*cs*::*char list*) **using** *pure-fresh* **by** *auto*

then have *j* ♯ *x* ∧ *atom i* = *j* ⟶ ((*j* ♯ *x3*[*i*::=*x*]$_{sb}$ ∨ *j* ∈ *set* [*atom x2*]) ∧ *j* ♯ *x1*) = (*atom i* ♯ *AS-branch x1 x2 x3* ∧ *j* ♯ *AS-branch x1 x2 x3* ∨ *j* ♯ *x* ∧ (*j* ♯ *AS-branch x1 x2 x3* ∨ *j* = *atom i*))
by (*metis* (*full-types*) *AS-branch.hyps*(*3*))
then have *j* ♯ *x* ⟶ ((*j* ♯ *x3*[*i*::=*x*]$_{sb}$ ∨ *j* ∈ *set* [*atom x2*]) ∧ *j* ♯ *x1*) = (*atom i* ♯ *AS-branch x1 x2 x3* ∧ *j* ♯ *AS-branch x1 x2 x3* ∨ *j* ♯ *x* ∧ (*j* ♯ *AS-branch x1 x2 x3* ∨ *j* = *atom i*))
using *AS-branch.hyps s-branch-s-branch-list.fresh* **by** *metis*
moreover

```
    { assume ¬ j ♯ x
      have ?thesis
        using f1 AS-branch.hyps(2) AS-branch.hyps(3) by force }
    ultimately show ?thesis
      by satx
  qed
 finally show ?case by auto

next
 case (AS-cons cs css i x)
 show ?case
   unfolding subst-branchlb.simps s-branch-s-branch-list.fresh
   using AS-cons by auto
next
 case (AS-val xx)
 then show ?case using subst-sb.simps(1) s-branch-s-branch-list.fresh has-subst-b-class.fresh-subst-if
subst-b-b-def subst-b-v-def by metis
next
 case (AS-let x1 x2 x3)
 then show ?case using subst-sb.simps s-branch-s-branch-list.fresh fresh-at-base has-subst-b-class.fresh-subst-if
list.distinct list.set-cases set-ConsD subst-b-e-def
   by fastforce
next
 case (AS-let2 x1 x2 x3 x4)
 then show ?case using subst-sb.simps s-branch-s-branch-list.fresh fresh-at-base has-subst-b-class.fresh-subst-if
list.distinct list.set-cases set-ConsD subst-b-τ-def
   by fastforce
next
 case (AS-if x1 x2 x3)
 then show ?case unfolding subst-sb.simps s-branch-s-branch-list.fresh using
  has-subst-b-class.fresh-subst-if subst-b-v-def by metis
next
 case (AS-var u t v s)

   have (((atom i ♯ s ∧ j ♯ s ∨ j ♯ x ∧ (j ♯ s ∨ j = atom i)) ∨ j ∈ set [atom u]) ∧ j ♯ t[i::=x]τb ∧ j
♯ v[i::=x]vb) =
        (((atom i ♯ s ∧ j ♯ s ∨ j ♯ x ∧ (j ♯ s ∨ j = atom i)) ∨ j ∈ set [atom u]) ∧
              ((atom i ♯ t ∧ j ♯ t ∨ j ♯ x ∧ (j ♯ t ∨ j = atom i))) ∧
              ((atom i ♯ v ∧ j ♯ v ∨ j ♯ x ∧ (j ♯ v ∨ j = atom i))))
     using has-subst-b-class.fresh-subst-if subst-b-v-def subst-b-τ-def by metis
   also have ... = (((atom i ♯ s ∨ atom i ∈ set [atom u]) ∧ atom i ♯ t ∧ atom i ♯ v) ∧
           (j ♯ s ∨ j ∈ set [atom u]) ∧ j ♯ t ∧ j ♯ v ∨ j ♯ x ∧ ((j ♯ s ∨ j ∈ set [atom u]) ∧ j ♯ t ∧ j
♯ v ∨ j = atom i))
     using u-fresh-b by auto
   finally show ?case using subst-sb.simps s-branch-s-branch-list.fresh AS-var
     by simp

next
 case (AS-assign u v )
 then show ?case unfolding subst-sb.simps s-branch-s-branch-list.fresh using
  has-subst-b-class.fresh-subst-if subst-b-v-def by force
next
```

case (*AS-match v cs*)
have *j* ♯ (*AS-match v cs*)[*i*::=*x*]$_{sb}$ = *j* ♯ (*AS-match* (*subst-vb v i x*) (*subst-branchlb cs i x*)) **using** *subst-sb.simps* **by** *auto*
also have ... = (*j* ♯ (*subst-vb v i x*) ∧ *j* ♯ (*subst-branchlb cs i x*)) **using** *s-branch-s-branch-list.fresh* **by** *simp*
also have ... = (*j* ♯ (*subst-vb v i x*) ∧ ((*atom i* ♯ *cs* ∧ *j* ♯ *cs*) ∨ *j* ♯ *x* ∧ (*j* ♯ *cs* ∨ *j* = *atom i*))) **using** *AS-match*[*of i x*] **by** *auto*
also have ... = (*atom i* ♯ *AS-match v cs* ∧ *j* ♯ *AS-match v cs* ∨ *j* ♯ *x* ∧ (*j* ♯ *AS-match v cs* ∨ *j* = *atom i*))
by (*metis* (*no-types*) *s-branch-s-branch-list.fresh has-subst-b-class.fresh-subst-if subst-b-v-def*)
finally show *?case* **by** *auto*

next
case (*AS-while x1 x2*)
then show *?case* **by** *auto*
next
case (*AS-seq x1 x2*)
then show *?case* **by** *auto*
next
case (*AS-assert x1 x2*)
then show *?case* **unfolding** *subst-sb.simps s-branch-s-branch-list.fresh*
using *fresh-at-base has-subst-b-class.fresh-subst-if list.distinct list.set-cases set-ConsD subst-b-e-def*
by (*metis subst-b-c-def*)

qed(*auto*+)

lemma
forget-subst-sb[*simp*]: *atom a* ♯ *A* ⟹ *subst-sb A a x* = *A* **and**
forget-subst-branchb [*simp*]: *atom a* ♯ *B* ⟹ *subst-branchb B a x* = *B* **and**
forget-subst-branchlb[*simp*]: *atom a* ♯ *C* ⟹ *subst-branchlb C a x* = *C*
proof (*nominal-induct A* **and** *B* **and** *C avoiding*: *a x rule*: *s-branch-s-branch-list.strong-induct*)
case (*AS-let x1 x2 x3*)
then show *?case* **using** *subst-sb.simps s-branch-s-branch-list.fresh subst-b-e-def has-subst-b-class.forget-subst subst-b-v-def* **by** *force*
next
case (*AS-let2 x1 x2 x3 x4*)
then show *?case* **using** *subst-sb.simps s-branch-s-branch-list.fresh subst-b-e-def has-subst-b-class.forget-subst subst-b-τ-def* **by** *force*
next
case (*AS-var x1 x2 x3 x4*)
then show *?case* **using** *subst-sb.simps s-branch-s-branch-list.fresh subst-b-e-def has-subst-b-class.forget-subst subst-b-v-def* **using** *subst-b-τ-def*
proof −
have *f1*: (*atom a* ♯ *x4* ∨ *atom a* ∈ *set* [*atom x1*]) ∧ *atom a* ♯ *x2* ∧ *atom a* ♯ *x3*
using *AS-var.prems s-branch-s-branch-list.fresh* **by** *simp*
then have *atom a* ♯ *x4*
by (*metis* (*no-types*) *Nominal*−*Utils.fresh-star-singleton AS-var.hyps*(*1*) *empty-set fresh-star-def list.simps*(*15*) *not-self-fresh*)
then show *?thesis*
using *f1* **by** (*metis AS-var.hyps*(*3*) *has-subst-b-class.forget-subst subst-b-τ-def subst-b-v-def subst-sb.simps*(*7*))

qed

next
case (*AS-branch x1 x2 x3*)
then show *?case* **using** *subst-sb.simps s-branch-s-branch-list.fresh subst-b-e-def has-subst-b-class.forget-subst subst-b-v-def* **by** *force*
next
case (*AS-cons x1 x2 x3 x4*)
then show *?case* **using** *subst-sb.simps s-branch-s-branch-list.fresh subst-b-e-def has-subst-b-class.forget-subst subst-b-v-def* **by** *force*
next
case (*AS-val x*)
then show *?case* **using** *subst-sb.simps s-branch-s-branch-list.fresh subst-b-e-def has-subst-b-class.forget-subst subst-b-v-def* **by** *force*
next
case (*AS-if x1 x2 x3*)
then show *?case* **using** *subst-sb.simps s-branch-s-branch-list.fresh subst-b-e-def has-subst-b-class.forget-subst subst-b-v-def* **by** *force*
next
case (*AS-assign x1 x2*)
then show *?case* **using** *subst-sb.simps s-branch-s-branch-list.fresh subst-b-e-def has-subst-b-class.forget-subst subst-b-v-def* **by** *force*
next
case (*AS-match x1 x2*)
then show *?case* **using** *subst-sb.simps s-branch-s-branch-list.fresh subst-b-e-def has-subst-b-class.forget-subst subst-b-v-def* **by** *force*
next
case (*AS-while x1 x2*)
then show *?case* **using** *subst-sb.simps s-branch-s-branch-list.fresh subst-b-e-def has-subst-b-class.forget-subst subst-b-v-def* **by** *force*
next
case (*AS-seq x1 x2*)
then show *?case* **using** *subst-sb.simps s-branch-s-branch-list.fresh subst-b-e-def has-subst-b-class.forget-subst subst-b-v-def* **by** *force*
next
case (*AS-assert c s*)
then show *?case* **unfolding** *subst-sb.simps* **using**
s-branch-s-branch-list.fresh subst-b-e-def has-subst-b-class.forget-subst subst-b-v-def subst-b-c-def subst-cb.simps **by** *force*
qed(*auto+*)

lemma *subst-sb-id*: *subst-sb A a* (*B-var a*) = *A* **and**
subst-branchb-id [*simp*]: *subst-branchb B a* (*B-var a*) = *B* **and**
subst-branchlb-id: *subst-branchlb C a* (*B-var a*) = *C*
proof(*nominal-induct A* **and** *B* **and** *C avoiding*: *a rule*: *s-branch-s-branch-list.strong-induct*)
case (*AS-branch x1 x2 x3*)
then show *?case* **using** *subst-sb.simps s-branch-s-branch-list.fresh subst-b-τ-def has-subst-b-class.subst-id subst-b-v-def*
by *simp*
next

case (*AS-cons x1 x2*)
then show *?case* **using** *subst-sb.simps s-branch-s-branch-list.fresh subst-b-τ-def has-subst-b-class.subst-id subst-b-v-def* **by** *simp*
next
case (*AS-val x*)
then show *?case* **using** *subst-sb.simps s-branch-s-branch-list.fresh subst-b-τ-def has-subst-b-class.subst-id subst-b-v-def* **by** *metis*
next
case (*AS-if x1 x2 x3*)
then show *?case* **using** *subst-sb.simps s-branch-s-branch-list.fresh subst-b-τ-def has-subst-b-class.subst-id subst-b-v-def* **by** *metis*
next
case (*AS-assign x1 x2*)
then show *?case* **using** *subst-sb.simps s-branch-s-branch-list.fresh subst-b-τ-def has-subst-b-class.subst-id subst-b-v-def* **by** *metis*
next
case (*AS-match x1 x2*)
then show *?case* **using** *subst-sb.simps s-branch-s-branch-list.fresh subst-b-τ-def has-subst-b-class.subst-id subst-b-v-def* **by** *metis*
next
case (*AS-while x1 x2*)
then show *?case* **using** *subst-sb.simps s-branch-s-branch-list.fresh subst-b-τ-def has-subst-b-class.subst-id subst-b-v-def* **by** *metis*
next
case (*AS-seq x1 x2*)
then show *?case* **using** *subst-sb.simps s-branch-s-branch-list.fresh subst-b-τ-def has-subst-b-class.subst-id subst-b-v-def* **by** *metis*
next
case (*AS-let x1 x2 x3*)
then show *?case* **using** *subst-sb.simps s-branch-s-branch-list.fresh subst-b-e-def has-subst-b-class.subst-id* **by** *metis*
next
case (*AS-let2 x1 x2 x3 x4*)
then show *?case* **using** *subst-sb.simps s-branch-s-branch-list.fresh subst-b-τ-def has-subst-b-class.subst-id* **by** *metis*
next
case (*AS-var x1 x2 x3 x4*)
then show *?case* **using** *subst-sb.simps s-branch-s-branch-list.fresh subst-b-τ-def has-subst-b-class.subst-id subst-b-v-def* **by** *metis*
next
case (*AS-assert c s*)
then show *?case* **unfolding** *subst-sb.simps* **using** *s-branch-s-branch-list.fresh subst-b-c-def has-subst-b-class.subst-id* **by** *metis*
qed (*auto*)

lemma *flip-subst-s*:
fixes *bv*::*bv* **and** *s*::*s* **and** *cs*::*branch-s* **and** *z*::*bv*
shows *atom bv* ♯ *s* ⟹ ((*bv* ↔ *z*) ∙ *s*) = *s*[*z*::=*B-var bv*]$_{sb}$ **and**
atom bv ♯ *cs* ⟹ ((*bv* ↔ *z*) ∙ *cs*) = *subst-branchb cs z* (*B-var bv*) **and**
atom bv ♯ *css* ⟹ ((*bv* ↔ *z*) ∙ *css*) = *subst-branchlb css z* (*B-var bv*)

proof(*nominal-induct s* **and** *cs* **and** *css rule*: *s-branch-s-branch-list.strong-induct*)

case (*AS-branch x1 x2 x3*)
hence ((*bv* ↔ *z*) · *x1*) = *x1* **using** *pure-fresh fresh-at-base flip-fresh-fresh* **by** *metis*
moreover have ((*bv* ↔ *z*) · *x2*) = *x2* **using** *fresh-at-base flip-fresh-fresh*[*of bv x2 z*] *AS-branch* **by** *auto*
ultimately show *?case* **unfolding** *s-branch-s-branch-list.perm-simps subst-branchb.simps* **using** *s-branch-s-branch-list.fresh*(*1*) *AS-branch* **by** *auto*
next
case (*AS-cons x1 x2*)
hence ((*bv* ↔ *z*) · *x1*) = *subst-branchb x1 z* (*B-var bv*) **using** *pure-fresh fresh-at-base flip-fresh-fresh s-branch-s-branch-list.fresh*(*13*) **by** *metis*
moreover have ((*bv* ↔ *z*) · *x2*) = *subst-branchlb x2 z* (*B-var bv*) **using** *fresh-at-base flip-fresh-fresh*[*of bv x2 z*] *AS-cons s-branch-s-branch-list.fresh* **by** *metis*
ultimately show *?case* **unfolding** *s-branch-s-branch-list.perm-simps subst-branchb.simps* **using** *s-branch-s-branch-list.fresh*(*1*) *AS-cons* **by** *auto*
next
case (*AS-val x*)
then show *?case* **unfolding** *s-branch-s-branch-list.perm-simps subst-branchb.simps* **using** *flip-subst subst-b-v-def* **by** *simp*
next
case (*AS-let x1 x2 x3*)
moreover hence ((*bv* ↔ *z*) · *x1*) = *x1* **using** *fresh-at-base flip-fresh-fresh*[*of bv x1 z*] **by** *auto*
ultimately show *?case*
unfolding *s-branch-s-branch-list.perm-simps subst-sb.simps*
using *flip-subst subst-b-e-def s-branch-s-branch-list.fresh* **by** *auto*
next
case (*AS-let2 x1 x2 x3 x4*)
moreover hence ((*bv* ↔ *z*) · *x1*) = *x1* **using** *fresh-at-base flip-fresh-fresh*[*of bv x1 z*] **by** *auto*
ultimately show *?case*
unfolding *s-branch-s-branch-list.perm-simps subst-sb.simps*
using *flip-subst s-branch-s-branch-list.fresh*(*5*) *subst-b-τ-def* **by** *auto*
next
case (*AS-if x1 x2 x3*)
thus *?case*
unfolding *s-branch-s-branch-list.perm-simps subst-sb.simps*
using *flip-subst subst-b-e-def subst-b-v-def s-branch-s-branch-list.fresh* **by** *auto*
next
case (*AS-var x1 x2 x3 x4*)
thus *?case*
unfolding *s-branch-s-branch-list.perm-simps subst-sb.simps*
using *flip-subst subst-b-e-def subst-b-v-def subst-b-τ-def s-branch-s-branch-list.fresh fresh-at-base flip-fresh-fresh*[*of bv x1 z*] **by** *auto*
next
case (*AS-assign x1 x2*)
thus *?case*
unfolding *s-branch-s-branch-list.perm-simps subst-sb.simps*
using *flip-subst subst-b-e-def subst-b-v-def s-branch-s-branch-list.fresh fresh-at-base flip-fresh-fresh*[*of bv x1 z*] **by** *auto*
next
case (*AS-match x1 x2*)
thus *?case*
unfolding *s-branch-s-branch-list.perm-simps subst-sb.simps*
using *flip-subst subst-b-e-def subst-b-v-def s-branch-s-branch-list.fresh* **by** *auto*

next
case (*AS-while x1 x2*)
thus *?case*
  unfolding *s-branch-s-branch-list.perm-simps subst-sb.simps*
  using *flip-subst subst-b-e-def subst-b-v-def s-branch-s-branch-list.fresh* **by** *auto*
next
case (*AS-seq x1 x2*)
thus *?case*
  unfolding *s-branch-s-branch-list.perm-simps subst-sb.simps*
  using *flip-subst subst-b-e-def subst-b-v-def s-branch-s-branch-list.fresh* **by** *auto*
next
case (*AS-assert x1 x2*)
thus *?case*
  unfolding *s-branch-s-branch-list.perm-simps subst-sb.simps*
  using *flip-subst subst-b-c-def subst-b-v-def s-branch-s-branch-list.fresh* **by** *simp*
qed(*auto*)

lemma *flip-subst-subst-s*:
 fixes *bv*::*bv* **and** *s*::*s* **and** *cs*::*branch-s* **and** *z*::*bv*
 shows *atom bv* ♯ *s* ⟹ ((*bv* ↔ *z*) ∙ *s*)[*bv*::=*v*]$_{sb}$ = *s*[*z*::=*v*]$_{sb}$ **and**
  atom bv ♯ *cs* ⟹ *subst-branchb* ((*bv* ↔ *z*) ∙ *cs*) *bv v* = *subst-branchb cs z v* **and**
  atom bv ♯ *css* ⟹ *subst-branchlb* ((*bv* ↔ *z*) ∙ *css*) *bv v* = *subst-branchlb css z v*
proof(*nominal-induct s* **and** *cs* **and** *css rule*: *s-branch-s-branch-list.strong-induct*)
 case (*AS-branch x1 x2 x3*)
 hence ((*bv* ↔ *z*) ∙ *x1*) = *x1* **using** *pure-fresh fresh-at-base flip-fresh-fresh* **by** *metis*
 moreover have ((*bv* ↔ *z*) ∙ *x2*) = *x2* **using** *fresh-at-base flip-fresh-fresh*[*of bv x2 z*] *AS-branch* **by** *auto*
 ultimately show *?case* **unfolding** *s-branch-s-branch-list.perm-simps subst-branchb.simps* **using** *s-branch-s-branch-list.fresh*(*1*) *AS-branch* **by** *auto*
next
 case (*AS-cons x1 x2*)
 thus *?case*
  unfolding *s-branch-s-branch-list.perm-simps subst-branchlb.simps*
  using *s-branch-s-branch-list.fresh*(*1*) *AS-cons* **by** *auto*

next
 case (*AS-val x*)
 then show *?case* **unfolding** *s-branch-s-branch-list.perm-simps subst-branchb.simps* **using** *flip-subst subst-b-v-def* **by** *simp*
next
 case (*AS-let x1 x2 x3*)
 moreover hence ((*bv* ↔ *z*) ∙ *x1*) = *x1* **using** *fresh-at-base flip-fresh-fresh*[*of bv x1 z*] **by** *auto*
 ultimately show *?case*
  unfolding *s-branch-s-branch-list.perm-simps subst-sb.simps*
  using *flip-subst-subst subst-b-e-def s-branch-s-branch-list.fresh* **by** *force*
next
case (*AS-let2 x1 x2 x3 x4*)
 moreover hence ((*bv* ↔ *z*) ∙ *x1*) = *x1* **using** *fresh-at-base flip-fresh-fresh*[*of bv x1 z*] **by** *auto*
 ultimately show *?case*
  unfolding *s-branch-s-branch-list.perm-simps subst-sb.simps*
  using *flip-subst s-branch-s-branch-list.fresh*(*5*) *subst-b-τ-def* **by** *auto*
next

case (*AS-if x1 x2 x3*)
thus *?case*
unfolding *s-branch-s-branch-list.perm-simps subst-sb.simps*
using *flip-subst subst-b-e-def subst-b-v-def s-branch-s-branch-list.fresh* **by** *auto*
next
case (*AS-var x1 x2 x3 x4*)
thus *?case*
unfolding *s-branch-s-branch-list.perm-simps subst-sb.simps*
using *flip-subst subst-b-e-def subst-b-v-def subst-b-τ-def s-branch-s-branch-list.fresh fresh-at-base flip-fresh-fresh*[*of bv x1 z*] **by** *auto*
next
case (*AS-assign x1 x2*)
thus *?case*
unfolding *s-branch-s-branch-list.perm-simps subst-sb.simps*
using *flip-subst subst-b-e-def subst-b-v-def s-branch-s-branch-list.fresh fresh-at-base flip-fresh-fresh*[*of bv x1 z*] **by** *auto*
next
case (*AS-match x1 x2*)
thus *?case*
unfolding *s-branch-s-branch-list.perm-simps subst-sb.simps*
using *flip-subst subst-b-e-def subst-b-v-def s-branch-s-branch-list.fresh* **by** *auto*
next
case (*AS-while x1 x2*)
thus *?case*
unfolding *s-branch-s-branch-list.perm-simps subst-sb.simps*
using *flip-subst subst-b-e-def subst-b-v-def s-branch-s-branch-list.fresh* **by** *auto*
next
case (*AS-seq x1 x2*)
thus *?case*
unfolding *s-branch-s-branch-list.perm-simps subst-sb.simps*
using *flip-subst subst-b-e-def subst-b-v-def s-branch-s-branch-list.fresh* **by** *auto*
next
case (*AS-assert x1 x2*)
thus *?case*
unfolding *s-branch-s-branch-list.perm-simps subst-sb.simps*
using *flip-subst subst-b-e-def subst-b-c-def s-branch-s-branch-list.fresh* **by** *auto*
qed(*auto*)

instantiation *s* :: *has-subst-b*
begin
definition *subst-b* = (λ*s bv b*. *subst-sb s bv b*)

instance proof
fix *j*::*atom* **and** *i*::*bv* **and** *x*::*b* **and** *t*::*s*
show *j* ♯ *subst-b t i x* = ((*atom i* ♯ *t* ∧ *j* ♯ *t*) ∨ (*j* ♯ *x* ∧ (*j* ♯ *t* ∨ *j* = *atom i*)))
using *fresh-subst-sb-if subst-b-s-def* **by** *metis*

fix *a*::*bv* **and** *tm*::*s* **and** *x*::*b*
show *atom a* ♯ *tm* ⟹ *subst-b tm a x* = *tm* **using** *subst-b-s-def forget-subst-sb* **by** *metis*

fix *a*::*bv* **and** *tm*::*s*
show *subst-b tm a* (*B-var a*) = *tm* **using** *subst-b-s-def subst-sb-id* **by** *metis*

fix *p*::*perm* **and** *x1*::*bv* **and** *v*::*b* **and** *t1*::*s*
show *p* ∙ *subst-b t1 x1 v* = *subst-b* (*p* ∙ *t1*) (*p* ∙ *x1*) (*p* ∙ *v*) **using** *subst-b-s-def subst-sb-subst-branchb-subst-branchlb.eqvt*
by *metis*

fix *bv*::*bv* **and** *c*::*s* **and** *z*::*bv*
show *atom bv* ♯ *c* ⟹ ((*bv* ↔ *z*) ∙ *c*) = *c*[*z*::=*B-var bv*]$_b$
using *subst-b-s-def flip-subst-s* **by** *metis*

fix *bv*::*bv* **and** *c*::*s* **and** *z*::*bv* **and** *v*::*b*
show *atom bv* ♯ *c* ⟹ ((*bv* ↔ *z*) ∙ *c*)[*bv*::=*v*]$_b$ = *c*[*z*::=*v*]$_b$
using *flip-subst-subst-s subst-b-s-def* **by** *metis*

qed
end

5.9 Function Type

nominal-function *subst-ft-b* :: *fun-typ* ⇒ *bv* ⇒ *b* ⇒ *fun-typ* **where**
subst-ft-b (*AF-fun-typ z b c t* (*s*::*s*)) *x v* = *AF-fun-typ z* (*subst-bb b x v*) (*subst-cb c x v*) *t*[*x*::=*v*]$_{\tau b}$
s[*x*::=*v*]$_{sb}$
apply(*simp add*: *eqvt-def subst-ft-b-graph-aux-def*)
apply(*simp add*:*fun-typ.strong-exhaust*,*auto*)
apply(*rule-tac y*=*a* **and** *c*=(*aa*,*b*) **in** *fun-typ.strong-exhaust*)
apply (*auto simp*: *eqvt-at-def fresh-star-def fresh-Pair fresh-at-base*)
by *blast*

nominal-termination (*eqvt*) **by** *lexicographic-order*

nominal-function *subst-ftq-b* :: *fun-typ-q* ⇒ *bv* ⇒ *b* ⇒ *fun-typ-q* **where**
atom bv ♯ (*x*,*v*) ⟹ *subst-ftq-b* (*AF-fun-typ-some bv ft*) *x v* = (*AF-fun-typ-some bv* (*subst-ft-b ft x v*))
| *subst-ftq-b* (*AF-fun-typ-none ft*) *x v* = (*AF-fun-typ-none* (*subst-ft-b ft x v*))
apply(*simp add*: *eqvt-def subst-ftq-b-graph-aux-def*)
apply(*simp add*:*fun-typ-q.strong-exhaust*,*auto*)
apply(*rule-tac y*=*a* **and** *c*=(*aa*,*b*) **in** *fun-typ-q.strong-exhaust*)
by (*auto simp*: *eqvt-at-def fresh-star-def fresh-Pair fresh-at-base*)
nominal-termination (*eqvt*) **by** *lexicographic-order*

instantiation *fun-typ* :: *has-subst-b*
begin
definition *subst-b* = *subst-ft-b*

instance proof
fix *j*::*atom* **and** *i*::*bv* **and** *x*::*b* **and** *t*::*fun-typ*
show *j* ♯ *subst-b t i x* = (*atom i* ♯ *t* ∧ *j* ♯ *t* ∨ *j* ♯ *x* ∧ (*j* ♯ *t* ∨ *j* = *atom i*))
apply(*nominal-induct t avoiding*: *i x rule*:*fun-typ.strong-induct*)
apply(*auto simp add*: *subst-b-fun-typ-def*)
by(*metis fresh-subst-if subst-b-s-def subst-b-τ-def subst-b-b-def subst-b-c-def*)+

fix *a*::*bv* **and** *tm*::*fun-typ* **and** *x*::*b*
show *atom a* ♯ *tm* ⟹ *subst-b tm a x* = *tm*
apply (*nominal-induct tm avoiding*: *a x rule*: *fun-typ.strong-induct*)
apply(*simp add*: *subst-b-fun-typ-def Abs1-eq-iff'*)
using *subst-b-b-def subst-b-fun-typ-def subst-b-τ-def subst-b-c-def subst-b-s-def*
forget-subst fresh-at-base list.set-cases neq-Nil-conv set-ConsD
subst-ft-b.simps **by** *metis*

fix *a*::*bv* **and** *tm*::*fun-typ*
show *subst-b tm a* (*B-var a*) = *tm*
apply (*nominal-induct tm rule*: *fun-typ.strong-induct*)
apply(*simp add*: *subst-b-fun-typ-def Abs1-eq-iff'*,*auto*)
using *subst-b-b-def subst-b-fun-typ-def subst-b-τ-def subst-b-c-def subst-b-s-def*
forget-subst fresh-at-base list.set-cases neq-Nil-conv set-ConsD
subst-ft-b.simps
by (*metis has-subst-b-class.subst-id*)+

fix *p*::*perm* **and** *x1*::*bv* **and** *v*::*b* **and** *t1*::*fun-typ*
show *p* ∙ *subst-b t1 x1 v* = *subst-b* (*p* ∙ *t1*) (*p* ∙ *x1*) (*p* ∙ *v*)
apply (*nominal-induct t1 avoiding*: *x1 v rule*: *fun-typ.strong-induct*)
by(*auto simp add*: *subst-b-fun-typ-def Abs1-eq-iff' fun-typ.perm-simps*)

fix *bv*::*bv* **and** *c*::*fun-typ* **and** *z*::*bv*
show *atom bv* ♯ *c* ⟹ ((*bv* ↔ *z*) ∙ *c*) = *c*[*z*::=*B-var bv*]$_b$
apply (*nominal-induct c avoiding*: *z bv rule*: *fun-typ.strong-induct*)
by(*auto simp add*: *subst-b-fun-typ-def Abs1-eq-iff' fun-typ.perm-simps subst-b-b-def subst-b-c-def subst-b-τ-def subst-b-s-def*)

fix *bv*::*bv* **and** *c*::*fun-typ* **and** *z*::*bv* **and** *v*::*b*
show *atom bv* ♯ *c* ⟹ ((*bv* ↔ *z*) ∙ *c*)[*bv*::=*v*]$_b$ = *c*[*z*::=*v*]$_b$
apply (*nominal-induct c avoiding*: *bv v z rule*: *fun-typ.strong-induct*)
apply(*auto simp add*: *subst-b-fun-typ-def Abs1-eq-iff' fun-typ.perm-simps subst-b-b-def subst-b-c-def subst-b-τ-def subst-b-s-def flip-subst-subst flip-subst*)
using *subst-b-fun-typ-def Abs1-eq-iff' fun-typ.perm-simps subst-b-b-def subst-b-c-def subst-b-τ-def subst-b-s-def flip-subst-subst flip-subst*
using *flip-subst-s*(*1*) *flip-subst-subst-s*(*1*) **by** *auto*

qed
end

instantiation *fun-typ-q* :: *has-subst-b*
begin
definition *subst-b* = *subst-ftq-b*

instance proof
fix *j*::*atom* **and** *i*::*bv* **and** *x*::*b* **and** *t*::*fun-typ-q*
show *j* ♯ *subst-b t i x* = (*atom i* ♯ *t* ∧ *j* ♯ *t* ∨ *j* ♯ *x* ∧ (*j* ♯ *t* ∨ *j* = *atom i*))
apply (*nominal-induct t avoiding*: *i x j rule*: *fun-typ-q.strong-induct*,*auto simp add*: *subst-b-fun-typ-q-def*

subst-ftq-b.simps)
using *fresh-subst-if subst-b-fun-typ-q-def subst-b-s-def subst-b-τ-def subst-b-b-def subst-b-c-def subst-b-fun-typ-def*
apply *metis+*
done

fix *a*::*bv* **and** *t*::*fun-typ-q* **and** *x*::*b*
show *atom a* ♯ *t* ⟹ *subst-b t a x* = *t*
apply (*nominal-induct t avoiding*: *a x rule*: *fun-typ-q.strong-induct*)
apply(*auto simp add*: *subst-b-fun-typ-q-def subst-ftq-b.simps Abs1-eq-iff'*)
using *forget-subst subst-b-fun-typ-q-def subst-b-s-def subst-b-τ-def subst-b-b-def subst-b-c-def subst-b-fun-typ-def eqvt* **by** *metis+*

fix *p*::*perm* **and** *x1*::*bv* **and** *v*::*b* **and** *t*::*fun-typ-q*
show *p* ∙ *subst-b t x1 v* = *subst-b* (*p* ∙ *t*) (*p* ∙ *x1*) (*p* ∙ *v*)
apply (*nominal-induct t avoiding*: *x1 v rule*: *fun-typ-q.strong-induct*)
by(*auto simp add*: *subst-b-fun-typ-q-def subst-ftq-b.simps Abs1-eq-iff'*)

fix *a*::*bv* **and** *tm*::*fun-typ-q*
show *subst-b tm a* (*B-var a*) = *tm*
apply (*nominal-induct tm avoiding*: *a rule*: *fun-typ-q.strong-induct*)
apply(*auto simp add*: *subst-b-fun-typ-q-def subst-ftq-b.simps Abs1-eq-iff'*)
using *subst-id subst-b-b-def subst-b-fun-typ-def subst-b-τ-def subst-b-c-def subst-b-s-def forget-subst fresh-at-base list.set-cases neq-Nil-conv set-ConsD subst-ft-b.simps* **by** *metis+*

fix *bv*::*bv* **and** *c*::*fun-typ-q* **and** *z*::*bv*
show *atom bv* ♯ *c* ⟹ ((*bv* ↔ *z*) ∙ *c*) = *c*[*z*::=*B-var bv*]$_b$
apply (*nominal-induct c avoiding*: *z bv rule*: *fun-typ-q.strong-induct*)
apply(*auto simp add*: *subst-b-fun-typ-q-def subst-ftq-b.simps Abs1-eq-iff'*)
using *forget-subst subst-b-fun-typ-q-def subst-b-s-def subst-b-τ-def subst-b-b-def subst-b-c-def subst-b-fun-typ-def eqvt* **by** *metis+*

fix *bv*::*bv* **and** *c*::*fun-typ-q* **and** *z*::*bv* **and** *v*::*b*
show *atom bv* ♯ *c* ⟹ ((*bv* ↔ *z*) ∙ *c*)[*bv*::=*v*]$_b$ = *c*[*z*::=*v*]$_b$
apply (*nominal-induct c avoiding*: *z v bv rule*: *fun-typ-q.strong-induct*)
apply(*auto simp add*: *subst-b-fun-typ-q-def subst-ftq-b.simps Abs1-eq-iff'*)
using *flip-subst flip-subst-subst forget-subst subst-b-fun-typ-q-def subst-b-s-def subst-b-τ-def subst-b-b-def subst-b-c-def subst-b-fun-typ-def eqvt* **by** *metis+*

qed
end

5.10 Contexts

5.10.1 Immutable Variables

nominal-function *subst-gb* :: Γ ⇒ *bv* ⇒ *b* ⇒ Γ **where**
subst-gb GNil - - = GNil
| *subst-gb* ((*y*,*b'*,*c*)#$_\Gamma$Γ) *bv b* = ((*y*,*b'*[*bv*::=*b*]$_{bb}$,*c*[*bv*::=*b*]$_{cb}$)#$_\Gamma$ (*subst-gb* Γ *bv b*))
apply (*simp add*: *eqvt-def subst-gb-graph-aux-def*)+
apply *auto*

proof(*goal-cases*)
case (*1 P a1 a2 b*)
then show *?case* **using** Γ.*exhaust neq-GNil-conv* **by** *force*
qed
nominal-termination (*eqvt*) **by** *lexicographic-order*

abbreviation
subst-gb-abbrev :: Γ ⇒ *bv* ⇒ *b* ⇒ Γ (-[-::=-]$_{\Gamma b}$ [*1000*,*50*,*50*] *1000*)
where
$g[bv::=b']_{\Gamma b}$ ≡ *subst-gb g bv b'*

instantiation Γ :: *has-subst-b*
begin
definition *subst-b* = *subst-gb*

instance proof
fix *j*::*atom* **and** *i*::*bv* **and** *x*::*b* **and** *t*::Γ
show *j* ♯ *subst-b t i x* = (*atom i* ♯ *t* ∧ *j* ♯ *t* ∨ *j* ♯ *x* ∧ (*j* ♯ *t* ∨ *j* = *atom i*))
proof(*induct t rule*: Γ-*induct*)
case *GNil*
then show *?case* **using** *fresh-GNil subst-gb.simps fresh-def pure-fresh subst-b-*Γ*-def has-subst-b-class.fresh-subst-if fresh-GNil fresh-GCons* **by** *metis*
next
case (*GCons x' b' c'* Γ')
have ∗: *atom i* ♯ *x'* **using** *fresh-at-base* **by** *simp*

have *j* ♯ *subst-b* (*x'*, *b'*, *c'*) #Γ Γ') *i x* = *j* ♯ ((*x'*, $b'[i::=x]_{bb}$, $c'[i::=x]_{cb}$) #Γ (*subst-b* Γ' *i x*)) **using** *subst-gb.simps subst-b-*Γ*-def* **by** *auto*
also have ... = (*j* ♯ ((*x'*, $b'[i::=x]_{bb}$, $c'[i::=x]_{cb}$)) ∧ (*j* ♯ (*subst-b* Γ' *i x*))) **using** *fresh-GCons* **by** *auto*
also have ... = (((*j* ♯ *x'*) ∧ (*j* ♯ $b'[i::=x]_{bb}$) ∧ (*j* ♯ $c'[i::=x]_{cb}$)) ∧ (*j* ♯ (*subst-b* Γ' *i x*))) **by** *auto*
also have ... = (((*j* ♯ *x'*) ∧ ((*atom i* ♯ *b'* ∧ *j* ♯ *b'* ∨ *j* ♯ *x* ∧ (*j* ♯ *b'* ∨ *j* = *atom i*))) ∧
((*atom i* ♯ *c'* ∧ *j* ♯ *c'* ∨ *j* ♯ *x* ∧ (*j* ♯ *c'* ∨ *j* = *atom i*))) ∧
((*atom i* ♯ Γ' ∧ *j* ♯ Γ' ∨ *j* ♯ *x* ∧ (*j* ♯ Γ' ∨ *j* = *atom i*)))))
using *fresh-subst-if*[*of j b' i x*] *fresh-subst-if*[*of j c' i x*] *GCons subst-b-b-def subst-b-c-def* **by** *simp*
also have ... = ((*atom i* ♯ (*x'*, *b'*, *c'*) #Γ Γ' ∧ *j* ♯ (*x'*, *b'*, *c'*) #Γ Γ') ∨ (*j* ♯ *x* ∧ (*j* ♯ (*x'*, *b'*, *c'*) #Γ Γ' ∨ *j* = *atom i*))) **using** ∗ *fresh-GCons fresh-prod3* **by** *metis*

finally show *?case* **by** *auto*
qed

fix *a*::*bv* **and** *tm*::Γ **and** *x*::*b*
show *atom a* ♯ *tm* ⟹ *subst-b tm a x* = *tm*
proof (*induct tm rule*: Γ-*induct*)
case *GNil*
then show *?case* **using** *subst-gb.simps subst-b-*Γ*-def* **by** *auto*
next
case (*GCons x' b' c'* Γ')
have ∗:$b'[a::=x]_{bb}$ = *b'* ∧ $c'[a::=x]_{cb}$ = *c'* **using** *GCons fresh-GCons*[*of atom a*] *fresh-prod3*[*of atom*

a] *has-subst-b-class.forget-subst subst-b-b-def subst-b-c-def* **by** *metis*
have *subst-b* $((x', b', c') \#_\Gamma \Gamma')$ *a x* = $((x', b'[a::=x]_{bb}, c'[a::=x]_{cb}) \#_\Gamma$ (*subst-b* Γ' *a x*)) **using** *subst-b-Γ-def subst-gb.simps* **by** *auto*
also have ... = $((x', b', c') \#_\Gamma \Gamma')$ **using** $*$ *GCons fresh-GCons*[*of atom a*] **by** *auto*
finally show *?case* **using** *has-subst-b-class.forget-subst fresh-GCons fresh-prod3 GCons subst-b-Γ-def has-subst-b-class.forget-subst*[*of a b' x*] *fresh-prod3*[*of atom a*] **by** *argo*
qed

fix *a*::*bv* **and** *tm*::Γ
show *subst-b tm a* (*B-var a*) = *tm*
proof(*induct tm rule*: Γ-*induct*)
case *GNil*
then show *?case* **using** *subst-gb.simps subst-b-Γ-def* **by** *auto*
next
case (*GCons x' b' c' Γ'*)
then show *?case* **using** *has-subst-b-class.subst-id subst-b-Γ-def subst-b-b-def subst-b-c-def subst-gb.simps* **by** *metis*
qed

fix *p*::*perm* **and** *x1*::*bv* **and** *v*::*b* **and** *t1*::Γ
show *p* $\cdot$ *subst-b t1 x1 v* = *subst-b* (*p* $\cdot$ *t1*) (*p* $\cdot$ *x1*) (*p* $\cdot$ *v*)
proof (*induct tm rule*: Γ-*induct*)
case *GNil*
then show *?case* **using** *subst-b-Γ-def subst-gb.simps* **by** *simp*
next
case (*GCons x' b' c' Γ'*)
then show *?case* **using** *subst-b-Γ-def subst-gb.simps has-subst-b-class.eqvt* **by** *argo*
qed

fix *bv*::*bv* **and** *c*::Γ **and** *z*::*bv*
show *atom bv* $\sharp$ *c* $\Longrightarrow$ ((*bv* $\leftrightarrow$ *z*) $\cdot$ *c*) = *c*[*z*::=*B-var bv*]$_b$
proof (*induct c rule*: Γ-*induct*)
case *GNil*
then show *?case* **using** *subst-b-Γ-def subst-gb.simps* **by** *auto*
next
case (*GCons x b c Γ'*)
have $*$:(*bv* $\leftrightarrow$ *z*) $\cdot$ (*x*, *b*, *c*) = (*x*, (*bv* $\leftrightarrow$ *z*) $\cdot$ *b*, (*bv* $\leftrightarrow$ *z*) $\cdot$ *c*) **using** *flip-bv-x-cancel* **by** *auto*
then show *?case*
unfolding *subst-gb.simps subst-b-Γ-def permute-Γ.simps* $*$
using *GCons subst-b-Γ-def subst-gb.simps flip-subst subst-b-b-def subst-b-c-def fresh-GCons* **by** *auto*
qed

fix *bv*::*bv* **and** *c*::Γ **and** *z*::*bv* **and** *v*::*b*
show *atom bv* $\sharp$ *c* $\Longrightarrow$ ((*bv* $\leftrightarrow$ *z*) $\cdot$ *c*)[*bv*::=*v*]$_b$ = *c*[*z*::=*v*]$_b$
proof (*induct c rule*: Γ-*induct*)
case *GNil*
then show *?case* **using** *subst-b-Γ-def subst-gb.simps* **by** *auto*
next
case (*GCons x b c Γ'*)
have $*$:(*bv* $\leftrightarrow$ *z*) $\cdot$ (*x*, *b*, *c*) = (*x*, (*bv* $\leftrightarrow$ *z*) $\cdot$ *b*, (*bv* $\leftrightarrow$ *z*) $\cdot$ *c*) **using** *flip-bv-x-cancel* **by** *auto*
then show *?case*
unfolding *subst-gb.simps subst-b-Γ-def permute-Γ.simps* $*$

using *GCons subst-b-Γ-def subst-gb.simps flip-subst subst-b-b-def subst-b-c-def fresh-GCons* **by** *auto*
qed
qed
end

lemma *subst-b-base-for-lit*:
(*base-for-lit l*)$[bv::=b]_{bb}$ = *base-for-lit l*
using *base-for-lit.simps l.strong-exhaust*
by (*metis subst-bb.simps*(*2*) *subst-bb.simps*(*3*) *subst-bb.simps*(*6*) *subst-bb.simps*(*7*))

lemma *subst-b-lookup*:
assumes *Some* (*b*, *c*) = *lookup* Γ *x*
shows *Some* ($b[bv::=b']_{bb}$, $c[bv::=b']_{cb}$) = *lookup* $\Gamma[bv::=b']_{\Gamma b}$ *x*
using *assms* **by**(*induct* Γ *rule*: Γ-*induct*, *auto*)

lemma *subst-g-b-x-fresh*:
fixes *x*::*x* **and** *b*::*b* **and** Γ::Γ **and** *bv*::*bv*
assumes *atom x* ♯ Γ
shows *atom x* ♯ $\Gamma[bv::=b]_{\Gamma b}$
using *subst-b-fresh-x subst-b-Γ-def assms* **by** *metis*

5.10.2 Mutable Variables

nominal-function *subst-db* :: Δ ⇒ *bv* ⇒ *b* ⇒ Δ **where**
subst-db $[]_\Delta$ - - = $[]_\Delta$
| *subst-db* ((*u*,*t*) $\#_\Delta$ Δ) *bv b* = ((*u*,*t*$[bv::=b]_{\tau b}$) $\#_\Delta$ (*subst-db* Δ *bv b*))
apply (*simp add*: *eqvt-def subst-db-graph-aux-def*,*auto*)
using *list.exhaust delete-aux.elims*
using *neq-DNil-conv* **by** *fastforce*
nominal-termination (*eqvt*) **by** *lexicographic-order*

abbreviation
subst-db-abbrev :: Δ ⇒ *bv* ⇒ *b* ⇒ Δ (-[-::=-]$_{\Delta b}$ [*1000*,*50*,*50*] *1000*)
where
Δ$[bv::=b]_{\Delta b}$ ≡ *subst-db* Δ *bv b*

instantiation Δ :: *has-subst-b*
begin
definition *subst-b* = *subst-db*

instance proof
fix *j*::*atom* **and** *i*::*bv* **and** *x*::*b* **and** *t*::Δ
show *j* ♯ *subst-b t i x* = (*atom i* ♯ *t* ∧ *j* ♯ *t* ∨ *j* ♯ *x* ∧ (*j* ♯ *t* ∨ *j* = *atom i*))
proof(*induct t rule*: Δ-*induct*)
case *DNil*
then show *?case* **using** *fresh-DNil subst-db.simps fresh-def pure-fresh subst-b-Δ-def has-subst-b-class.fresh-subst-if fresh-DNil fresh-DCons* **by** *metis*
next
case (*DCons u t* Γ′)
have *j* ♯ *subst-b* ((*u*, *t*) $\#_\Delta$ Γ′) *i x* = *j* ♯ ((*u*, *t*$[i::=x]_{\tau b}$) $\#_\Delta$ (*subst-b* Γ′ *i x*)) **using** *subst-db.simps subst-b-Δ-def* **by** *auto*

also have ... = $(j \sharp ((u,\, t[i::=x]_{\tau b})) \wedge (j \sharp\ (subst\text{-}b\ \Gamma'\ i\ x)))$ **using** *fresh-DCons* **by** *auto*
also have ... = $(((j \sharp u) \wedge (j \sharp\ t[i::=x]_{\tau b})) \wedge (j \sharp\ (subst\text{-}b\ \Gamma'\ i\ x)))$ **by** *auto*
also have ... = $((j \sharp u) \wedge ((atom\ i \sharp t \wedge j \sharp t) \vee (j \sharp x \wedge (j \sharp t \vee j = atom\ i))) \wedge (atom\ i \sharp \Gamma' \wedge j \sharp \Gamma' \vee j \sharp x \wedge (j \sharp \Gamma' \vee j = atom\ i)))$
using *has-subst-b-class.fresh-subst-if*[*of j t i x*] *subst-b-τ-def DCons subst-b-Δ-def* **by** *auto*
also have ... = $(atom\ i \sharp (u,\, t) \#_\Delta \Gamma' \wedge j \sharp (u,\, t) \#_\Delta \Gamma' \vee j \sharp x \wedge (j \sharp (u,\, t) \#_\Delta \Gamma' \vee j = atom\ i))$
using *DCons subst-db.simps*(2) *has-subst-b-class.fresh-subst-if fresh-DCons subst-b-Δ-def pure-fresh fresh-at-base* **by** *auto*
finally show *?case* **by** *auto*
qed

fix *a*::*bv* **and** *tm*::Δ **and** *x*::*b*
show $atom\ a \sharp tm \Longrightarrow subst\text{-}b\ tm\ a\ x = tm$
proof (*induct tm rule*: Δ-*induct*)
case *DNil*
then show *?case* **using** *subst-db.simps subst-b-Δ-def* **by** *auto*
next
case (*DCons u t* Γ′)
have ∗:$t[a::=x]_{\tau b} = t$ **using** *DCons fresh-DCons*[*of atom a*] *fresh-prod2*[*of atom a*] *has-subst-b-class.forget-subst subst-b-τ-def* **by** *metis*
have $subst\text{-}b\ ((u,t)\ \#_\Delta\ \Gamma')\ a\ x = ((u,t[a::=x]_{\tau b})\ \#_\Delta\ (subst\text{-}b\ \Gamma'\ a\ x))$ **using** *subst-b-Δ-def subst-db.simps* **by** *auto*
also have ... = $((u,\, t)\ \#_\Delta\ \Gamma')$ **using** ∗ *DCons fresh-DCons*[*of atom a*] **by** *auto*
finally show *?case* **using**
has-subst-b-class.forget-subst fresh-DCons fresh-prod3
DCons subst-b-Δ-def has-subst-b-class.forget-subst[*of a t x*] *fresh-prod3*[*of atom a*] **by** *argo*
qed

fix *a*::*bv* **and** *tm*::Δ
show *subst-b tm a* (*B-var a*) = *tm*
proof(*induct tm rule*: Δ-*induct*)
case *DNil*
then show *?case* **using** *subst-db.simps subst-b-Δ-def* **by** *auto*
next
case (*DCons u t* Γ′)
then show *?case* **using** *has-subst-b-class.subst-id subst-b-Δ-def subst-b-τ-def subst-db.simps* **by** *metis*
qed

fix *p*::*perm* **and** *x1*::*bv* **and** *v*::*b* **and** *t1*::Δ
show $p \cdot subst\text{-}b\ t1\ x1\ v\ = subst\text{-}b\ \ (p \cdot t1)\ (p \cdot x1)\ (p \cdot v)$
proof (*induct tm rule*: Δ-*induct*)
case *DNil*
then show *?case* **using** *subst-b-Δ-def subst-db.simps* **by** *simp*
next
case (*DCons x′ b′* Γ′)
then show *?case* **by** *argo*
qed

fix *bv*::*bv* **and** *c*::Δ **and** *z*::*bv*
show $atom\ bv \sharp c \Longrightarrow ((bv \leftrightarrow z) \cdot c) = c[z::=B\text{-}var\ bv]_b$

proof (*induct c rule*: Δ-*induct*)
case *DNil*
then show *?case* **using** *subst-b-*Δ*-def subst-db.simps* **by** *auto*
next
case (*DCons u t'*)
then show *?case*
unfolding *subst-db.simps subst-b-*Δ*-def permute-*Δ*.simps*
using *DCons subst-b-*Δ*-def subst-db.simps flip-subst subst-b-*τ*-def flip-fresh-fresh fresh-at-base fresh-DCons flip-bv-u-cancel* **by** *simp*
qed

fix *bv*::*bv* **and** *c*::Δ **and** *z*::*bv* **and** *v*::*b*
show *atom bv* ♯ *c* ⟹ ((*bv* ↔ *z*) ∙ *c*)[*bv*::=*v*]$_b$ = *c*[*z*::=*v*]$_b$
proof (*induct c rule*: Δ-*induct*)
case *DNil*
then show *?case* **using** *subst-b-*Δ*-def subst-db.simps* **by** *auto*
next
case (*DCons u t'*)
then show *?case*
unfolding *subst-db.simps subst-b-*Δ*-def permute-*Δ*.simps*
using *DCons subst-b-*Δ*-def subst-db.simps flip-subst subst-b-*τ*-def flip-fresh-fresh fresh-at-base fresh-DCons flip-bv-u-cancel* **by** *simp*
qed
qed
end

lemma *subst-d-b-member*:
assumes (*u*, τ) ∈ *setD* Δ
shows (*u*, τ[*bv*::=*b*]$_{\tau b}$) ∈ *setD* Δ[*bv*::=*b*]$_{\Delta b}$
using *assms* **by** (*induct* Δ,*auto*)

lemmas *ms-fresh-all* = *e.fresh s-branch-s-branch-list.fresh* τ*.fresh c.fresh ce.fresh v.fresh l.fresh fresh-at-base opp.fresh pure-fresh ms-fresh*

lemmas *fresh-intros*[*intro*] = *fresh-GNil x-not-in-b-set x-not-in-u-atoms x-fresh-b u-not-in-x-atoms bv-not-in-x-atoms u-not-in-b-atoms*

lemmas *subst-b-simps* = *subst-tb.simps subst-cb.simps subst-ceb.simps subst-vb.simps subst-bb.simps subst-eb.simps subst-branchb.simps subst-sb.simps*

ML ‹*Ctr-Sugar.ctr-sugar-of* @{*context*} @{*type-name b*} |> *Option.map* #*ctrs*›

lemma *subst-d-b-x-fresh*:
fixes *x*::*x* **and** *b*::*b* **and** Δ::Δ **and** *bv*::*bv*
assumes *atom x* ♯ Δ
shows *atom x* ♯ Δ[*bv*::=*b*]$_{\Delta b}$
using *subst-b-fresh-x subst-b-*Δ*-def assms* **by** *metis*

lemma *subst-b-fresh-x*:
fixes *x*::*x*

shows $atom\ x \sharp v \Longrightarrow atom\ x \sharp v[bv::=b']_{vb}$ **and**
$atom\ x \sharp ce \Longrightarrow atom\ x \sharp ce[bv::=b']_{ceb}$ **and**
$atom\ x \sharp e \Longrightarrow atom\ x \sharp e[bv::=b']_{eb}$ **and**
$atom\ x \sharp c \Longrightarrow atom\ x \sharp c[bv::=b']_{cb}$ **and**
$atom\ x \sharp t \Longrightarrow atom\ x \sharp t[bv::=b']_{\tau b}$ **and**
$atom\ x \sharp d \Longrightarrow atom\ x \sharp d[bv::=b']_{\Delta b}$ **and**
$atom\ x \sharp g \Longrightarrow atom\ x \sharp g[bv::=b']_{\Gamma b}$ **and**
$atom\ x \sharp s \Longrightarrow atom\ x \sharp s[bv::=b']_{sb}$
using *fresh-subst-if x-fresh-b subst-b-v-def subst-b-ce-def subst-b-e-def subst-b-c-def subst-b-τ-def subst-b-s-def subst-g-b-x-fresh subst-d-b-x-fresh*
by *metis+*

lemma *subst-b-fresh-u-cls*:
fixes $tm::'a::has\text{-}subst\text{-}b$ **and** $x::u$
shows $atom\ x \sharp tm = atom\ x \sharp tm[bv::=b']_b$
using *fresh-subst-if*[*of atom x tm bv b'*] **using** *u-fresh-b* **by** *auto*

lemma *subst-g-b-u-fresh*:
fixes $x::u$ **and** $b::b$ **and** $\Gamma::\Gamma$ **and** $bv::bv$
assumes $atom\ x \sharp \Gamma$
shows $atom\ x \sharp \Gamma[bv::=b]_{\Gamma b}$
using *subst-b-fresh-u-cls subst-b-Γ-def assms* **by** *metis*

lemma *subst-d-b-u-fresh*:
fixes $x::u$ **and** $b::b$ **and** $\Gamma::\Delta$ **and** $bv::bv$
assumes $atom\ x \sharp \Gamma$
shows $atom\ x \sharp \Gamma[bv::=b]_{\Delta b}$
using *subst-b-fresh-u-cls subst-b-Δ-def assms* **by** *metis*

lemma *subst-b-fresh-u*:
fixes $x::u$
shows $atom\ x \sharp v \Longrightarrow atom\ x \sharp v[bv::=b']_{vb}$ **and**
$atom\ x \sharp ce \Longrightarrow atom\ x \sharp ce[bv::=b']_{ceb}$ **and**
$atom\ x \sharp e \Longrightarrow atom\ x \sharp e[bv::=b']_{eb}$ **and**
$atom\ x \sharp c \Longrightarrow atom\ x \sharp c[bv::=b']_{cb}$ **and**
$atom\ x \sharp t \Longrightarrow atom\ x \sharp t[bv::=b']_{\tau b}$ **and**
$atom\ x \sharp d \Longrightarrow atom\ x \sharp d[bv::=b']_{\Delta b}$ **and**
$atom\ x \sharp g \Longrightarrow atom\ x \sharp g[bv::=b']_{\Gamma b}$ **and**
$atom\ x \sharp s \Longrightarrow atom\ x \sharp s[bv::=b']_{sb}$
using *fresh-subst-if u-fresh-b subst-b-v-def subst-b-ce-def subst-b-e-def subst-b-c-def subst-b-τ-def subst-b-s-def subst-g-b-u-fresh subst-d-b-u-fresh*
by *metis+*

lemma *subst-db-u-fresh*:
fixes $u::u$ **and** $b::b$ **and** $D::\Delta$
assumes $atom\ u \sharp D$
shows $atom\ u \sharp D[bv::=b]_{\Delta b}$
using *assms* **proof**(*induct D rule*: Δ-*induct*)
case *DNil*
then show *?case* **by** *auto*
next
case (*DCons u' t' D'*)

then show *?case* **using** *subst-db.simps fresh-def fresh-DCons fresh-subst-if subst-b-τ-def*
by (*metis fresh-Pair u-not-in-b-atoms*)
qed

lemma *flip-bt-subst4*:
fixes $t::\tau$ **and** $bv::bv$
assumes *atom bv* $\sharp$ *t*
shows $t[bv'::=b]_{\tau b} = ((bv' \leftrightarrow bv) \cdot t)[bv::=b]_{\tau b}$
using *flip-subst-subst*[*OF assms*,*of bv' b*]
by (*simp add*: *flip-commute subst-b-τ-def*)

lemma *subst-bt-flip-sym*:
fixes $t1::\tau$ **and** $t2::\tau$
assumes *atom bv* $\sharp$ *b* **and** *atom bv* $\sharp$ (*bv1*, *bv2*, *t1*, *t2*) **and** $(bv1 \leftrightarrow bv) \cdot t1 = (bv2 \leftrightarrow bv) \cdot t2$
shows $t1[bv1::=b]_{\tau b} = t2[bv2::=b]_{\tau b}$
using *assms flip-bt-subst4*[*of bv t1 bv1 b*] *flip-bt-subst4 fresh-prod4 fresh-Pair* **by** *metis*

end

Chapter 6

Wellformed Terms

We require that expressions and values are well-sorted. We identify sort with base. Define a large cluster of mutually recursive inductive predicates. Some of the proofs are across all of the predicates and although they seemed at first to be daunting they have all worked out well with only the cases where you think something special needs to be done having some non-uniform part of the proof.

named-theorems *ms-wb Facts for helping with well−sortedness*

6.1 Definitions

inductive $wfV :: \Theta \Rightarrow \mathcal{B} \Rightarrow \Gamma \Rightarrow v \Rightarrow b \Rightarrow bool\ (\ -\ ;\ -\ ;\ -\ \vdash_{wf}\ -\ :\ -\ \ [50,50,50]\ 50)$ **and**
$wfC :: \Theta \Rightarrow \mathcal{B} \Rightarrow \Gamma \Rightarrow c \Rightarrow bool\ (\ -\ ;\ -\ ;\ -\ \ \vdash_{wf}\ -\ \ [50,50]\ 50)$ **and**
$wfG :: \Theta \Rightarrow \mathcal{B} \Rightarrow \Gamma \Rightarrow bool\ (\ -\ ;\ -\ \vdash_{wf}\ -\ \ [50,50]\ 50)$ **and**
$wfT :: \Theta \Rightarrow \mathcal{B} \Rightarrow \Gamma \Rightarrow \tau \Rightarrow bool\ (\ -\ ;\ -\ ;\ -\ \ \vdash_{wf}\ -\ \ [50,50]\ 50)$ **and**
$wfTs :: \Theta \Rightarrow \mathcal{B} \Rightarrow \Gamma \Rightarrow (string*\tau)\ list \Rightarrow bool\ (\ -\ ;\ -\ \ ;\ -\ \vdash_{wf}\ -\ \ [50,50]\ 50)$ **and**
$wfTh :: \Theta \Rightarrow bool\ (\ \ \vdash_{wf}\ -\ \ [50]\ 50)$ **and**
$wfB :: \Theta \Rightarrow \mathcal{B} \Rightarrow b \Rightarrow bool\ (\ -\ ;\ -\ \vdash_{wf}\ -\ \ [50,50]\ 50)$ **and**
$wfCE :: \Theta \Rightarrow \mathcal{B} \Rightarrow \Gamma \Rightarrow ce \Rightarrow b \Rightarrow bool\ (\ \ -\ ;\ -\ ;\ -\ \vdash_{wf}\ -\ :\ -\ \ [50,50,50]\ 50)$ **and**
$wfTD :: \Theta \Rightarrow type\text{-}def \Rightarrow bool\ (\ -\ \vdash_{wf}\ -\ \ [50,50]\ 50)$
where

$wfB\text{-}intI$: $\vdash_{wf} \Theta \Longrightarrow \Theta\ ;\ \mathcal{B} \vdash_{wf} B\text{-}int$
| $wfB\text{-}boolI$: $\vdash_{wf} \Theta \Longrightarrow \Theta\ ;\ \mathcal{B} \vdash_{wf} B\text{-}bool$
| $wfB\text{-}unitI$: $\vdash_{wf} \Theta \Longrightarrow \Theta\ ;\ \mathcal{B} \vdash_{wf} B\text{-}unit$
| $wfB\text{-}bitvecI$: $\vdash_{wf} \Theta \Longrightarrow \Theta\ ;\ \mathcal{B} \vdash_{wf} B\text{-}bitvec$
| $wfB\text{-}pairI$: $[\![\ \Theta\ ;\ \mathcal{B} \vdash_{wf} b1\ ;\ \Theta\ ;\ \mathcal{B} \vdash_{wf} b2\ ]\!] \Longrightarrow \Theta\ ;\ \mathcal{B} \vdash_{wf} B\text{-}pair\ b1\ b2$

| $wfB\text{-}consI$: $[\![$
$\vdash_{wf} \Theta$;
$(AF\text{-}typedef\ s\ dclist) \in set\ \Theta$
$]\!] \Longrightarrow$
$\Theta\ ;\ \mathcal{B} \vdash_{wf} B\text{-}id\ s$

| $wfB\text{-}appI$: $[\![$
$\vdash_{wf} \Theta$;
$\Theta\ ;\ \mathcal{B} \vdash_{wf} b$;

$(AF\text{-}typedef\text{-}poly\ s\ bv\ dclist) \in set\ \Theta$
$\rrbracket \Longrightarrow$
$\Theta\ ;\ \mathcal{B} \vdash_{wf} B\text{-}app\ s\ b$

| *wfV-varI*: $\llbracket\ \Theta\ ;\ \mathcal{B} \vdash_{wf} \Gamma\ ;\ Some\ (b,c) = lookup\ \Gamma\ x\ \rrbracket \Longrightarrow \Theta\ ;\ \mathcal{B}\ ;\ \Gamma \vdash_{wf} V\text{-}var\ x : b$
| *wfV-litI*: $\Theta\ ;\ \mathcal{B} \vdash_{wf} \Gamma\ \Longrightarrow \Theta\ ;\ \mathcal{B}\ ;\ \Gamma \vdash_{wf} V\text{-}lit\ l : base\text{-}for\text{-}lit\ l$

| *wfV-pairI*: $\llbracket$
$\Theta\ ;\ \mathcal{B}\ ;\ \Gamma \vdash_{wf} v1 : b1\ ;$
$\Theta\ ;\ \mathcal{B}\ ;\ \Gamma \vdash_{wf} v2 : b2$
$\rrbracket \Longrightarrow$
$\Theta\ ;\ \mathcal{B}\ ;\ \Gamma \vdash_{wf} (V\text{-}pair\ v1\ v2) : B\text{-}pair\ b1\ b2$

| *wfV-consI*: $\llbracket$
$AF\text{-}typedef\ s\ dclist \in set\ \Theta;$
$(dc, \{\!|\ x : b'\ |\ c\ |\!\}) \in set\ dclist\ ;$
$\Theta\ ;\ \mathcal{B}\ ;\ \Gamma \vdash_{wf} v : b'$
$\rrbracket \Longrightarrow$
$\Theta\ ;\ \mathcal{B}\ ;\ \Gamma \vdash_{wf} V\text{-}cons\ s\ dc\ v : B\text{-}id\ s$

| *wfV-conspI*: $\llbracket$
$AF\text{-}typedef\text{-}poly\ s\ bv\ dclist \in set\ \Theta;$
$(dc, \{\!|\ x : b'\ |\ c\ |\!\}) \in set\ dclist\ ;$
$\Theta\ ;\ \mathcal{B} \vdash_{wf} b;$
$atom\ bv \sharp (\Theta, \mathcal{B}, \Gamma, b\ , v);$
$\Theta\ ;\ \mathcal{B}\ ;\ \Gamma \vdash_{wf} v : b'[bv::=b]_{bb}$
$\rrbracket \Longrightarrow$
$\Theta\ ;\ \mathcal{B}\ ;\ \Gamma \vdash_{wf} V\text{-}consp\ s\ dc\ b\ v : B\text{-}app\ s\ b$

| *wfCE-valI* : $\llbracket$
$\Theta\ ;\ \mathcal{B}\ ;\ \Gamma \vdash_{wf} v\ : b$
$\rrbracket \Longrightarrow$
$\Theta\ ;\ \mathcal{B}\ ;\ \Gamma\ \vdash_{wf} CE\text{-}val\ v\ : b$

| *wfCE-plusI*: $\llbracket$
$\Theta\ ;\ \mathcal{B}\ ;\ \Gamma \vdash_{wf} v1 : B\text{-}int;$
$\Theta\ ;\ \mathcal{B}\ ;\ \Gamma \vdash_{wf} v2 : B\text{-}int$
$\rrbracket \Longrightarrow$
$\Theta\ ;\ \mathcal{B}\ ;\ \Gamma\ \vdash_{wf} CE\text{-}op\ Plus\ v1\ v2 : B\text{-}int$

| *wfCE-leqI*:$\llbracket$
$\Theta\ ;\ \mathcal{B}\ ;\ \Gamma \vdash_{wf} v1 : B\text{-}int;$
$\Theta\ ;\ \mathcal{B}\ ;\ \Gamma \vdash_{wf} v2 : B\text{-}int$
$\rrbracket \Longrightarrow$
$\Theta\ ;\ \mathcal{B}\ ;\ \Gamma \vdash_{wf} CE\text{-}op\ LEq\ v1\ v2 : B\text{-}bool$

| *wfCE-fstI*: $\llbracket$
$\Theta\ ;\ \mathcal{B}\ ;\ \Gamma \vdash_{wf} v1 : B\text{-}pair\ b1\ b2$
$\rrbracket \Longrightarrow$
$\Theta\ ;\ \mathcal{B}\ ;\ \Gamma\ \vdash_{wf} CE\text{-}fst\ v1 : b1$

$$
\begin{array}{l}
\mid \mathit{wfCE\text{-}sndI}: [\![\\
\quad \Theta\ ;\ \mathcal{B}\ ;\ \Gamma \vdash_{wf} \mathit{v1} : \mathit{B\text{-}pair\ b1\ b2} \\
]\!] \Longrightarrow \\
\quad \Theta\ ;\ \mathcal{B}\ ;\ \Gamma\ \vdash_{wf} \mathit{CE\text{-}snd\ v1} : \mathit{b2} \\
\\
\mid \mathit{wfCE\text{-}concatI}: [\![\\
\quad \Theta\ ;\ \mathcal{B}\ ;\ \Gamma \vdash_{wf} \mathit{v1} : \mathit{B\text{-}bitvec}\ ; \\
\quad \Theta\ ;\ \mathcal{B}\ ;\ \Gamma \vdash_{wf} \mathit{v2} : \mathit{B\text{-}bitvec} \\
]\!] \Longrightarrow \\
\quad \Theta\ ;\ \mathcal{B}\ ;\ \Gamma\ \vdash_{wf} \mathit{CE\text{-}concat\ v1\ v2} : \mathit{B\text{-}bitvec} \\
\\
\mid \mathit{wfCE\text{-}lenI}: [\![\\
\quad \Theta\ ;\ \mathcal{B}\ ;\ \Gamma \vdash_{wf} \mathit{v1} : \mathit{B\text{-}bitvec} \\
]\!] \Longrightarrow \\
\quad \Theta\ ;\ \mathcal{B}\ ;\ \Gamma\ \vdash_{wf} \mathit{CE\text{-}len\ v1} : \mathit{B\text{-}int} \\
\\
\mid \mathit{wfTI} : [\![\\
\quad \mathit{atom}\ z\ \sharp\ (\Theta, \mathcal{B}, \Gamma)\ ; \\
\quad \Theta\ ;\ \mathcal{B} \vdash_{wf} b; \\
\quad \Theta\ ;\ \mathcal{B}\ ;\ (z,b,\mathit{C\text{-}true})\ \#_{\Gamma}\ \Gamma \vdash_{wf} c \\
]\!] \Longrightarrow \\
\quad \Theta\ ;\ \mathcal{B}\ ;\ \Gamma \vdash_{wf} \{\!|\ z : b \mid c\ |\!\} \\
\\
\mid \mathit{wfC\text{-}eqI}: [\![\\
\qquad \Theta\ ;\ \mathcal{B}\ ;\ \Gamma\ \vdash_{wf} \mathit{e1}\ : b\ ; \\
\qquad \Theta\ ;\ \mathcal{B}\ ;\ \Gamma\ \vdash_{wf} \mathit{e2}\ : b\ ]\!] \Longrightarrow \\
\qquad \Theta\ ;\ \mathcal{B}\ ;\ \Gamma \vdash_{wf} \mathit{C\text{-}eq\ e1\ e2} \\
\mid \mathit{wfC\text{-}trueI}:\ \Theta\ ;\ \mathcal{B} \vdash_{wf} \Gamma\ \Longrightarrow \Theta\ ;\ \mathcal{B}\ ;\ \Gamma\ \vdash_{wf} \mathit{C\text{-}true} \\
\mid \mathit{wfC\text{-}falseI}:\ \Theta\ ;\ \mathcal{B} \vdash_{wf} \Gamma\ \Longrightarrow \Theta\ ;\ \mathcal{B}\ ;\ \Gamma \vdash_{wf} \mathit{C\text{-}false} \\
\\
\mid \mathit{wfC\text{-}conjI}: [\![\ \Theta\ ;\ \mathcal{B}\ ;\ \Gamma \vdash_{wf} \mathit{c1}\ ;\ \Theta\ ;\ \mathcal{B}\ ;\ \Gamma \vdash_{wf} \mathit{c2}\ ]\!] \Longrightarrow \Theta\ ;\ \mathcal{B}\ ;\ \Gamma \vdash_{wf} \mathit{C\text{-}conj\ c1\ c2} \\
\mid \mathit{wfC\text{-}disjI}: [\![\ \Theta\ ;\ \mathcal{B}\ ;\ \Gamma \vdash_{wf} \mathit{c1}\ ;\ \Theta\ ;\ \mathcal{B}\ ;\ \Gamma \vdash_{wf} \mathit{c2}\ ]\!] \Longrightarrow \Theta\ ;\ \mathcal{B}\ ;\ \Gamma \vdash_{wf} \mathit{C\text{-}disj\ c1\ c2} \\
\mid \mathit{wfC\text{-}notI}: [\![\ \Theta\ ;\ \mathcal{B}\ ;\ \Gamma \vdash_{wf} \mathit{c1}\ ]\!] \Longrightarrow \Theta\ ;\ \mathcal{B}\ ;\ \Gamma \vdash_{wf} \mathit{C\text{-}not\ c1} \\
\mid \mathit{wfC\text{-}impI}: [\![\ \Theta\ ;\ \mathcal{B}\ ;\ \Gamma \vdash_{wf} \mathit{c1}\ ; \\
\qquad \Theta\ ;\ \mathcal{B}\ ;\ \Gamma\ \vdash_{wf} \mathit{c2}\ ]\!] \Longrightarrow \Theta\ ;\ \mathcal{B}\ ;\ \Gamma \vdash_{wf} \mathit{C\text{-}imp\ c1\ c2} \\
\\
\mid \mathit{wfG\text{-}nilI}:\ \vdash_{wf} \Theta\ \Longrightarrow \Theta\ ;\ \mathcal{B} \vdash_{wf} \mathit{GNil} \\
\mid \mathit{wfG\text{-}cons1I}: [\![\ c \notin \{\ \mathit{TRUE}, \mathit{FALSE}\ \}\ ; \\
\qquad \Theta\ ;\ \mathcal{B}\ \vdash_{wf} \Gamma\ ; \\
\qquad \mathit{atom}\ x\ \sharp\ \Gamma\ ; \\
\qquad \Theta\ \ ;\ \mathcal{B}\ ;\ (x,b,\mathit{C\text{-}true})\#_{\Gamma}\Gamma \vdash_{wf} c\ ;\ \mathit{wfB}\ \Theta\ \mathcal{B}\ b \\
\quad]\!] \Longrightarrow\ \Theta\ ;\ \mathcal{B}\ \vdash_{wf} ((x,b,c)\#_{\Gamma}\Gamma) \\
\mid \mathit{wfG\text{-}cons2I}: [\![\ c \in \{\ \mathit{TRUE}, \mathit{FALSE}\ \}\ ; \\
\qquad \Theta\ ;\ \mathcal{B}\ \vdash_{wf} \Gamma\ ; \\
\qquad \mathit{atom}\ x\ \sharp\ \Gamma\ ; \\
\qquad \mathit{wfB}\ \Theta\ \mathcal{B}\ b \\
\quad]\!] \Longrightarrow\ \Theta\ ;\ \mathcal{B} \vdash_{wf} ((x,b,c)\#_{\Gamma}\Gamma) \\
\\
\\
\mid \mathit{wfTh\text{-}emptyI}:\ \vdash_{wf} [\,]
\end{array}
$$

| *wfTh-consI*: ⟦
(*name-of-type tdef*) ∉ *name-of-type* ‘ *set* Θ ;
⊢$_{wf}$ Θ ;
Θ ⊢$_{wf}$ *tdef* ⟧ ⟹ ⊢$_{wf}$ *tdef*#Θ

| *wfTD-simpleI*: ⟦
Θ ; {||} ; *GNil* ⊢$_{wf}$ *lst*
⟧ ⟹
Θ ⊢$_{wf}$ (*AF-typedef s lst*)

| *wfTD-poly*: ⟦
Θ ; {|*bv*|} ; *GNil* ⊢$_{wf}$ *lst*
⟧ ⟹
Θ ⊢$_{wf}$ (*AF-typedef-poly s bv lst*)

| *wfTs-nil*: Θ ; $\mathcal{B}$ ⊢$_{wf}$ Γ ⟹ Θ ; $\mathcal{B}$; Γ ⊢$_{wf}$ []::(*string*∗τ) *list*
| *wfTs-cons*: ⟦ Θ ; $\mathcal{B}$; Γ ⊢$_{wf}$ τ ;
dc ∉ *fst* ‘ *set ts*;
Θ ; $\mathcal{B}$; Γ ⊢$_{wf}$ *ts*::(*string*∗τ) *list* ⟧ ⟹ Θ ; $\mathcal{B}$; Γ ⊢$_{wf}$ ((*dc*,τ)#*ts*)

inductive-cases *wfC-elims*:
Θ ; $\mathcal{B}$; Γ ⊢$_{wf}$ *C-true*
Θ ; $\mathcal{B}$; Γ ⊢$_{wf}$ *C-false*
Θ ; $\mathcal{B}$; Γ ⊢$_{wf}$ *C-eq e1 e2*
Θ ; $\mathcal{B}$; Γ ⊢$_{wf}$ *C-conj c1 c2*
Θ ; $\mathcal{B}$; Γ ⊢$_{wf}$ *C-disj c1 c2*
Θ ; $\mathcal{B}$; Γ ⊢$_{wf}$ *C-not c1*
Θ ; $\mathcal{B}$; Γ ⊢$_{wf}$ *C-imp c1 c2*

inductive-cases *wfV-elims*:
Θ ; $\mathcal{B}$; Γ ⊢$_{wf}$ *V-var x* : *b*
Θ ; $\mathcal{B}$; Γ ⊢$_{wf}$ *V-lit l* : *b*
Θ ; $\mathcal{B}$; Γ ⊢$_{wf}$ *V-pair v1 v2* : *b*
Θ ; $\mathcal{B}$; Γ ⊢$_{wf}$ *V-cons tyid dc v* : *b*
Θ ; $\mathcal{B}$; Γ ⊢$_{wf}$ *V-consp tyid dc b v* : *b*′

inductive-cases *wfCE-elims*:
Θ ; $\mathcal{B}$; Γ ⊢$_{wf}$ *CE-val v* : *b*
Θ ; $\mathcal{B}$; Γ ⊢$_{wf}$ *CE-op Plus v1 v2* : *b*
Θ ; $\mathcal{B}$; Γ ⊢$_{wf}$ *CE-op LEq v1 v2* : *b*
Θ ; $\mathcal{B}$; Γ ⊢$_{wf}$ *CE-fst v1* : *b*
Θ ; $\mathcal{B}$; Γ ⊢$_{wf}$ *CE-snd v1* : *b*
Θ ; $\mathcal{B}$; Γ ⊢$_{wf}$ *CE-concat v1 v2* : *b*
Θ ; $\mathcal{B}$; Γ ⊢$_{wf}$ *CE-len v1* : *b*
Θ ; $\mathcal{B}$; Γ ⊢$_{wf}$ *CE-op opp v1 v2* : *b*

inductive-cases *wfT-elims*:
Π; $\mathcal{B}$; Γ ⊢$_{wf}$ τ::τ
Π; $\mathcal{B}$; Γ ⊢$_{wf}$ {| *z* : *b* | *c* |}

inductive-cases *wfG-elims*:

Π ; $\mathcal{B} \vdash_{wf}$ *GNil*
Π ; $\mathcal{B} \vdash_{wf}$ $(x,b,c)\#_\Gamma\Gamma$
Π ; $\mathcal{B} \vdash_{wf}$ $(x,b,TRUE)\#_\Gamma\Gamma$
Π ; $\mathcal{B} \vdash_{wf}$ $(x,b,FALSE)\#_\Gamma\Gamma$

inductive-cases *wfTh-elims*:
$\vdash_{wf}$ []
$\vdash_{wf}$ *td*#Π

inductive-cases *wfTD-elims*:
$\Theta \vdash_{wf}$ (*AF-typedef s lst*)
$\Theta \vdash_{wf}$ (*AF-typedef-poly s bv lst*)

inductive-cases *wfTs-elims*:
P ; $\mathcal{B}$; *GNil* $\vdash_{wf}$ ([]::((*string*$*\tau$) *list*))
P ; $\mathcal{B}$; *GNil* $\vdash_{wf}$ ((*t*#*ts*)::((*string*$*\tau$) *list*))

inductive-cases *wfB-elims*:
Θ ; $\mathcal{B} \vdash_{wf}$ *B-pair b1 b2*
Θ ; $\mathcal{B} \vdash_{wf}$ *B-id s*
Θ ; $\mathcal{B} \vdash_{wf}$ *B-app s b*

equivariance *wfV*

nominal-inductive *wfV*
avoids *wfV-conspI*: *bv* | *wfTI*: *z*
proof(*goal-cases*)
case (*1 s bv dclist* Θ *dc x b'* *c* $\mathcal{B}$ *b* Γ *v*)

moreover hence *atom bv* ♯ *V-consp s dc b v* **using** *v.fresh fresh-prodN pure-fresh* **by** *metis*
moreover have *atom bv* ♯ *B-app s b* **using** *b.fresh fresh-prodN pure-fresh 1* **by** *metis*
ultimately show *?case* **using** *b.fresh v.fresh pure-fresh fresh-star-def fresh-prodN* **by** *fastforce*
next
case (*2 s bv dclist* Θ *dc x b'* *c* $\mathcal{B}$ *b* Γ *v*)
then show *?case* **by** *auto*

next
case (*3 z* Γ Θ $\mathcal{B}$ *b c*)
then show *?case* **using** τ*.fresh fresh-star-def fresh-prodN* **by** *fastforce*
next
case (*4 z* Γ Θ $\mathcal{B}$ *b c*)
then show *?case* **by** *auto*
qed

inductive
wfE :: $\Theta \Rightarrow \Phi \Rightarrow \mathcal{B} \Rightarrow \Gamma \Rightarrow \Delta \Rightarrow e \Rightarrow b \Rightarrow$ *bool* (- ; - ; - ; - ; - $\vdash_{wf}$ - : - [*50*,*50*,*50*] *50*) **and**

wfS :: $\Theta \Rightarrow \Phi \Rightarrow \mathcal{B} \Rightarrow \Gamma \Rightarrow \Delta \Rightarrow s \Rightarrow b \Rightarrow$ *bool* (- ; - ; - ; - ; - $\vdash_{wf}$ - : - [*50*,*50*,*50*] *50*) **and**
wfCS :: $\Theta \Rightarrow \Phi \Rightarrow \mathcal{B} \Rightarrow \Gamma \Rightarrow \Delta \Rightarrow$ *tyid* $\Rightarrow$ *string* $\Rightarrow \tau \Rightarrow$ *branch-s* $\Rightarrow b \Rightarrow$ *bool* (- ; - ; - ; - ; - ; - ; - ; - $\vdash_{wf}$ - : - [*50*,*50*,*50*,*50*,*50*] *50*) **and**
wfCSS :: $\Theta \Rightarrow \Phi \Rightarrow \mathcal{B} \Rightarrow \Gamma \Rightarrow \Delta \Rightarrow$ *tyid* $\Rightarrow$ (*string* $*$ τ) *list* $\Rightarrow$ *branch-list* $\Rightarrow b \Rightarrow$ *bool* (- ; - ; - ; - ; - ; - ; - ; - $\vdash_{wf}$ - : - [*50*,*50*,*50*,*50*,*50*] *50*) **and**
wfPhi :: $\Theta \Rightarrow \Phi \Rightarrow$ *bool* (- $\vdash_{wf}$ - [*50*,*50*] *50*) **and**
wfD :: $\Theta \Rightarrow \mathcal{B} \Rightarrow \Gamma \Rightarrow \Delta \Rightarrow$ *bool* (- ; - ; - $\vdash_{wf}$ - [*50*,*50*] *50*) **and**
wfFTQ :: $\Theta \Rightarrow \Phi \Rightarrow$ *fun-typ-q* $\Rightarrow$ *bool* (- ; - $\vdash_{wf}$ - [*50*] *50*) **and**
wfFT :: $\Theta \Rightarrow \Phi \Rightarrow \mathcal{B} \Rightarrow$ *fun-typ* $\Rightarrow$ *bool* (- ; - ; - $\vdash_{wf}$ - [*50*] *50*) **where**

wfE-valI : ⟦ (
$\Theta \vdash_{wf} \Phi$) ;
Θ ; $\mathcal{B}$; $\Gamma \vdash_{wf} \Delta$;
Θ ; $\mathcal{B}$; $\Gamma \vdash_{wf} v$: *b*
⟧ ⟹
Θ ; Φ ; $\mathcal{B}$; Γ ; Δ $\vdash_{wf}$ *AE-val v* : *b*

| *wfE-plusI*: ⟦
$\Theta \vdash_{wf} \Phi$;
Θ ; $\mathcal{B}$; $\Gamma \vdash_{wf} \Delta$;
Θ ; $\mathcal{B}$; $\Gamma \vdash_{wf}$ *v1* : *B-int*;
Θ ; $\mathcal{B}$; $\Gamma \vdash_{wf}$ *v2* : *B-int*
⟧ ⟹
Θ ; Φ ; $\mathcal{B}$; Γ ; Δ $\vdash_{wf}$ *AE-op Plus v1 v2* : *B-int*

| *wfE-leqI*:⟦
$\Theta \vdash_{wf} \Phi$;
Θ ; $\mathcal{B}$; $\Gamma \vdash_{wf} \Delta$;
Θ ; $\mathcal{B}$; $\Gamma \vdash_{wf}$ *v1* : *B-int*;
Θ ; $\mathcal{B}$; $\Gamma \vdash_{wf}$ *v2* : *B-int*
⟧ ⟹
Θ ; Φ ; $\mathcal{B}$; Γ ; Δ $\vdash_{wf}$ *AE-op LEq v1 v2* : *B-bool*

| *wfE-fstI*: ⟦
$\Theta \vdash_{wf} \Phi$;
Θ ; $\mathcal{B}$; $\Gamma \vdash_{wf} \Delta$;
Θ ; $\mathcal{B}$; $\Gamma \vdash_{wf}$ *v1* : *B-pair b1 b2*
⟧ ⟹
Θ ; Φ ; $\mathcal{B}$; Γ ; $\Delta \vdash_{wf}$ *AE-fst v1* : *b1*

| *wfE-sndI*: ⟦
$\Theta \vdash_{wf} \Phi$;
Θ ; $\mathcal{B}$; $\Gamma \vdash_{wf} \Delta$;
Θ ; $\mathcal{B}$; $\Gamma \vdash_{wf}$ *v1* : *B-pair b1 b2*
⟧ ⟹
Θ ; Φ ; $\mathcal{B}$; Γ ; $\Delta \vdash_{wf}$ *AE-snd v1* : *b2*

| *wfE-concatI*: ⟦
$\Theta \vdash_{wf} \Phi$;
Θ ; $\mathcal{B}$; $\Gamma \vdash_{wf} \Delta$;
Θ ; $\mathcal{B}$; $\Gamma \vdash_{wf}$ *v1* : *B-bitvec*;
Θ ; $\mathcal{B}$; $\Gamma \vdash_{wf}$ *v2* : *B-bitvec*

$]\!] \Longrightarrow$
Θ ; Φ ; $\mathcal{B}$; Γ; Δ $\vdash_{wf}$ *AE-concat v1 v2* : *B-bitvec*

| *wfE-splitI*: $[\![$
$\Theta \vdash_{wf} \Phi$;
Θ ; $\mathcal{B}$; $\Gamma \vdash_{wf} \Delta$;
Θ ; $\mathcal{B}$; $\Gamma \vdash_{wf}$ *v1* : *B-bitvec*;
Θ ; $\mathcal{B}$; $\Gamma \vdash_{wf}$ *v2* : *B-int*
$]\!] \Longrightarrow$
Θ ; Φ ; $\mathcal{B}$; Γ; Δ $\vdash_{wf}$ *AE-split v1 v2* : *B-pair B-bitvec B-bitvec*

| *wfE-lenI*: $[\![$
$\Theta \vdash_{wf} \Phi$;
Θ ; $\mathcal{B}$; $\Gamma \vdash_{wf} \Delta$;
Θ ; $\mathcal{B}$; $\Gamma \vdash_{wf}$ *v1* : *B-bitvec*
$]\!] \Longrightarrow$
Θ ; Φ ; $\mathcal{B}$; Γ ; Δ $\vdash_{wf}$ *AE-len v1* : *B-int*

| *wfE-appI*: $[\![$
$\Theta \vdash_{wf} \Phi$;
Θ ; $\mathcal{B}$; $\Gamma \vdash_{wf} \Delta$;
Some (*AF-fundef f* (*AF-fun-typ-none* (*AF-fun-typ x b c* τ *s*))) = *lookup-fun* Φ *f* ;
Θ ; $\mathcal{B}$; $\Gamma \vdash_{wf}$ *v* : *b*
$]\!] \Longrightarrow$
Θ ; Φ ; $\mathcal{B}$; Γ ; $\Delta \vdash_{wf}$ *AE-app f v* : *b-of* τ

| *wfE-appPI*: $[\![$
$\Theta \vdash_{wf} \Phi$;
Θ ; $\mathcal{B}$; $\Gamma \vdash_{wf} \Delta$;
Θ ; $\mathcal{B}$ $\vdash_{wf}$ b';
atom bv $\sharp$ (Φ, Θ, $\mathcal{B}$, Γ, Δ, b', *v*, (*b-of* τ)$[bv::=b']_b$);
Some (*AF-fundef f* (*AF-fun-typ-some bv* (*AF-fun-typ x b c* τ *s*))) = *lookup-fun* Φ *f*;
Θ ; $\mathcal{B}$; $\Gamma \vdash_{wf}$ *v* : ($b[bv::=b']_b$)
$]\!] \Longrightarrow$
Θ ; Φ ; $\mathcal{B}$; Γ ; $\Delta \vdash_{wf}$ (*AE-appP f b' v*) : ((*b-of* τ)$[bv::=b']_b$)

| *wfE-mvarI*: $[\![$
$\Theta \vdash_{wf} \Phi$;
Θ ; $\mathcal{B}$; $\Gamma \vdash_{wf} \Delta$;
(u,τ) $\in$ *setD* Δ
$]\!] \Longrightarrow$
Θ ; Φ ; $\mathcal{B}$; Γ ; $\Delta \vdash_{wf}$ *AE-mvar u* : *b-of* τ

| *wfS-valI*: $[\![$
$\Theta \vdash_{wf} \Phi$;
Θ ; $\mathcal{B}$; $\Gamma \vdash_{wf}$ *v* : *b* ;
Θ ; $\mathcal{B}$; $\Gamma \vdash_{wf} \Delta$
$]\!] \Longrightarrow$
Θ ; Φ ; $\mathcal{B}$; Γ ; $\Delta \vdash_{wf}$ (*AS-val v*) : *b*

| *wfS-letI*: $[\![$
wfE Θ Φ $\mathcal{B}$ Γ Δ *e b'* ;

Θ ; Φ ; $\mathcal{B}$; (x,b',*C-true*) $\#_\Gamma$ Γ ; $\Delta \vdash_{wf} s : b$;
Θ ; $\mathcal{B}$; $\Gamma \vdash_{wf} \Delta$;
atom $x \sharp (\Phi, \Theta, \mathcal{B}, \Gamma, \Delta, e, b)$
$\rrbracket \Longrightarrow$
Θ ; Φ ; $\mathcal{B}$; Γ ; $\Delta \vdash_{wf}$ *LET* $x = e$ *IN* $s : b$

| *wfS-assertI*: $[\![$
Θ ; Φ ; $\mathcal{B}$; (x,*B-bool*,c) $\#_\Gamma$ Γ ; $\Delta \vdash_{wf} s : b$;
Θ ; $\mathcal{B}$; $\Gamma \vdash_{wf} c$;
Θ ; $\mathcal{B}$; $\Gamma \vdash_{wf} \Delta$;
atom $x \sharp (\Phi, \Theta, \mathcal{B}, \Gamma, \Delta, c, b, s)$
$\rrbracket \Longrightarrow$
Θ ; Φ ; $\mathcal{B}$; Γ ; $\Delta \vdash_{wf}$ *ASSERT* c *IN* $s : b$

| *wfS-let2I*: $[\![$ Θ ; Φ ; $\mathcal{B}$; Γ ; $\Delta \vdash_{wf}$ *s1* : *b-of* τ ;
Θ ; $\mathcal{B}$; $\Gamma \vdash_{wf} \tau$;
Θ ; Φ ; $\mathcal{B}$; (x,*b-of* τ,*C-true*) $\#_\Gamma$ Γ ; $\Delta \vdash_{wf}$ *s2* : b ;
atom $x \sharp (\Phi, \Theta, \mathcal{B}, \Gamma, \Delta,$ *s1*, b,τ)
$\rrbracket \Longrightarrow$
Θ ; Φ ; $\mathcal{B}$; Γ ; $\Delta \vdash_{wf}$ *LET* $x : \tau =$ *s1 IN s2* : b
| *wfS-ifI*: $[\![$ Θ ; $\mathcal{B}$; $\Gamma \vdash_{wf} v$: *B-bool*;
Θ ; Φ ; $\mathcal{B}$; Γ ; $\Delta \vdash_{wf}$ *s1* : b ;
Θ ; Φ ; $\mathcal{B}$; Γ ; $\Delta \vdash_{wf}$ *s2* : b $\rrbracket \Longrightarrow$
Θ ; Φ ; $\mathcal{B}$; Γ ; $\Delta \vdash_{wf}$ *IF v THEN s1 ELSE s2* : b

| *wfS-varI* : $[\![$ *wfT* Θ $\mathcal{B}$ Γ τ ;
Θ ; $\mathcal{B}$; $\Gamma \vdash_{wf} v$: *b-of* τ;
atom $u \sharp (\Phi, \Theta, \mathcal{B}, \Gamma, \Delta, \tau, v, b)$;
Θ ; Φ ; $\mathcal{B}$; Γ ; (u,τ)$\#_\Delta\Delta \vdash_{wf} s : b$ $\rrbracket \Longrightarrow$
Θ ; Φ ; $\mathcal{B}$; Γ ; $\Delta \vdash_{wf}$ *VAR* $u : \tau = v$ *IN* $s : b$

| *wfS-assignI*: $[\![$ (u,τ) $\in$ *setD* Δ ;
Θ ; $\mathcal{B}$; $\Gamma \vdash_{wf} \Delta$;
$\Theta \vdash_{wf} \Phi$;
Θ ; $\mathcal{B}$; $\Gamma \vdash_{wf} v$: *b-of* τ $\rrbracket \Longrightarrow$
Θ ; Φ ; $\mathcal{B}$; Γ ; $\Delta \vdash_{wf} u ::= v$: *B-unit*

| *wfS-whileI*: $[\![$ Θ ; Φ ; $\mathcal{B}$; Γ ; $\Delta \vdash_{wf}$ *s1* : *B-bool* ;
Θ ; Φ ; $\mathcal{B}$; Γ ; $\Delta \vdash_{wf}$ *s2* : b]$\!\rrbracket \Longrightarrow$
Θ ; Φ ; $\mathcal{B}$; Γ ; $\Delta \vdash_{wf}$ *WHILE s1 DO* { *s2* } : b

| *wfS-seqI*: $[\![$ Θ ; Φ ; $\mathcal{B}$; Γ ; $\Delta \vdash_{wf}$ *s1* : *B-unit* ;
Θ ; Φ ; $\mathcal{B}$; Γ ; $\Delta \vdash_{wf}$ *s2* : b $\rrbracket \Longrightarrow$
Θ ; Φ ; $\mathcal{B}$; Γ ; $\Delta \vdash_{wf}$ *s1* ;; *s2* : b

| *wfS-matchI*: $[\![$ *wfV* Θ $\mathcal{B}$ Γ v (*B-id tid*) ;
(*AF-typedef tid dclist*) $\in$ *set* Θ;
wfD Θ $\mathcal{B}$ Γ Δ ;
$\Theta \vdash_{wf} \Phi$;
Θ ; Φ ; $\mathcal{B}$; Γ ; Δ ; *tid* ; *dclist* $\vdash_{wf}$ *cs* : b $\rrbracket \Longrightarrow \Theta$; Φ ; $\mathcal{B}$; Γ ; $\Delta \vdash_{wf}$ *AS-match v cs* : b

| *wfS-branchI*: $[\![$ Θ ; Φ ; $\mathcal{B}$; (x,*b-of* τ,*C-true*) $\#_\Gamma$ Γ ; $\Delta \vdash_{wf} s : b$;

$atom\ x \sharp (\Phi, \Theta, \mathcal{B}, \Gamma, \Delta, \Gamma, \tau)$;
$\Theta ; \mathcal{B} ; \Gamma \vdash_{wf} \Delta$
$\rrbracket \Longrightarrow$
$\Theta ; \Phi ; \mathcal{B} ; \Gamma ; \Delta ; tid ; dc ; \tau \vdash_{wf} dc\ x \Rightarrow s : b$

| *wfS-finalI*: $\llbracket$
$\Theta ; \Phi ; \mathcal{B} ; \Gamma ; \Delta ; tid ; dc ; t \vdash_{wf} cs : b$
$\rrbracket \Longrightarrow$
$\Theta ; \Phi ; \mathcal{B} ; \Gamma ; \Delta ; tid ; [(dc,t)] \vdash_{wf} AS\text{-}final\ cs : b$

| *wfS-cons*: $\llbracket$
$\Theta ; \Phi ; \mathcal{B} ; \Gamma ; \Delta ; tid ; dc ; t \vdash_{wf} cs : b$;
$\Theta ; \Phi ; \mathcal{B} ; \Gamma ; \Delta ; tid ; dclist \vdash_{wf} css : b$
$\rrbracket \Longrightarrow$
$\Theta ; \Phi ; \mathcal{B} ; \Gamma ; \Delta ; tid ; (dc,t)\#dclist \vdash_{wf} AS\text{-}cons\ cs\ css : b$

| *wfD-emptyI*: $\Theta ; \mathcal{B} \vdash_{wf} \Gamma \Longrightarrow \Theta ; \mathcal{B} ; \Gamma \vdash_{wf} []_{\Delta}$
| *wfD-cons*: $\llbracket \Theta ; \mathcal{B} ; \Gamma \vdash_{wf} \Delta{::}\Delta$;
$\Theta ; \mathcal{B} ; \Gamma \vdash_{wf} \tau$;
$u \notin fst\ `\ setD\ \Delta \rrbracket \Longrightarrow \Theta ; \mathcal{B} ; \Gamma \vdash_{wf} ((u,\tau) \#_{\Delta} \Delta)$

| *wfPhi-emptyI*: $\vdash_{wf} \Theta \Longrightarrow \Theta \vdash_{wf} []$
| *wfPhi-consI*: $\llbracket$
$f \notin name\text{-}of\text{-}fun\ `\ set\ \Phi$;
$\Theta ; \Phi \vdash_{wf} ft$;
$\Theta \vdash_{wf} \Phi$
$\rrbracket \Longrightarrow$
$\Theta \vdash_{wf} ((AF\text{-}fundef\ f\ ft)\#\Phi)$
| *wfFTNone*: $\Theta ; \Phi ; \{||\} \vdash_{wf} ft \Longrightarrow \Theta ; \Phi \vdash_{wf} AF\text{-}fun\text{-}typ\text{-}none\ ft$
| *wfFTSome*: $\Theta ; \Phi ; \{|\ bv\ |\} \vdash_{wf} ft \Longrightarrow \Theta ; \Phi \vdash_{wf} AF\text{-}fun\text{-}typ\text{-}some\ bv\ ft$
| *wfFTI*: $\llbracket$
$\Theta ; B \vdash_{wf} b$;
$\Theta ; \Phi ; B ; (x,b,c) \#_{\Gamma} GNil ; []_{\Delta} \vdash_{wf} s : b\text{-}of\ \tau$;
$supp\ s \subseteq \{atom\ x\}$;
$supp\ c \subseteq \{\ atom\ x\ \}$;
$\Theta ; B ; (x,b,c) \#_{\Gamma} GNil \vdash_{wf} \tau$
$\rrbracket \Longrightarrow$
$\Theta ; \Phi ; B \vdash_{wf} (AF\text{-}fun\text{-}typ\ x\ b\ c\ \tau\ s)$

inductive-cases *wfE-elims*:
$\Theta ; \Phi ; \mathcal{B} ; \Gamma ; \Delta \vdash_{wf} AE\text{-}val\ v : b$
$\Theta ; \Phi ; \mathcal{B} ; \Gamma ; \Delta \vdash_{wf} AE\text{-}op\ Plus\ v1\ v2 : b$
$\Theta ; \Phi ; \mathcal{B} ; \Gamma ; \Delta \vdash_{wf} AE\text{-}op\ LEq\ v1\ v2 : b$
$\Theta ; \Phi ; \mathcal{B} ; \Gamma ; \Delta \vdash_{wf} AE\text{-}fst\ v1 : b$
$\Theta ; \Phi ; \mathcal{B} ; \Gamma ; \Delta \vdash_{wf} AE\text{-}snd\ v1 : b$
$\Theta ; \Phi ; \mathcal{B} ; \Gamma ; \Delta \vdash_{wf} AE\text{-}concat\ v1\ v2 : b$
$\Theta ; \Phi ; \mathcal{B} ; \Gamma ; \Delta \vdash_{wf} AE\text{-}len\ v1 : b$
$\Theta ; \Phi ; \mathcal{B} ; \Gamma ; \Delta \vdash_{wf} AE\text{-}op\ opp\ v1\ v2 : b$
$\Theta ; \Phi ; \mathcal{B} ; \Gamma ; \Delta \vdash_{wf} AE\text{-}app\ f\ v$: b
$\Theta ; \Phi ; \mathcal{B} ; \Gamma ; \Delta \vdash_{wf} AE\text{-}appP\ f\ b'\ v$: b
$\Theta ; \Phi ; \mathcal{B} ; \Gamma ; \Delta \vdash_{wf} AE\text{-}mvar\ u : b$

inductive-cases *wfCS-elims*:
Θ ; Φ ; $\mathcal{B}$; Γ ; Δ ; *tid* ; *dc* ; $t \vdash_{wf}$ (*cs*::*branch-s*) : *b*
Θ ; Φ ; $\mathcal{B}$; Γ ; Δ ; *tid* ; *dc* $\vdash_{wf}$ (*cs*::*branch-list*) : *b*

inductive-cases *wfPhi-elims*:
$\Theta \vdash_{wf} []$
$\Theta \vdash_{wf}$ ((*AF-fundef f ft*)#Π)
$\Theta \vdash_{wf}$ (*fd*#Φ::Φ)

declare[[*simproc del*: *alpha-lst*]]

inductive-cases *wfFTQ-elims*:
Θ ; $\Phi \vdash_{wf}$ *AF-fun-typ-none ft*
Θ ; $\Phi \vdash_{wf}$ *AF-fun-typ-some bv ft*
Θ ; $\Phi \vdash_{wf}$ *AF-fun-typ-some bv* (*AF-fun-typ x b c* τ *s*)

inductive-cases *wfFT-elims*:
Θ ; Φ ; $\mathcal{B} \vdash_{wf}$ *AF-fun-typ x b c* τ *s*

declare[[*simproc add*: *alpha-lst*]]

inductive-cases *wfD-elims*:
Π ; $\mathcal{B}$; (Γ::Γ) $\vdash_{wf}$ $[]_\Delta$
Π ; $\mathcal{B}$; (Γ::Γ) $\vdash_{wf}$ (u,τ) $\#_\Delta$ Δ::Δ

equivariance *wfE*

nominal-inductive *wfE*
avoids *wfE-appPI*: *bv* | *wfS-varI*: *u* | *wfS-letI*: *x* | *wfS-let2I*: *x* | *wfS-branchI*: *x* | *wfS-assertI*: *x*

proof(*goal-cases*)
case (*1* Θ Φ $\mathcal{B}$ Γ Δ *b'* *bv v* τ *f x b c s*)
moreover hence *atom bv* $\sharp$ *AE-appP f b' v* **using** *pure-fresh fresh-prodN e.fresh* **by** *auto*
ultimately show *?case* **using** *fresh-star-def* **by** *fastforce*
next
case (*2* Θ Φ $\mathcal{B}$ Γ Δ *b'* *bv v* τ *f x b c s*)
then show *?case* **by** *auto*
next

case (*3* Φ Θ $\mathcal{B}$ Γ Δ *e b' x s b*)
moreover hence *atom x* $\sharp$ *LET x = e IN s* **using** *fresh-prodN* **by** *auto*
ultimately show *?case* **using** *fresh-prodN fresh-star-def* **by** *fastforce*
next
case (*4* Φ Θ $\mathcal{B}$ Γ Δ *e b' x s b*)
then show *?case* **by** *auto*
next
case (*5* Θ Φ $\mathcal{B}$ *x c* Γ Δ *s b*)
hence *atom x* $\sharp$ *ASSERT c IN s* **using** *s-branch-s-branch-list.fresh* **by** *auto*

then show *?case* **using** *fresh-prodN fresh-star-def 5* **by** *fastforce*
next
case (*6* Θ Φ $\mathcal{B}$ *x c* Γ Δ *s b*)
then show *?case* **by** *auto*
next
case (*7* Φ Θ $\mathcal{B}$ Γ Δ *s1* τ *x s2 b*)
hence *atom x* ♯ τ ∧ *atom x* ♯ *s1* **using** *fresh-prodN* **by** *metis*
moreover hence *atom x* ♯ *LET x* : τ = *s1 IN s2*
using *s-branch-s-branch-list.fresh*(*3*)[*of atom x x* τ *s1 s2*] *fresh-prodN* **by** *simp*
ultimately show *?case* **using** *fresh-prodN fresh-star-def 7* **by** *fastforce*
next
case (*8* Φ Θ $\mathcal{B}$ Γ Δ *s1* τ *x s2 b*)
then show *?case* **by** *auto*
next
case (*9* Θ $\mathcal{B}$ Γ τ *v u* Φ Δ *b s*)
moreover hence *atom u* ♯ *AS-var u* τ *v s* **using** *fresh-prodN s-branch-s-branch-list.fresh* **by** *simp*
ultimately show *?case* **using** *fresh-star-def fresh-prodN s-branch-s-branch-list.fresh* **by** *fastforce*
next
case (*10* Θ $\mathcal{B}$ Γ τ *v u* Φ Δ *b s*)
then show *?case* **by** *auto*
next
case (*11* Φ Θ $\mathcal{B}$ *x* τ Γ Δ *s b tid dc*)
moreover have *atom x* ♯ (*dc x* ⇒ *s*) **using** *pure-fresh s-branch-s-branch-list.fresh* **by** *auto*
ultimately show *?case* **using** *fresh-prodN fresh-star-def pure-fresh* **by** *fastforce*
next
case (*12* Φ Θ $\mathcal{B}$ *x* τ Γ Δ *s b tid dc*)
then show *?case* **by** *auto*
qed

inductive *wfVDs* :: *var-def list* ⇒ *bool* **where**

wfVDs-nilI: *wfVDs* []

| *wfVDs-consI*: ⟦
atom u ♯ *ts*;
wfV ([]::Θ) {||} *GNil v* (*b-of* τ);
wfT ([]::Θ) {||} *GNil* τ;
wfVDs ts
⟧ ⟹ *wfVDs* ((*AV-def u* τ *v*)#*ts*)

equivariance *wfVDs*
nominal-inductive *wfVDs* .

end

hide-const *Syntax.dom*

Chapter 7

Refinement Constraint Logic

Semantics for the logic we use in the refinement constraints. It is a multi-sorted, quantifier free logic with polymorphic datatypes and linear arithmetic. We could have modelled by using one of the encodings to FOL however we wanted to explore using a more direct model.

7.1 Evaluation and Satisfiability

7.1.1 Valuation

RCL values. This is our universe. SUt is a value for uninterpreted sort that corresponds to base type variables. For now we only need one of these universes. We wrap an smt_val inside it during a process we call 'boxing' that is introduced in the RCLModelLemmass theory

nominal-datatype *rcl-val* = *SBitvec bit list* | *SNum int* | *SBool bool* | *SPair rcl-val rcl-val* |
SCons tyid string rcl-val | *SConsp tyid string b rcl-val* |
SUnit | *SUt rcl-val*

RCL sorts. Represent our domains. The universe is the union of all of the these. S_Ut is the single uninterpreted sort. Map almost directly to base type but should have them to clearly distinguish syntax (base types) and semantics (RCL sorts)

nominal-datatype *rcl-sort* = *S-bool* | *S-int* | *S-unit* | *S-pair rcl-sort rcl-sort* | *S-id tyid* | *S-app tyid rcl-sort* | *S-bitvec* | *S-ut*

type-synonym *valuation* = (*x*,*rcl-val*) *map*

type-synonym *type-valuation* = (*bv*,*rcl-sort*) *map*

inductive *wfRCV*:: $\Theta \Rightarrow$ *rcl-val* $\Rightarrow$ *b* $\Rightarrow$ *bool* (- $\vdash$ - : - [*50*,*50*] *50*) **where**
wfRCV-BBitvecI: $P \vdash$ (*SBitvec bv*) : *B-bitvec*
| *wfRCV-BIntI*: $P \vdash$ (*SNum n*) : *B-int*
| *wfRCV-BBoolI*: $P \vdash$ (*SBool b*) : *B-bool*
| *wfRCV-BPairI*: $[\![P \vdash s1 : b1 ; P \vdash s2 : b2]\!] \Longrightarrow P \vdash$ (*SPair s1 s2*) : (*B-pair b1 b2*)
| *wfRCV-BConsI*: $[\![$ *AF-typedef s dclist* $\in$ *set* Θ;
(*dc*, $\{\!|$ *x* : *b* | *c* $|\!\}$) $\in$ *set dclist* ;
$\Theta \vdash s1 : b]\!] \Longrightarrow \Theta \vdash$(*SCons s dc s1*) : (*B-id s*)
| *wfRCV-BConsPI*:$[\![$ *AF-typedef-poly s bv dclist* $\in$ *set* Θ;

$(dc, \{\!|\ x : b\ \mid c\ |\!\}) \in set\ dclist$;
$atom\ bv \sharp (\Theta, SConsp\ s\ dc\ b'\ s1, B\text{-}app\ s\ b')$;
$\Theta \vdash s1 : b[bv::=b']_{bb}\ ]\!] \Longrightarrow \Theta \vdash (SConsp\ s\ dc\ b'\ s1) : (B\text{-}app\ s\ b')$
| *wfRCV-BUnitI*: $P \vdash SUnit : B\text{-}unit$
| *wfRCV-BVarI*: $P \vdash (SUt\ n) : (B\text{-}var\ bv)$
equivariance *wfRCV*
nominal-inductive *wfRCV*
avoids *wfRCV-BConsPI*: *bv*
proof(*goal-cases*)
case (*1 s bv dclist* Θ *dc x b c b' s1*)
then show *?case* **using** *fresh-star-def* **by** *auto*
next
case (*2 s bv dclist* Θ *dc x b c s1 b'*)
then show *?case* **by** *auto*
qed

inductive-cases *wfRCV-elims* :
wfRCV P s B-bitvec
wfRCV P s (*B-pair b1 b2*)
wfRCV P s (*B-int*)
wfRCV P s (*B-bool*)
wfRCV P s (*B-id ss*)
wfRCV P s (*B-var bv*)
wfRCV P s (*B-unit*)
wfRCV P s (*B-app tyid b*)
wfRCV P (*SBitvec bv*) *b*
wfRCV P (*SNum n*) *b*
wfRCV P (*SBool n*) *b*
wfRCV P (*SPair s1 s2*) *b*
wfRCV P (*SCons s dc s1*) *b*
wfRCV P (*SConsp s dc b' s1*) *b*
wfRCV P SUnit b
wfRCV P (*SUt s1*) *b*

thm *wfRCV-elims*(*9*)

Sometimes we want to do $P \vdash s \sim b[bv=b']$ and we want to know what b is however substitution is not injective so we can't write this in terms of *wfRCV*. So we define a relation that makes the variable and thing being substitited in explicit.

inductive *wfRCV-subst*:: $\Theta \Rightarrow rcl\text{-}val \Rightarrow b \Rightarrow (bv*b)\ option \Rightarrow bool$ **where**
wfRCV-subst-BBitvecI: *wfRCV-subst P* (*SBitvec bv*) *B-bitvec sub*
| *wfRCV-subst-BIntI*: *wfRCV-subst P* (*SNum n*) *B-int sub*
| *wfRCV-subst-BBoolI*: *wfRCV-subst P* (*SBool b*) *B-bool sub*
| *wfRCV-subst-BPairI*: $[\![$ *wfRCV-subst P s1 b1 sub* ; *wfRCV-subst P s2 b2 sub* $]\!] \Longrightarrow$ *wfRCV-subst P* (*SPair s1 s2*) (*B-pair b1 b2*) *sub*
| *wfRCV-subst-BConsI*: $[\![$ *AF-typedef s dclist* $\in$ *set* Θ;
$(dc, \{\!|\ x : b\ \mid c\ |\!\}) \in set\ dclist$;
wfRCV-subst Θ *s1 b None* $]\!] \Longrightarrow$ *wfRCV-subst* Θ (*SCons s dc s1*) (*B-id s*) *sub*
| *wfRCV-subst-BConspI*: $[\![$ *AF-typedef-poly s bv dclist* $\in$ *set* Θ;
$(dc, \{\!|\ x : b\ \mid c\ |\!\}) \in set\ dclist$;
wfRCV-subst Θ *s1* $(b[bv::=b']_{bb})$ *sub* $]\!] \Longrightarrow$ *wfRCV-subst* Θ (*SConsp s dc b' s1*) (*B-app s b'*) *sub*
| *wfRCV-subst-BUnitI*: *wfRCV-subst P SUnit B-unit sub*

| *wfRCV-subst-BVar1I*: *bvar* $\neq$ *bv* $\Longrightarrow$ *wfRCV-subst P* (*SUt n*) (*B-var bv*) (*Some* (*bvar*,*bin*))
| *wfRCV-subst-BVar2I*: $[\![$ *bvar* = *bv*; *wfRCV-subst P s bin None* $]\!] \Longrightarrow$ *wfRCV-subst P s* (*B-var bv*) (*Some* (*bvar*,*bin*))
| *wfRCV-subst-BVar3I*: *wfRCV-subst P* (*SUt n*) (*B-var bv*) *None*
equivariance *wfRCV-subst*
nominal-inductive *wfRCV-subst* .

7.1.2 Evaluation base-types

inductive *eval-b* :: *type-valuation* $\Rightarrow$ *b* $\Rightarrow$ *rcl-sort* $\Rightarrow$ *bool* (- $[\![$ - $]\!]^{\sim}$ -) **where**
v $[\![$ *B-bool* $]\!]^{\sim}$ *S-bool*
| *v* $[\![$ *B-int* $]\!]^{\sim}$ *S-int*
| *Some s* = *v bv* $\Longrightarrow$ *v* $[\![$ *B-var bv* $]\!]^{\sim}$ *s*
equivariance *eval-b*
nominal-inductive *eval-b* .

7.1.3 Wellformed Evaluation

definition *wfI* :: $\Theta \Rightarrow \Gamma \Rightarrow$ *valuation* $\Rightarrow$ *bool* (- ; - $\vdash$ -) **where**
Θ ; $\Gamma \vdash i = (\forall\ (x,b,c) \in setG\ \Gamma.\ \exists s.\ Some\ s = i\ x \wedge \Theta \vdash s : b)$

7.1.4 Evaluating Terms

nominal-function *eval-l* :: *l* $\Rightarrow$ *rcl-val* ($[\![$ - $]\!]$) **where**
$[\![$ *L-true* $]\!]$ = *SBool True*
| $[\![$ *L-false* $]\!]$ = *SBool False*
| $[\![$ *L-num n* $]\!]$ = *SNum n*
| $[\![$ *L-unit* $]\!]$ = *SUnit*
| $[\![$ *L-bitvec n* $]\!]$ = *SBitvec n*
apply(*auto simp*: *eqvt-def eval-l-graph-aux-def*)
by (*metis l.exhaust*)
nominal-termination (*eqvt*) **by** *lexicographic-order*

inductive *eval-v* :: *valuation* $\Rightarrow$ *v* $\Rightarrow$ *rcl-val* $\Rightarrow$ *bool* (- $[\![$ - $]\!]^{\sim}$ -) **where**
eval-v-litI: *i* $[\![$ *V-lit l* $]\!]^{\sim}$ $[\![$ *l* $]\!]$
| *eval-v-varI*: *Some sv* = *i x* $\Longrightarrow$ *i* $[\![$ *V-var x* $]\!]^{\sim}$ *sv*
| *eval-v-pairI*: $[\![$ *i* $[\![$ *v1* $]\!]^{\sim}$ *s1* ; *i* $[\![$ *v2* $]\!]^{\sim}$ *s2* $]\!] \Longrightarrow$ *i* $[\![$ *V-pair v1 v2* $]\!]^{\sim}$ *SPair s1 s2*
| *eval-v-consI*: *i* $[\![$ *v* $]\!]^{\sim}$ *s* $\Longrightarrow$ *i* $[\![$ *V-cons tyid dc v* $]\!]^{\sim}$ *SCons tyid dc s*
| *eval-v-conspI*: *i* $[\![$ *v* $]\!]^{\sim}$ *s* $\Longrightarrow$ *i* $[\![$ *V-consp tyid dc b v* $]\!]^{\sim}$ *SConsp tyid dc b s*
equivariance *eval-v*
nominal-inductive *eval-v* .

inductive-cases *eval-v-elims*:
i $[\![$ *V-lit l* $]\!]^{\sim}$ *s*
i $[\![$ *V-var x* $]\!]^{\sim}$ *s*
i $[\![$ *V-pair v1 v2* $]\!]^{\sim}$ *s*
i $[\![$ *V-cons tyid dc v* $]\!]^{\sim}$ *s*
i $[\![$ *V-consp tyid dc b v* $]\!]^{\sim}$ *s*

inductive *eval-e* :: *valuation* $\Rightarrow$ *ce* $\Rightarrow$ *rcl-val* $\Rightarrow$ *bool* (- $[\![$ - $]\!]^{\sim}$ -) **where**
eval-e-valI: *i* $[\![$ *v* $]\!]^{\sim}$ *sv* $\Longrightarrow$ *i* $[\![$ *CE-val v* $]\!]^{\sim}$ *sv*

| *eval-e-plusI*: ⟦ *i* ⟦ *v1* ⟧ ~ *SNum n1*; *i* ⟦ *v2* ⟧ ~ *SNum n2* ⟧ ⟹ *i* ⟦ (*CE-op Plus v1 v2*) ⟧ ~ (*SNum* (*n1*+*n2*))
| *eval-e-leqI*: ⟦ *i* ⟦ *v1* ⟧ ~ (*SNum n1*); *i* ⟦ *v2* ⟧ ~ (*SNum n2*) ⟧ ⟹ *i* ⟦ (*CE-op LEq v1 v2*) ⟧ ~ (*SBool* (*n1* ≤ *n2*))
| *eval-e-fstI*: ⟦ *i* ⟦ *v* ⟧ ~ *SPair v1 v2* ⟧ ⟹ *i* ⟦ (*CE-fst v*) ⟧ ~ *v1*
| *eval-e-sndI*: ⟦ *i* ⟦ *v* ⟧ ~ *SPair v1 v2* ⟧ ⟹ *i* ⟦ (*CE-snd v*) ⟧ ~ *v2*
| *eval-e-concatI*:⟦ *i* ⟦ *v1* ⟧ ~ (*SBitvec bv1*); *i* ⟦ *v2* ⟧ ~ (*SBitvec bv2*) ⟧ ⟹ *i* ⟦ (*CE-concat v1 v2*) ⟧ ~ (*SBitvec* (*bv1*@*bv2*))
| *eval-e-lenI*:⟦ *i* ⟦ *v* ⟧ ~ (*SBitvec bv*) ⟧ ⟹ *i* ⟦ (*CE-len v*) ⟧ ~ (*SNum* (*int* (*List.length bv*)))
equivariance *eval-e*
nominal-inductive *eval-e* .

thm *eval-e.induct*

inductive-cases *eval-e-elims*:
i ⟦ (*CE-val v*) ⟧ ~ *s*
i ⟦ (*CE-op Plus v1 v2*) ⟧ ~ *s*
i ⟦ (*CE-op LEq v1 v2*) ⟧ ~ *s*
i ⟦ (*CE-fst v*) ⟧ ~ *s*
i ⟦ (*CE-snd v*) ⟧ ~ *s*
i ⟦ (*CE-concat v1 v2*) ⟧ ~ *s*
i ⟦ (*CE-len v*) ⟧ ~ *s*

inductive *eval-c* :: *valuation* ⇒ *c* ⇒ *bool* ⇒ *bool* (- ⟦ - ⟧ ~ -) **where**
eval-c-trueI: *i* ⟦ *C-true* ⟧ ~ *True*
| *eval-c-falseI*:*i* ⟦ *C-false* ⟧ ~ *False*
| *eval-c-conjI*: ⟦ *i* ⟦ *c1* ⟧ ~ *b1* ; *i* ⟦ *c2* ⟧ ~ *b2* ⟧ ⟹ *i* ⟦ (*C-conj c1 c2*) ⟧ ~ (*b1* ∧ *b2*)
| *eval-c-disjI*: ⟦ *i* ⟦ *c1* ⟧ ~ *b1* ; *i* ⟦ *c2* ⟧ ~ *b2* ⟧ ⟹ *i* ⟦ (*C-disj c1 c2*) ⟧ ~ (*b1* ∨ *b2*)
| *eval-c-impI*:⟦ *i* ⟦ *c1* ⟧ ~ *b1* ; *i* ⟦ *c2* ⟧ ~ *b2* ⟧ ⟹ *i* ⟦ (*C-imp c1 c2*) ⟧ ~ (*b1* ⟶ *b2*)
| *eval-c-notI*:⟦ *i* ⟦ *c* ⟧ ~ *b* ⟧ ⟹ *i* ⟦ (*C-not c*) ⟧ ~ (¬ *b*)
| *eval-c-eqI*:⟦ *i* ⟦ *e1* ⟧ ~ *sv1*; *i* ⟦ *e2* ⟧ ~ *sv2* ⟧ ⟹ *i* ⟦ (*C-eq e1 e2*) ⟧ ~ (*sv1*=*sv2*)
equivariance *eval-c*
nominal-inductive *eval-c* .

inductive-cases *eval-c-elims*:
i ⟦ *C-true* ⟧ ~ *True*
i ⟦ *C-false* ⟧ ~ *False*
i ⟦ (*C-conj c1 c2*)⟧ ~ *s*
i ⟦ (*C-disj c1 c2*)⟧ ~ *s*
i ⟦ (*C-imp c1 c2*)⟧ ~ *s*
i ⟦ (*C-not c*) ⟧ ~ *s*
i ⟦ (*C-eq e1 e2*)⟧ ~ *s*
i ⟦ *C-true* ⟧ ~ *s*
i ⟦ *C-false* ⟧ ~ *s*

7.1.5 Satisfiability

inductive *is-satis* :: *valuation* ⇒ *c* ⇒ *bool* (- ⊨ -) **where**
i ⟦ *c* ⟧ ~ *True* ⟹ *i* ⊨ *c*
equivariance *is-satis*
nominal-inductive *is-satis* .

nominal-function *is-satis-g* :: *valuation* ⇒ Γ ⇒ *bool* (- ⊨ -) **where**
i ⊨ *GNil* = *True*
| *i* ⊨ ((*x*,*b*,*c*) #Γ *G*) = (*i* ⊨ *c* ∧ *i* ⊨ *G*)
apply(*auto simp*: *eqvt-def is-satis-g-graph-aux-def*)
by (*metis* Γ.*exhaust old.prod.exhaust*)
nominal-termination (*eqvt*) **by** *lexicographic-order*

7.2 Validity

nominal-function *valid* :: Θ ⇒ *B* ⇒ Γ ⇒ *c* ⇒ *bool* (- ; - ; - ⊨ - [*50*, *50*] *50*) **where**
P ; *B* ; *G* ⊨ *c* = ((*P* ; *B* ; *G* ⊢*wf* *c*) ∧ (∀ *i*. (*P* ; *G* ⊢ *i*) ∧ *i* ⊨ *G* ⟶ *i* ⊨ *c*))
by (*auto simp*: *eqvt-def wfI-def valid-graph-aux-def*)
nominal-termination (*eqvt*) **by** *lexicographic-order*

7.3 Lemmas

Lemmas needed for Examples

lemma *valid-trueI* [*intro*]:
fixes *G*::Γ
assumes *P* ; *B* ⊢*wf* *G*
shows *P* ; *B* ; *G* ⊨ *C-true*
proof −
have ∀ *i*. *i* ⊨ *C-true* **using** *is-satis.simps eval-c-trueI* **by** *simp*
moreover have *P* ; *B* ; *G* ⊢*wf* *C-true* **using** *wfC-trueI assms* **by** *simp*
ultimately show *?thesis* **using** *valid.simps* **by** *simp*
qed

inductive *split* :: *int* ⇒ *bit list* ⇒ *bit list* ∗ *bit list* ⇒ *bool* **where**
split 0 xs ([], *xs*)
| *split m xs* (*ys*,*zs*) ⟹ *split* (*m*+*1*) (*x*#*xs*) ((*x* # *ys*), *zs*)
equivariance *split*
nominal-inductive *split* **.**

lemma *split-concat*:
assumes *split n v* (*v1*,*v2*)
shows *v* = *append v1 v2*
using *assms* **proof**(*induct* (*v1*,*v2*) *arbitrary*: *v1 v2 rule*: *split.inducts*)
case *1*
then show *?case* **by** *auto*
next
case (*2 m xs ys zs x*)
then show *?case* **by** *auto*
qed

lemma *split-n*:
assumes *split n v* (*v1*,*v2*)
shows *0* ≤ *n* ∧ *n* ≤ *int* (*length v*)
using *assms* **proof**(*induct rule*: *split.inducts*)
case (*1 xs*)

then show *?case* **by** *auto*
next
case (*2 m xs ys zs x*)
then show *?case* **by** *auto*
qed

lemma *split-length*:
assumes *split n v* (*v1*,*v2*)
shows *n* = *int* (*length v1*)
using *assms* **proof**(*induct* (*v1*,*v2*) *arbitrary*: *v1 v2 rule*: *split.inducts*)
case (*1 xs*)
then show *?case* **by** *auto*
next
case (*2 m xs ys zs x*)
then show *?case* **by** *auto*
qed

lemma *obtain-split*:
assumes *0* ≤ *n* **and** *n* ≤ *int* (*length bv*)
shows ∃ *bv1 bv2*. *split n bv* (*bv1* , *bv2*)
using *assms* **proof**(*induct bv arbitrary*: *n*)
case *Nil*
then show *?case* **using** *split.intros* **by** *auto*
next
case (*Cons b bv*)
show *?case* **proof**(*cases n* = *0*)
case *True*
then show *?thesis* **using** *split.intros* **by** *auto*
next
case *False*
then obtain *m* **where** *m*:*n*=*m*+*1* **using** *Cons*
by (*metis add.commute add-minus-cancel*)
moreover have *0* ≤ *m* **using** *False m Cons* **by** *linarith*
then obtain *bv1* **and** *bv2* **where** *split m bv* (*bv1* , *bv2*) **using** *Cons m* **by** *force*
hence *split n* (*b* # *bv*) ((*b*#*bv1*), *bv2*) **using** *m split.intros* **by** *auto*
then show *?thesis* **by** *auto*
qed
qed

end

7.4 Syntax Lemmas

lemma *supp-v-tau* [*simp*]:
assumes *atom z* ♯ *v*
shows *supp* ({| *z* : *b* | *CE-val* (*V-var z*) == *CE-val v* |}) = *supp v* ∪ *supp b*
using *assms τ.supp c.supp ce.supp*
by (*simp add*: *fresh-def supp-at-base*)

lemma *supp-v-var-tau* [*simp*]:

assumes $z \neq x$
shows *supp* ({| $z : b \mid$ *CE-val* (*V-var z*) == *CE-val* (*V-var x*) |}) = { *atom x* } ∪ *supp b*
using *supp-v-tau assms*
using *supp-at-base* **by** *fastforce*

Sometimes we need to work with a version of a binder where the variable is fresh in something else, such as a bigger context. I think these could be generated automatically

lemma *obtain-fresh-fun-def*:
fixes *t*::'*b*::*fs*
shows ∃ *y*::*x*. *atom y* ♯ (*s*,*c*,*τ*,*t*) ∧ (*AF-fundef f* (*AF-fun-typ-none* (*AF-fun-typ x b c τ s*)) = *AF-fundef f* (*AF-fun-typ-none* (*AF-fun-typ y b* ((*y* ↔ *x*) • *c*) ((*y* ↔ *x*) • *τ*) ((*y* ↔ *x*) • *s*))))
proof −
obtain *y*::*x* **where** *y*: *atom y* ♯ (*s*,*c*,*τ*,*t*) **using** *obtain-fresh* **by** *blast*
moreover have *AF-fundef f* (*AF-fun-typ-none* (*AF-fun-typ y b* ((*y* ↔ *x*) • *c*) ((*y* ↔ *x*) • *τ*) ((*y* ↔ *x*) • *s*))) = (*AF-fundef f* (*AF-fun-typ-none* (*AF-fun-typ x b c τ s*)))
proof(*cases x*=*y*)
case *True*
then show *?thesis* **using** *fun-def.eq-iff Abs1-eq-iff*(*3*) *flip-commute flip-fresh-fresh fresh-PairD* **by** *auto*
next
case *False*
thm *fun-typ.eq-iff*
have (*AF-fun-typ y b* ((*y* ↔ *x*) • *c*) ((*y* ↔ *x*) • *τ*) ((*y* ↔ *x*) • *s*)) =(*AF-fun-typ x b c τ s*) **proof**(*subst fun-typ.eq-iff*, *subst Abs1-eq-iff*(*3*))
show ‹(*y* = *x* ∧ (((*y* ↔ *x*) • *c*, (*y* ↔ *x*) • *τ*), (*y* ↔ *x*) • *s*) = ((*c*, *τ*), *s*) ∨
y ≠ *x* ∧ (((*y* ↔ *x*) • *c*, (*y* ↔ *x*) • *τ*), (*y* ↔ *x*) • *s*) = (*y* ↔ *x*) • ((*c*, *τ*), *s*) ∧ *atom y* ♯ ((*c*, *τ*), *s*)) ∧
b = *b*› **using** *False flip-commute flip-fresh-fresh fresh-PairD y* **by** *auto*
qed
thus *?thesis* **by** *metis*
qed
ultimately show *?thesis* **using** *y fresh-Pair* **by** *metis*
qed

lemma *lookup-fun-member*:
assumes *Some* (*AF-fundef f ft*) = *lookup-fun* Φ *f*
shows *AF-fundef f ft* ∈ *set* Φ
using *assms* **proof** (*induct* Φ)
case *Nil*
then show *?case* **by** *auto*
next
case (*Cons a* Φ)
then show *?case* **using** *lookup-fun.simps*
by (*metis fun-def.exhaust insert-iff list.simps*(*15*) *option.inject*)
qed

lemma *rig-dom-eq*:
dom (*G*[*x* ⟼ *c*]) = *dom G*
proof(*induct G rule*: Γ.*induct*)
case *GNil*

then show *?case* **using** *replace-in-g.simps* **by** *presburger*
next
case (*GCons xbc* Γ′)
obtain *x′* **and** *b′* **and** *c′* **where** *xbc*: *xbc*=(*x′*,*b′*,*c′*) **using** *prod-cases3* **by** *blast*
then show *?case* **using** *replace-in-g.simps GCons* **by** *simp*
qed

lemma *lookup-in-rig-eq*:
assumes *Some* (*b*,*c*) = *lookup* Γ *x*
shows *Some* (*b*,*c′*) = *lookup* (Γ[*x*↦⟶*c′*]) *x*
using *assms* **proof**(*induct* Γ *rule*: Γ-*induct*)
case *GNil*
then show *?case* **by** *auto*
next
case (*GCons x b c* Γ′)
then show *?case* **using** *replace-in-g.simps lookup.simps* **by** *auto*
qed

lemma *lookup-in-rig-neq*:
assumes *Some* (*b*,*c*) = *lookup* Γ *y* **and** *x*≠*y*
shows *Some* (*b*,*c*) = *lookup* (Γ[*x*↦⟶*c′*]) *y*
using *assms* **proof**(*induct* Γ *rule*: Γ-*induct*)
case *GNil*
then show *?case* **by** *auto*
next
case (*GCons x′ b′ c′* Γ′)
then show *?case* **using** *replace-in-g.simps lookup.simps* **by** *auto*
qed

lemma *lookup-in-rig*:
assumes *Some* (*b*,*c*) = *lookup* Γ *y*
shows ∃ *c″*. *Some* (*b*,*c″*) = *lookup* (Γ[*x*↦⟶*c′*]) *y*
proof(*cases x*=*y*)
case *True*
then show *?thesis* **using** *lookup-in-rig-eq* **using** *assms* **by** *blast*
next
case *False*
then show *?thesis* **using** *lookup-in-rig-neq* **using** *assms* **by** *blast*
qed

lemma *lookup-inside*[*simp*]:
assumes *x* ∉ *fst* ‘ *setG* Γ′
shows *Some* (*b1*,*c1*) = *lookup* (Γ′@(*x*,*b1*,*c1*)#Γ Γ) *x*
using *assms* **by**(*induct* Γ′,*auto*)

lemma *lookup-inside2*:
assumes *Some* (*b1*,*c1*) = *lookup* (Γ′@((*x*,*b0*,*c0*)#Γ Γ)) *y* **and** *x*≠*y*
shows *Some* (*b1*,*c1*) = *lookup* (Γ′@((*x*,*b0*,*c0′*)#Γ Γ)) *y*
using *assms* **by**(*induct* Γ′ *rule*: Γ.*induct*,*auto*+)

fun *tail*:: *′a list* ⇒ *′a list* **where**
tail [] = []

| *tail* (*x*#*xs*) = *xs*

lemma *lookup-options*:
  assumes *Some* (*b*,*c*) = *lookup* (*xt*#$_\Gamma$ *G*) *x*
  shows ((*x*,*b*,*c*) = *xt*) ∨ (*Some* (*b*,*c*) = *lookup G x*)
by (*metis assms lookup.simps*(*2*) *option.inject surj-pair*)

lemma *lookup-x*:
  assumes *Some* (*b*,*c*) = *lookup G x*
  shows *x* ∈ *fst* ‘ *setG G*
  using *assms*
  by(*induct G rule*: Γ.*induct* ,*auto*+)

lemma *GCons-eq-appendI*:
  fixes *xs1*::Γ
  shows [| *x* #$_\Gamma$ *xs1* = *ys*; *xs* = *xs1* @ *zs* |] ==> *x* #$_\Gamma$ *xs* = *ys* @ *zs*
by (*drule sym*) *simp*

lemma *split-G*: *x* : *setG xs* ⟹ ∃ *ys zs*. *xs* = *ys* @ *x* #$_\Gamma$ *zs*
proof (*induct xs*)
  case *GNil* **thus** *?case* **by** *simp*
next
  case *GCons* **thus** *?case* **using** *GCons-eq-appendI*
    by (*metis Un-iff append-g.simps*(*1*) *singletonD setG.simps*(*2*))
qed

lemma *lookup-not-empty*:
  assumes *Some* τ = *lookup G x*
  shows *G* ≠ *GNil*
  using *assms* **by** *auto*

lemma *lookup-in-g*:
 assumes *Some* (*b*,*c*) = *lookup* Γ *x*
 shows (*x*,*b*,*c*) ∈ *setG* Γ
using *assms* **apply**(*induct* Γ, *simp*)
using *lookup-options* **by** *fastforce*

lemma *lookup-split*:
  fixes Γ::Γ
  assumes *Some* (*b*,*c*) = *lookup* Γ *x*
  shows ∃ *G G'* . Γ = *G'*@(*x*,*b*,*c*)#$_\Gamma$ *G*
  by (*meson assms*(*1*) *lookup-in-g split-G*)

lemma *setG-splitU*[*simp*]:
  (*x'*,*b'*,*c'*) ∈ *setG* (Γ' @ (*x*, *b*, *c*) #$_\Gamma$ Γ) ⟷ (*x'*,*b'*,*c'*) ∈ (*setG* Γ' ∪ {(*x*, *b*, *c*)} ∪ *setG* Γ)
  using *append-g-setGU setG.simps* **by** *auto*

lemma *setG-splitP*[*simp*]:
 (∀ (*x'*, *b'*, *c'*)∈*setG* (Γ' @ (*x*, *b*, *c*) #$_\Gamma$ Γ). *P x' b' c'*) ⟷ (∀ (*x'*, *b'*, *c'*)∈*setG* Γ'. *P x' b' c'*) ∧ *P x b c* ∧ (∀ (*x'*, *b'*, *c'*)∈*setG* Γ. *P x' b' c'*) (**is** *?A* ⟷ *?B*)
  using *setG-splitU* **by** *force*

lemma *lookup-restrict*:
assumes *Some* $(b',c') = lookup\ (\Gamma'@(x,b,c)\#_\Gamma\Gamma)\ y$ **and** $x \neq y$
shows *Some* $(b',c') = lookup\ (\Gamma'@\Gamma)\ y$
using *assms* **proof**(*induct* Γ' *rule*:Γ-*induct*)
case *GNil*
then show *?case* **by** *auto*
next
case (*GCons x1 b1 c1* Γ')
then show *?case* **by** *auto*
qed

lemma *supp-list-member*:
fixes *x*::'*a*::*fs* **and** *l*::'*a list*
assumes $x \in set\ l$
shows $supp\ x \subseteq supp\ l$
using *assms* **apply**(*induct l*, *auto*)
using *supp-Cons* **by** *auto*

lemma *GNil-append*:
assumes *GNil* = *G1*@*G2*
shows *G1* = *GNil* ∧ *G2* = *GNil*
proof(*rule ccontr*)
assume ¬ (*G1* = *GNil* ∧ *G2* = *GNil*)
hence *G1*@*G2* ≠ *GNil* **using** *append-g.simps* **by** (*metis* Γ.*distinct*(*1*) Γ.*exhaust*)
thus *False* **using** *assms* **by** *auto*
qed

lemma *GCons-eq-append-conv*:
fixes *xs*::Γ
shows $x\#_\Gamma xs = ys@zs = (ys = GNil \wedge x\#_\Gamma xs = zs \vee (\exists\, ys'.\ x\#_\Gamma ys' = ys \wedge xs = ys'@zs))$
by(*cases ys*) *auto*

7.5 Type Definitions

lemma *exist-fresh-bv*:
fixes *tm*::'*a*::*fs*
shows ∃ *bva2 dclist2*. *AF-typedef-poly tyid bva dclist* = *AF-typedef-poly tyid bva2 dclist2* ∧
atom bva2 ♯ *tm*
proof −
obtain *bva2*::*bv* **where** ∗:*atom bva2* ♯ (*bva*, *dclist*,*tyid*,*tm*) **using** *obtain-fresh* **by** *metis*
moreover hence *bva2* ≠ *bva* **using** *fresh-at-base* **by** *auto*
moreover have *dclist* = (*bva* ↔ *bva2*) ∙ (*bva2* ↔ *bva*) ∙ *dclist* **by** *simp*
moreover have *atom bva* ♯ (*bva2* ↔ *bva*) ∙ *dclist* **proof** −
have *atom bva2* ♯ *dclist* **using** ∗ *fresh-prodN* **by** *auto*
hence *atom* ((*bva2* ↔ *bva*) ∙ *bva2*) ♯ (*bva2* ↔ *bva*) ∙ *dclist* **using** *fresh-eqvt True-eqvt*
proof −
have (*bva2* ↔ *bva*) ∙ *atom bva2* ♯ (*bva2* ↔ *bva*) ∙ *dclist*
by (*metis True-eqvt* ‹*atom bva2* ♯ *dclist*› *fresh-eqvt*)
then show *?thesis*
by *simp*
qed

thus *?thesis* **by** *auto*
qed
ultimately have *AF-typedef-poly tyid bva dclist = AF-typedef-poly tyid bva2 ((bva2 ↔ bva) · dclist)*

unfolding *type-def.eq-iff Abs1-eq-iff* **by** *metis*
thus *?thesis* **using** ∗ *fresh-prodN* **by** *metis*
qed

lemma *obtain-fresh-bv*:
fixes *tm*::*'a*::*fs*
obtains *bva2*::*bv* **and** *dclist2* **where** *AF-typedef-poly tyid bva dclist = AF-typedef-poly tyid bva2 dclist2* ∧
atom bva2 ♯ *tm*
using *exist-fresh-bv* **by** *metis*

7.6 Function Definitions

lemma *fun-typ-flip*:
fixes *bv1*::*bv* **and** *c*::*bv*
shows *(bv1 ↔ c) · AF-fun-typ x1 b1 c1 τ1 s1 = AF-fun-typ x1 ((bv1 ↔ c) · b1) ((bv1 ↔ c) · c1) ((bv1 ↔ c) · τ1) ((bv1 ↔ c) · s1)*
using *fun-typ.perm-simps flip-fresh-fresh supp-at-base fresh-def flip-fresh-fresh fresh-def supp-at-base*
by (*simp add*: *flip-fresh-fresh*)

lemma *fun-def-eq*:
assumes *AF-fundef fa (AF-fun-typ-none (AF-fun-typ xa ba ca τa sa)) = AF-fundef f (AF-fun-typ-none (AF-fun-typ x b c τ s))*
shows *f = fa* **and** *b = ba* **and** *[[atom xa]]lst. sa = [[atom x]]lst. s* **and** *[[atom xa]]lst. τa = [[atom x]]lst. τ* **and**
[[atom xa]]lst. ca = [[atom x]]lst. c
using *fun-def.eq-iff fun-typ-q.eq-iff fun-typ.eq-iff lst-snd lst-fst* **using** *assms* **apply** *metis*
using *fun-def.eq-iff fun-typ-q.eq-iff fun-typ.eq-iff lst-snd lst-fst* **using** *assms* **apply** *metis*
proof −
have *([[atom xa]]lst. ((ca, τa), sa) = [[atom x]]lst. ((c, τ), s))* **using** *assms fun-def.eq-iff fun-typ-q.eq-iff fun-typ.eq-iff* **by** *auto*
thus *[[atom xa]]lst. sa = [[atom x]]lst. s* **and** *[[atom xa]]lst. τa = [[atom x]]lst. τ* **and**
[[atom xa]]lst. ca = [[atom x]]lst. c **using** *lst-snd lst-fst* **by** *metis+*
qed

lemma *fun-arg-unique-aux*:
assumes *AF-fun-typ x1 b1 c1 τ1′ s1′ = AF-fun-typ x2 b2 c2 τ2′ s2′*
shows *{| x1 : b1 | c1 |} = {| x2 : b2 | c2|}*
proof −
have *([[atom x1]]lst. c1 = [[atom x2]]lst. c2)* **using** *fun-def-eq assms* **by** *metis*
moreover have *b1 = b2* **using** *fun-typ.eq-iff assms* **by** *metis*
ultimately show *?thesis* **using** *τ.eq-iff* **by** *fast*
qed

lemma *fresh-x-neq*:

fixes $x::x$ **and** $y::x$
shows *atom* $x \sharp y = (x \neq y)$
using *fresh-at-base fresh-def* **by** *auto*

lemma *obtain-fresh-z3*:
fixes $tm::'b::fs$
obtains $z::x$ **where** $\{\!|\ x : b\ |\ c\ |\!\} = \{\!|\ z : b\ |\ c[x::=V\text{-}var\ z]_{cv}\ |\!\} \wedge$ *atom* $z \sharp tm \wedge$ *atom* $z \sharp (x,c)$
proof −
obtain $z::x$ **and** $c'::c$ **where** $z:\{\!|\ x : b\ |\ c\ |\!\} = \{\!|\ z : b \mid c'\ |\!\} \wedge$ *atom* $z \sharp (tm,x,c)$ **using** *obtain-fresh-z2 b-of.simps* **by** *metis*
hence $c' = c[x::=V\text{-}var\ z]_{cv}$ **proof** −
have $([[atom\ z]]lst.\ c' = [[atom\ x]]lst.\ c)$ **using** $z\ \tau.eq\text{-}iff$ **by** *metis*
hence $c' = (z \leftrightarrow x) \cdot c$ **using** *Abs1-eq-iff*[*of z c' x c*] *fresh-x-neq fresh-prodN* **by** *fastforce*
also have ... $= c[x::=V\text{-}var\ z]_{cv}$
using *subst-v-c-def flip-subst-v*[*of z c x*] *z fresh-prod3* **by** *metis*
finally show *?thesis* **by** *auto*
qed
thus *?thesis* **using** *z fresh-prodN that* **by** *metis*
qed

lemma *u-fresh-v*:
fixes $u::u$ **and** $t::v$
shows *atom* $u \sharp t$
by(*nominal-induct t rule*:*v.strong-induct*,*auto*)

lemma *u-fresh-ce*:
fixes $u::u$ **and** $t::ce$
shows *atom* $u \sharp t$
apply(*nominal-induct t rule*:*ce.strong-induct*)
using *u-fresh-v pure-fresh*
apply (*auto simp add*: *opp.fresh ce.fresh opp.fresh opp.exhaust*)
unfolding *ce.fresh opp.fresh opp.exhaust* **by** (*simp add*: *fresh-opp-all*)

lemma *u-fresh-c*:
fixes $u::u$ **and** $t::c$
shows *atom* $u \sharp t$
by(*nominal-induct t rule*:*c.strong-induct*,*auto simp add*: *c.fresh u-fresh-ce*)

lemma *u-fresh-g*:
fixes $u::u$ **and** $t::\Gamma$
shows *atom* $u \sharp t$
by(*induct t rule*:Γ-*induct*, *auto simp add*: *u-fresh-b u-fresh-c fresh-GCons fresh-GNil*)

lemma *u-fresh-t*:
fixes $u::u$ **and** $t::\tau$
shows *atom* $u \sharp t$
by(*nominal-induct t rule*:τ.*strong-induct*,*auto simp add*: τ.*fresh u-fresh-c u-fresh-b*)

lemma *b-of-c-of-eq*:

assumes *atom z* $\sharp$ τ
shows $\{\!|\ z : b\text{-}of\ \tau\ |\ \ c\text{-}of\ \tau\ z\ |\!\} = \tau$
using *assms* **proof**(*nominal-induct* τ *avoiding*: *z rule*: τ.*strong-induct*)
case (*T-refined-type x1a x2a x3a*)
hence $\{\!|\ z : b\text{-}of\ \{\!|\ x1a : x2a\ |\ x3a\ |\!\}\ |\ c\text{-}of\ \{\!|\ x1a : x2a\ |\ x3a\ |\!\}\ z\ |\!\} = \{\!|\ z : x2a\ |\ x3a[x1a::=V\text{-}var\ z]_{cv}\ |\!\}$
using *b-of.simps c-of.simps c-of-eq* **by** *auto*
thus *?case* **using** *T-refined-type* **by** *auto*
qed

lemma *fresh-d-not-in*:
assumes *atom u2* $\sharp$ Δ'
shows *u2* $\notin$ *fst* ‘ *setD* Δ'
using *assms* **proof**(*induct* Δ' *rule*: Δ-*induct*)
case *DNil*
then show *?case* **by** *simp*
next
case (*DCons u t* Δ')
hence $*$: *atom u2* $\sharp$ $\Delta' \wedge$ *atom u2* $\sharp$ (*u*,*t*)
by (*simp add*: *fresh-def supp-DCons*)
hence *u2* $\notin$ *fst* ‘ *setD* Δ' **using** *DCons* **by** *auto*
moreover have *u2* $\neq$ *u* **using** $*$ *fresh-Pair*
by (*metis eq-fst-iff not-self-fresh*)
ultimately show *?case* **by** *simp*
qed

end

Chapter 8

Wellformedness Lemmas

8.1 Prelude

lemma *b-of-subst-bb-commute*:
(*b-of* (τ[*bv*::=*b*]$_{\tau b}$)) = (*b-of* τ)[*bv*::=*b*]$_{bb}$
proof −
obtain *z'* **and** *b'* **and** *c'* **where** τ = {‖ *z'* : *b'* | *c'* ‖} **using** *obtain-fresh-z* **by** *metis*
moreover hence (*b-of* (τ[*bv*::=*b*]$_{\tau b}$)) = *b-of* {‖ *z'* : *b'*[*bv*::=*b*]$_{bb}$ | *c'* ‖} **using** *subst-tb.simps* **by** *simp*
ultimately show *?thesis* **using** *subst-tv.simps subst-tb.simps* **by** *simp*
qed

lemmas *wf-intros* = *wfV-wfC-wfG-wfT-wfTs-wfTh-wfB-wfCE-wfTD.intros wfE-wfS-wfCS-wfCSS-wfPhi-wfD-wfFTQ-wfF*

lemmas *freshers* = *fresh-prodN b.fresh c.fresh v.fresh ce.fresh fresh-GCons fresh-GNil fresh-at-base*

8.2 Strong Elimination

lemma *wf-strong-elim*:
fixes Γ::Γ **and** Γ'::Γ **and** *v*::*v* **and** *e*::*e* **and** *c*::*c* **and** τ::τ **and** *ts*::(*string*∗τ) *list*
and Δ::Δ **and** *b*::*b* **and** *ftq*::*fun-typ-q* **and** *ft*::*fun-typ* **and** *ce*::*ce* **and** *td*::*type-def* **and** *s*::*s*
and *tm*::'*a*::*fs*
and *cs*::*branch-s* **and** *css*::*branch-list* **and** Θ::Θ
shows Θ ; B ; Γ ⊢$_{wf}$ (*V-consp tyid dc b v*) : *b''* ⟹ (∃ *bv dclist x b' c*. *b''* = *B-app tyid b* ∧
AF-typedef-poly tyid bv dclist ∈ *set* Θ ∧
(*dc*, {‖ *x* : *b'* | *c* ‖}) ∈ *set dclist* ∧
Θ ; B ⊢$_{wf}$ *b* ∧ *atom bv* ♯ (Θ, B, Γ, *b*, *v*) ∧ Θ ; B ; Γ ⊢$_{wf}$ *v* : *b'*[*bv*::=*b*]$_{bb}$ ∧ *atom bv* ♯ *tm*) **and**
Θ ; B ; Γ ⊢$_{wf}$ *c* ⟹ *True* **and**
Θ ; B ⊢$_{wf}$ Γ ⟹ *True* **and**
Θ ; B ; Γ ⊢$_{wf}$ τ ⟹
∃ *z b c*. τ = {‖ *z* : *b* | *c* ‖} ∧ *atom z* ♯ (Θ, B, Γ) ∧ *atom z* ♯ *tm* ∧
Θ ; B ⊢$_{wf}$ *b* ∧ Θ ; B ; (*z*, *b*, *TRUE*) #$_{\Gamma}$ Γ ⊢$_{wf}$ *c* **and**
Θ ; B ; Γ ⊢$_{wf}$ *ts* ⟹ *True* **and**
⊢$_{wf}$ Θ ⟹ *True* **and**
Θ ; B ⊢$_{wf}$ *b* ⟹ *True* **and**
Θ ; B ; Γ ⊢$_{wf}$ *ce* : *b'* ⟹ *True* **and**
Θ ⊢$_{wf}$ *td* ⟹ *True*
proof(*nominal-induct*

V-consp tyid dc b v b'' **and** *c* **and** Γ **and** τ **and** *ts* **and** Θ **and** *b* **and** *b'* **and** *td*
avoiding: *tm*

rule:*wfV-wfC-wfG-wfT-wfTs-wfTh-wfB-wfCE-wfTD.strong-induct*)
case (*wfV-conspI bv dclist* Θ *x b' c* $\mathcal{B}$ Γ)
then show *?case* **by** *force*
next
case (*wfTI z* Θ $\mathcal{B}$ Γ *b c*)
then show *?case* **by** *force*
qed(*auto+*)

8.3 Context Extension

definition *wfExt* :: $\Theta \Rightarrow \mathcal{B} \Rightarrow \Gamma \Rightarrow \Gamma \Rightarrow$ *bool* (- ; - $\vdash_{wf}$ - < - [*50*,*50*,*50*] *50*) **where**
wfExt T B G1 G2 = (*wfG T B G2* $\wedge$ *wfG T B G1* $\wedge$ *setG G1* $\subseteq$ *setG G2*)

8.4 Context

lemma *wfG-cons*[*ms-wb*]:
fixes Γ::Γ
assumes *P* ; $\mathcal{B}$ $\vdash_{wf}$ (*z*,*b*,*c*) $\#_\Gamma\Gamma$
shows *P* ; $\mathcal{B}$ $\vdash_{wf}$ Γ $\wedge$ *atom z* $\sharp$ Γ $\wedge$ *wfB P* $\mathcal{B}$ *b*
using *wfG-elims*(*2*)[*OF assms*] **by** *metis*

lemma *wfG-cons2*[*ms-wb*]:
fixes Γ::Γ
assumes *P* ; $\mathcal{B}$ $\vdash_{wf}$ *zbc* $\#_\Gamma\Gamma$
shows *P* ; $\mathcal{B}$ $\vdash_{wf}$ Γ
proof −
obtain *z* **and** *b* **and** *c* **where** *zbc*: *zbc*=(*z*,*b*,*c*) **using** *prod-cases3* **by** *blast*
hence *P* ; $\mathcal{B}$ $\vdash_{wf}$ (*z*,*b*,*c*) $\#_\Gamma\Gamma$ **using** *assms* **by** *auto*
thus *?thesis* **using** *zbc wfG-cons assms* **by** *simp*
qed

lemma *wf-g-unique*:
fixes Γ::Γ
assumes Θ ; $\mathcal{B}$ $\vdash_{wf}$ Γ **and** (*x*,*b*,*c*) $\in$ *setG* Γ **and** (*x*,*b'*,*c'*) $\in$ *setG* Γ
shows *b*=*b'* $\wedge$ *c*=*c'*
using *assms* **proof**(*induct* Γ *rule*: Γ.*induct*)
case *GNil*
then show *?case* **by** *simp*
next
case (*GCons a* Γ)
consider (*x*,*b*,*c*)=*a* $\wedge$ (*x*,*b'*,*c'*)=*a* | (*x*,*b*,*c*)=*a* $\wedge$ (*x*,*b'*,*c'*)≠*a* | (*x*,*b*,*c*)≠*a* $\wedge$ (*x*,*b'*,*c'*)=*a* | (*x*,*b*,*c*)≠ *a* $\wedge$ (*x*,*b'*,*c'*)≠*a* **by** *blast*
then show *?case* **proof**(*cases*)
case *1*
then show *?thesis* **by** *auto*
next
case *2*
hence *atom x* $\sharp$ Γ **using** *wfG-elims*(*2*) *GCons* **by** *blast*

moreover have $(x,b',c') \in setG\ \Gamma$ **using** *GCons 2* **by** *force*
ultimately show *?thesis* **using** *forget-subst-gv fresh-GCons fresh-GNil fresh-gamma-elem* Γ*.distinct subst-gv.simps 2 GCons* **by** *metis*
next
case *3*
hence *atom* $x \sharp \Gamma$ **using** *wfG-elims(2) GCons* **by** *blast*
moreover have $(x,b,c) \in setG\ \Gamma$ **using** *GCons 3* **by** *force*
ultimately show *?thesis*
using *forget-subst-gv fresh-GCons fresh-GNil fresh-gamma-elem* Γ*.distinct subst-gv.simps 3 GCons* **by** *metis*
next
case *4*
then obtain x'' **and** b'' **and** $c''::c$ **where** *xbc*: $a=(x'',b'',c'')$
using *prod-cases3* **by** *blast*
hence $\Theta\ ;\ \mathcal{B} \vdash_{wf} ((x'',b'',c'')\ \#_\Gamma\Gamma)$ **using** *GCons wfG-elims* **by** *blast*
hence $\Theta\ ;\ \mathcal{B} \vdash_{wf} \Gamma \wedge (x,\ b,\ c) \in setG\ \Gamma \wedge (x,\ b',\ c') \in setG\ \Gamma$ **using** *GCons wfG-elims 4 xbc prod-cases3 set-GConsD* **using** *forget-subst-gv fresh-GCons fresh-GNil fresh-gamma-elem* Γ*.distinct subst-gv.simps 4 GCons* **by** *meson*
thus *?thesis* **using** *GCons* **by** *auto*
qed
qed

lemma *lookup-if1*:
fixes $\Gamma::\Gamma$
assumes $\Theta\ ;\ \mathcal{B} \vdash_{wf} \Gamma$ **and** *Some* $(b,c) = lookup\ \Gamma\ x$
shows $(x,b,c) \in setG\ \Gamma \wedge (\forall\, b'\ c'.\ (x,b',c') \in setG\ \Gamma \longrightarrow b'=b \wedge c'=c)$
using *assms* **proof**(*induct* Γ *rule*: Γ*.induct*)
case *GNil*
then show *?case* **by** *auto*
next
case (*GCons xbc* Γ)
then obtain x' **and** b' **and** $c'::c$ **where** *xbc*: $xbc=(x',b',c')$
using *prod-cases3* **by** *blast*
then show *?case* **using** *wf-g-unique GCons lookup-in-g xbc lookup.simps set-GConsD wfG.cases insertE insert-is-Un setG.simps wfG-elims* **by** *metis*
qed

lemma *lookup-if2*:
assumes *wfG P* $\mathcal{B}$ Γ **and** $(x,b,c) \in setG\ \Gamma \wedge (\forall\, b'\ c'.\ (x,b',c') \in setG\ \Gamma \longrightarrow b'=b \wedge c'=c)$
shows *Some* $(b,c) = lookup\ \Gamma\ x$
using *assms* **proof**(*induct* Γ *rule*: Γ*.induct*)
case *GNil*
then show *?case* **by** *auto*
next
case (*GCons xbc* Γ)
then obtain x' **and** b' **and** $c'::c$ **where** *xbc*: $xbc=(x',b',c')$
using *prod-cases3* **by** *blast*
then show *?case* **proof**(*cases* $x=x'$)
case *True*
then show *?thesis* **using** *lookup.simps GCons xbc* **by** *simp*
next

case *False*
then show *?thesis* **using** *lookup.simps GCons xbc setG.simps Un-iff set-GConsD wfG-cons2*
by (*metis* (*full-types*) *Un-iff set-GConsD setG.simps*(*2*) *wfG-cons2*)
qed
qed

lemma *lookup-iff*:
fixes Θ::Θ **and** Γ::Γ
assumes Θ ; $\mathcal{B} \vdash_{wf}$ Γ
shows *Some* (*b*,*c*) = *lookup* Γ *x* ⟷ (*x*,*b*,*c*) ∈ *setG* Γ ∧ (∀ *b*′ *c*′. (*x*,*b*′,*c*′) ∈ *setG* Γ ⟶ *b*′=*b* ∧ *c*′=*c*)
using *assms lookup-if1 lookup-if2* **by** *meson*

lemma *wfG-lookup-wf*:
fixes Θ::Θ **and** Γ::Γ **and** *b*::*b* **and** $\mathcal{B}$::$\mathcal{B}$
assumes Θ ; $\mathcal{B} \vdash_{wf}$ Γ **and** *Some* (*b*,*c*) = *lookup* Γ *x*
shows Θ ; $\mathcal{B} \vdash_{wf}$ *b*
using *assms* **proof**(*induct* Γ *rule*: Γ-*induct*)
case *GNil*
then show *?case* **by** *auto*
next
case (*GCons x*′ *b*′ *c*′ Γ′)
then show *?case* **proof**(*cases x*=*x*′)
case *True*
then show *?thesis* **using** *lookup.simps wfG-elims*(*2*) *GCons* **by** *fastforce*
next
case *False*
then show *?thesis* **using** *lookup.simps wfG-elims*(*2*) *GCons* **by** *fastforce*
qed
qed

lemma *wfG-unique*:
fixes Γ::Γ
assumes *wfG B* Θ ((*x*, *b*, *c*) #Γ Γ) **and** (*x1*,*b1*,*c1*) ∈ *setG* ((*x*, *b*, *c*) #Γ Γ) **and** *x1*=*x*
shows *b1* = *b* ∧ *c1* = *c*
proof −
have (*x*, *b*, *c*) ∈ *setG* ((*x*, *b*, *c*) #Γ Γ) **by** *simp*
thus *?thesis* **using** *wf-g-unique assms* **by** *blast*
qed

lemma *wfG-unique-full*:
fixes Γ::Γ
assumes *wfG* Θ *B* (Γ′@(*x*, *b*, *c*) #Γ Γ) **and** (*x1*,*b1*,*c1*) ∈ *setG* (Γ′@(*x*, *b*, *c*) #Γ Γ) **and** *x1*=*x*
shows *b1* = *b* ∧ *c1* = *c*
proof −
have (*x*, *b*, *c*) ∈ *setG* (Γ′@(*x*, *b*, *c*) #Γ Γ) **by** *simp*
thus *?thesis* **using** *wf-g-unique assms* **by** *blast*
qed

8.5 Converting between wb forms

We cannot prove wfB properties here for expressions and statements as need some more facts about Φ context which we can prove without this lemma. Trying to cram everything into a single large mutually recursive lemma is not a good idea

lemma *wfX-wfY1*:
fixes Γ::Γ **and** Γ'::Γ **and** *v*::*v* **and** *e*::*e* **and** *c*::*c* **and** τ::τ **and** *ts*::(*string*$*\tau$) *list* **and** Δ::Δ **and** *s*::*s* **and** *b*::*b* **and** *ftq*::*fun-typ-q* **and** *ft*::*fun-typ* **and** *ce*::*ce* **and** *td*::*type-def* **and** *cs*::*branch-s* **and** *css*::*branch-list*
shows *wfV-wf*: Θ ; $\mathcal{B}$; Γ $\vdash_{wf}$ *v* : *b* $\Longrightarrow$ Θ ; $\mathcal{B}$ $\vdash_{wf}$ Γ $\wedge$ $\vdash_{wf}$ Θ **and**
wfC-wf: Θ ; $\mathcal{B}$; Γ $\vdash_{wf}$ *c* $\Longrightarrow$ Θ ; $\mathcal{B}$ $\vdash_{wf}$ Γ $\wedge$ $\vdash_{wf}$ Θ **and**
wfG-wf :Θ ; $\mathcal{B}$ $\vdash_{wf}$ Γ $\Longrightarrow$ $\vdash_{wf}$ Θ **and**
wfT-wf: Θ ; $\mathcal{B}$; Γ $\vdash_{wf}$ τ $\Longrightarrow$ Θ ; $\mathcal{B}$ $\vdash_{wf}$ Γ $\wedge$ $\vdash_{wf}$ Θ $\wedge$ Θ ; $\mathcal{B}$ $\vdash_{wf}$ *b-of* τ **and**
wfTs-wf:Θ ; $\mathcal{B}$; Γ $\vdash_{wf}$ *ts* $\Longrightarrow$ Θ ; $\mathcal{B}$ $\vdash_{wf}$ Γ $\wedge$ $\vdash_{wf}$ Θ **and**
$\vdash_{wf}$ Θ $\Longrightarrow$ *True* **and**
wfB-wf: Θ ; $\mathcal{B}$ $\vdash_{wf}$ *b* $\Longrightarrow$ $\vdash_{wf}$ Θ **and**
wfCE-wf: Θ ; $\mathcal{B}$; Γ $\vdash_{wf}$ *ce* : *b* $\Longrightarrow$ Θ ; $\mathcal{B}$ $\vdash_{wf}$ Γ $\wedge$ $\vdash_{wf}$ Θ **and**
wfTD-wf: Θ $\vdash_{wf}$ *td* $\Longrightarrow$ $\vdash_{wf}$ Θ
proof(*induct* *rule:wfV-wfC-wfG-wfT-wfTs-wfTh-wfB-wfCE-wfTD.inducts*)

case (*wfV-varI* Θ $\mathcal{B}$ Γ *b c x*)
hence (*x*,*b*,*c*) $\in$ *setG* Γ **using** *lookup-iff lookup-in-g* **by** *presburger*
hence *b* $\in$ *fst*‘*snd*‘*setG* Γ **by** *force*
hence *wfB* Θ $\mathcal{B}$ *b* **using** *wfV-varI* **using** *wfG-lookup-wf* **by** *auto*
then show *?case* **using** *wfV-varI wfV-elims wf-intros* **by** *metis*
next
case (*wfV-litI* Θ $\mathcal{B}$ Γ *l*)
moreover have *wfTh* Θ **using** *wfV-litI* **by** *metis*
ultimately show *?case* **using** *wf-intros base-for-lit.simps l.exhaust* **by** *metis*
next
case (*wfV-pairI* Θ $\mathcal{B}$ Γ *v1 b1 v2 b2*)
then show *?case* **using** *wfB-pairI* **by** *simp*
next
case (*wfV-consI s dclist* Θ *dc x b c* $\mathcal{B}$ Γ *v*)
then show *?case* **using** *wf-intros* **by** *metis*
next
case (*wfTI z* Γ Θ $\mathcal{B}$ *b c*)
then show *?case* **using** *wf-intros b-of.simps wfG-cons2* **by** *metis*
qed(*auto*)

lemma *wfX-wfY2*:
fixes Γ::Γ **and** Γ'::Γ **and** *v*::*v* **and** *e*::*e* **and** *c*::*c* **and** τ::τ **and** *ts*::(*string*$*\tau$) *list* **and** Δ::Δ **and** *s*::*s* **and** *b*::*b* **and** *ftq*::*fun-typ-q* **and** *ft*::*fun-typ* **and** *ce*::*ce* **and** *td*::*type-def* **and** *cs*::*branch-s* **and** *css*::*branch-list*
shows
wfE-wf: Θ ; Φ ; $\mathcal{B}$; Γ ; Δ $\vdash_{wf}$ *e* : *b* $\Longrightarrow$ Θ ; $\mathcal{B}$ $\vdash_{wf}$ Γ $\wedge$ Θ ; $\mathcal{B}$; Γ $\vdash_{wf}$ Δ $\wedge$ $\vdash_{wf}$ Θ $\wedge$ Θ $\vdash_{wf}$ Φ **and**
wfS-wf: Θ ; Φ ; $\mathcal{B}$; Γ ; Δ $\vdash_{wf}$ *s* : *b* $\Longrightarrow$ Θ ; $\mathcal{B}$ $\vdash_{wf}$ Γ $\wedge$ Θ ; $\mathcal{B}$; Γ $\vdash_{wf}$ Δ $\wedge$ $\vdash_{wf}$ Θ $\wedge$ Θ $\vdash_{wf}$ Φ **and**
Θ ; Φ ; $\mathcal{B}$; Γ ; Δ ; *tid* ; *dc* ; *t* $\vdash_{wf}$ *cs* : *b* $\Longrightarrow$ Θ ; $\mathcal{B}$ $\vdash_{wf}$ Γ $\wedge$ Θ ; $\mathcal{B}$; Γ $\vdash_{wf}$ Δ $\wedge$ $\vdash_{wf}$ Θ $\wedge$ Θ $\vdash_{wf}$ Φ **and**
Θ ; Φ ; $\mathcal{B}$; Γ ; Δ ; *tid* ; *dclist* $\vdash_{wf}$ *css* : *b* $\Longrightarrow$ Θ ; $\mathcal{B}$ $\vdash_{wf}$ Γ $\wedge$ Θ ; $\mathcal{B}$; Γ $\vdash_{wf}$ Δ $\wedge$ $\vdash_{wf}$ Θ $\wedge$ Θ

$\vdash_{wf}\ \Phi$ **and**
wfPhi-wf: $\Theta \vdash_{wf} (\Phi::\Phi) \Longrightarrow \vdash_{wf} \Theta$ **and**
wfD-wf: $\Theta\ ;\ \mathcal{B}\ ;\ \Gamma \vdash_{wf} \Delta \Longrightarrow \Theta\ ;\ \mathcal{B} \vdash_{wf} \Gamma \wedge \vdash_{wf} \Theta$ **and**
wfFTQ-wf: $\Theta\ ;\ \Phi \vdash_{wf} ftq \Longrightarrow \Theta \vdash_{wf} \Phi \wedge \vdash_{wf} \Theta$ **and**
wfFT-wf: $\Theta\ ;\ \Phi\ ;\ \mathcal{B} \vdash_{wf} ft \Longrightarrow \Theta \vdash_{wf} \Phi \wedge \vdash_{wf} \Theta$
proof(*induct rule:wfE-wfS-wfCS-wfCSS-wfPhi-wfD-wfFTQ-wfFT.inducts*)
case (*wfS-varI* $\Theta\ \mathcal{B}\ \Gamma\ \tau\ v\ u\ \Delta\ \Phi\ s\ b$)
then show *?case* **using** *wfD-elims* **by** *auto*
next
case (*wfS-assignI* $u\ \tau\ \Delta\ \Theta\ \mathcal{B}\ \Gamma\ \Phi\ v$)
then show *?case* **using** *wf-intros* **by** *metis*
next
case (*wfD-emptyI* $\Theta\ \mathcal{B}\ \Gamma$)
then show *?case* **using** *wfX-wfY1* **by** *auto*
next
case (*wfS-assertI* $\Theta\ \Phi\ \mathcal{B}\ x\ c\ \Gamma\ \Delta\ s\ b$)
then have $\Theta\ ;\ \mathcal{B} \vdash_{wf} \Gamma$ **using** *wfX-wfY1* **by** *auto*
moreover have $\Theta\ ;\ \mathcal{B}\ ;\ \Gamma \vdash_{wf} \Delta$ **using** *wfS-assertI* **by** *auto*
moreover have $\vdash_{wf} \Theta \wedge \Theta \vdash_{wf} \Phi$ **using** *wfS-assertI* **by** *auto*
ultimately show *?case* **by** *auto*
qed(*auto*)

lemmas *wfX-wfY=wfX-wfY1 wfX-wfY2*

lemma *setD-ConsD*:
$ut \in setD\ (ut'\ \#_{\Delta}\ D) = (ut = ut' \vee ut \in setD\ D)$
proof(*induct D rule*: Δ-*induct*)
case *DNil*
then show *?case* **by** *auto*
next
case (*DCons u' t' x2*)
then show *?case* **using** *setD.simps* **by** *auto*
qed

lemma *wfD-wfT*:
fixes $\Delta::\Delta$ **and** $\tau::\tau$
assumes $\Theta\ ;\ \mathcal{B}\ ;\ \Gamma \vdash_{wf} \Delta$
shows $\forall (u,\tau) \in setD\ \Delta.\ \Theta\ ;\ \mathcal{B}\ ;\ \Gamma \vdash_{wf} \tau$
using *assms* **proof**(*induct* Δ *rule*: Δ-*induct*)
case *DNil*
then show *?case* **by** *auto*
next
case (*DCons u' t' x2*)
then show *?case* **using** *wfD-elims DCons setD-ConsD*
by (*metis case-prodI2 set-ConsD*)
qed

lemma *subst-b-lookup-d*:
assumes $u \notin fst\ `\ setD\ \Delta$
shows $u \notin fst\ `\ setD\ \Delta[bv::=b]_{\Delta b}$
using *assms* **proof**(*induct* Δ *rule*: Δ-*induct*)
case *DNil*

then show *?case* **by** *auto*
next
case (*DCons u' t' x2*)
hence $u \neq u'$ **using** *DCons* **by** *simp*
show *?case* **using** *DCons subst-db.simps* **by** *simp*
qed

lemma *wfG-cons-splitI*:
fixes $\Phi::\Phi$ **and** $\Gamma::\Gamma$
assumes Θ ; $\mathcal{B} \vdash_{wf} \Gamma$ **and** *atom* $x \sharp \Gamma$ **and** *wfB* Θ $\mathcal{B}$ b **and**
$c \in \{$ *TRUE*, *FALSE* $\} \longrightarrow \Theta$; $\mathcal{B} \vdash_{wf} \Gamma$ **and**
$c \notin \{$ *TRUE*, *FALSE* $\} \longrightarrow \Theta$;$\mathcal{B}$; $(x,b,C\text{-}true)$ $\#_\Gamma\Gamma \vdash_{wf} c$
shows Θ ; $\mathcal{B} \vdash_{wf} ((x,b,c)$ $\#_\Gamma\Gamma)$
using *wfG-cons1I wfG-cons2I assms* **by** *metis*

lemma *wfG-consI*:
fixes $\Phi::\Phi$ **and** $\Gamma::\Gamma$ **and** $c::c$
assumes Θ ; $\mathcal{B} \vdash_{wf} \Gamma$ **and** *atom* $x \sharp \Gamma$ **and** *wfB* Θ $\mathcal{B}$ b **and**
Θ ; $\mathcal{B}$; $(x,b,C\text{-}true)$ $\#_\Gamma\Gamma \vdash_{wf} c$
shows Θ ; $\mathcal{B} \vdash_{wf} ((x,b,c)$ $\#_\Gamma\Gamma)$
using *wfG-cons1I wfG-cons2I wfG-cons-splitI wfC-trueI assms* **by** *metis*

lemma *wfG-elim2*:
fixes $c::c$
assumes *wfG P* $\mathcal{B}$ $((x,b,c)$ $\#_\Gamma\Gamma)$
shows P ; $\mathcal{B}$; $(x,\ b,\ TRUE)$ $\#_\Gamma$ Γ $\vdash_{wf} c \wedge$ *wfB P* $\mathcal{B}$ b
proof(*cases* $c \in \{TRUE,FALSE\}$)
case *True*
have P ; $\mathcal{B} \vdash_{wf} \Gamma \wedge$ *atom* $x \sharp \Gamma \wedge$ *wfB P* $\mathcal{B}$ b **using** *wfG-elims*(*2*)[*OF assms*] **by** *auto*
hence P ; $\mathcal{B} \vdash_{wf} ((x,b,TRUE)$ $\#_\Gamma\Gamma) \wedge$ *wfB P* $\mathcal{B}$ b **using** *wfG-cons2I* **by** *auto*
thus *?thesis* **using** *wfC-trueI wfC-falseI True* **by** *auto*
next
case *False*
then show *?thesis* **using** *wfG-elims*(*2*)[*OF assms*] **by** *auto*
qed

lemma *wfG-cons-wfC*:
fixes $\Gamma::\Gamma$ **and** $c::c$
assumes Θ ; B $\vdash_{wf} (x,\ b,\ c)$ $\#_\Gamma$ Γ
shows Θ ; B ; $((x,\ b,\ TRUE)$ $\#_\Gamma$ $\Gamma) \vdash_{wf} c$
using *assms wfG-elim2* **by** *auto*

lemma *wfG-wfB*:
assumes *wfG P* $\mathcal{B}$ Γ **and** $b \in$ *fst‘snd‘setG* Γ
shows *wfB P* $\mathcal{B}$ b
using *assms* **proof**(*induct* Γ *rule*:Γ-*induct*)
case *GNil*
then show *?case* **by** *auto*
next
case (*GCons x' b' c' Γ'*)

show *?case* **proof**(*cases* $b{=}b'$)
case *True*
then show *?thesis* **using** *wfG-elim2 GCons* **by** *auto*
next
case *False*
hence $b \in$ *fst'snd'setG* Γ' **using** *GCons* **by** *auto*
moreover have *wfG P* $\mathcal{B}$ Γ' **using** *wfG-cons GCons* **by** *auto*
ultimately show *?thesis* **using** *GCons* **by** *auto*
qed
qed

lemma *wfG-cons-TRUE*:
fixes Γ::Γ **and** b::b
assumes P ; $\mathcal{B} \vdash_{wf} \Gamma$ **and** *atom* $z \sharp \Gamma$ **and** P ; $\mathcal{B} \vdash_{wf} b$
shows P ; $\mathcal{B} \vdash_{wf} (z,\ b,\ TRUE)$ $\#_\Gamma\ \Gamma$
using *wfG-cons2I wfG-wfB assms* **by** *simp*

lemma *wfG-cons-TRUE2*:
assumes P ; $\mathcal{B} \vdash_{wf} (z,b,c)$ $\#_\Gamma\Gamma$ **and** *atom* $z \sharp \Gamma$
shows P ; $\mathcal{B} \vdash_{wf} (z,\ b,\ TRUE)$ $\#_\Gamma\ \Gamma$
using *wfG-cons wfG-cons2I assms* **by** *simp*

lemma *wfG-suffix*:
fixes Γ::Γ
assumes *wfG P* $\mathcal{B}$ (Γ'@Γ)
shows *wfG P* $\mathcal{B}$ Γ
using *assms* **proof**(*induct* Γ' *rule*: Γ-*induct*)
case *GNil*
then show *?case* **by** *auto*
next
case (*GCons x b c* Γ')
hence P ; $\mathcal{B} \vdash_{wf} \Gamma'$ @ Γ **using** *wfG-elims* **by** *auto*
then show *?case* **using** *GCons wfG-elims* **by** *auto*
qed

lemma *wfV-wfCE*:
fixes v::v
assumes Θ ; $\mathcal{B}$; Γ $\vdash_{wf}$ v : b
shows Θ ; $\mathcal{B}$; $\Gamma \vdash_{wf}$ *CE-val* v : b
proof −
have $\Theta \vdash_{wf} ([]$::$\Phi)$ **using** *wfPhi-emptyI wfV-wf wfG-wf assms* **by** *metis*
moreover have Θ ; $\mathcal{B}$; Γ $\vdash_{wf} []_\Delta$ **using** *wfD-emptyI wfV-wf wfG-wf assms* **by** *metis*
ultimately show *?thesis* **using** *wfCE-valI assms* **by** *auto*
qed

8.6 Support

lemma *wf-supp1*:
fixes Γ::Γ **and** Γ'::Γ **and** v::v **and** e::e **and** c::c **and** τ::τ **and** ts::(*string*$*\tau$) *list* **and** Δ::Δ **and** s::s **and** b::b **and** *ftq*::*fun-typ-q* **and** *ft*::*fun-typ* **and** *ce*::*ce* **and** *td*::*type-def* **and** *cs*::*branch-s* **and** *css* ::*branch-list*

shows *wfV-supp*: Θ ; $\mathcal{B}$; Γ $\vdash_{wf}$ v : $b \Longrightarrow$ *supp* $v \subseteq$ *atom-dom* $\Gamma \cup$ *supp* $\mathcal{B}$ **and**
wfC-supp: Θ ; $\mathcal{B}$; $\Gamma \vdash_{wf} c \Longrightarrow$ *supp* $c \subseteq$ *atom-dom* $\Gamma \cup$ *supp* $\mathcal{B}$ **and**
wfG-supp: Θ ; $\mathcal{B} \vdash_{wf} \Gamma \Longrightarrow$ *atom-dom* $\Gamma \subseteq$ *supp* Γ **and**
wfT-supp: Θ ; $\mathcal{B}$; $\Gamma \vdash_{wf} \tau \Longrightarrow$ *supp* $\tau \subseteq$ *atom-dom* $\Gamma \cup$ *supp* $\mathcal{B}$ **and**
wfTs-supp: Θ ; $\mathcal{B}$; $\Gamma \vdash_{wf} ts \Longrightarrow$ *supp* $ts \subseteq$ *atom-dom* $\Gamma \cup$ *supp* $\mathcal{B}$ **and**
wfTh-supp: $\vdash_{wf} \Theta \Longrightarrow$ *supp* $\Theta = \{\}$ **and**
wfB-supp: Θ ; $\mathcal{B} \vdash_{wf} b \Longrightarrow$ *supp* $b \subseteq$ *supp* $\mathcal{B}$ **and**
wfCE-supp: Θ ; $\mathcal{B}$; $\Gamma \vdash_{wf} ce : b \Longrightarrow$ *supp* $ce \subseteq$ *atom-dom* $\Gamma \cup$ *supp* $\mathcal{B}$ **and**
wfTD-supp: Θ $\vdash_{wf} td \Longrightarrow$ *supp* $td \subseteq \{\}$
proof(*induct* *rule:wfV-wfC-wfG-wfT-wfTs-wfTh-wfB-wfCE-wfTD.inducts*)
case (*wfB-consI* Θ *s dclist* $\mathcal{B}$)
then show *?case* **by**(*auto simp add*: *b.supp pure-supp*)
next
case (*wfB-appI* Θ $\mathcal{B}$ *b s bv dclist*)
then show *?case* **by**(*auto simp add*: *b.supp pure-supp*)
next
case (*wfV-varI* Θ $\mathcal{B}$ Γ *b c x*)
then show *?case* **using** *v.supp wfV-elims*
empty-subsetI insert-subset supp-at-base
fresh-dom-free2 lookup-if1
by (*metis sup.coboundedI1*)
next
case (*wfV-litI* Θ $\mathcal{B}$ Γ *l*)
then show *?case* **using** *supp-l-empty v.supp* **by** *simp*
next
case (*wfV-pairI* Θ $\mathcal{B}$ Γ *v1 b1 v2 b2*)
then show *?case* **using** *v.supp wfV-elims* **by** (*metis Un-subset-iff*)
next
case (*wfV-consI s dclist* Θ *dc x b c* $\mathcal{B}$ Γ *v*)
then show *?case* **using** *v.supp wfV-elims*
Un-commute b.supp sup-bot.right-neutral supp-b-empty pure-supp **by** *metis*
next
case (*wfV-conspI typid bv dclist* Θ *dc x b' c* $\mathcal{B}$ Γ *v b*)
then show *?case* **unfolding** *v.supp*
using *wfV-elims*
Un-commute b.supp sup-bot.right-neutral supp-b-empty pure-supp
by (*simp add*: *Un-commute pure-supp sup.coboundedI1*)
next

case (*wfC-eqI* Θ $\mathcal{B}$ Γ *e1 b e2*)
hence *supp e1* $\subseteq$ *atom-dom* $\Gamma \cup$ *supp* $\mathcal{B}$ **using** *c.supp wfC-elims*
image-empty list.set(1) sup-bot.right-neutral **by** (*metis IntI UnE empty-iff subsetCE subsetI*)
moreover have *supp e2* $\subseteq$ *atom-dom* $\Gamma \cup$ *supp* $\mathcal{B}$ **using** *c.supp wfC-elims*
image-empty list.set(1) sup-bot.right-neutral IntI UnE empty-iff subsetCE subsetI
by (*metis wfC-eqI.hyps(4)*)
ultimately show *?case* **using** *c.supp* **by** *auto*
next
case (*wfG-cons1I c* Θ $\mathcal{B}$ Γ *x b*)
then show *?case* **using** *atom-dom.simps dom-supp-g supp-GCons* **by** *metis*
next
case (*wfG-cons2I c* Θ $\mathcal{B}$ Γ *x b*)
then show *?case* **using** *atom-dom.simps dom-supp-g supp-GCons* **by** *metis*

next
case *wfTh-emptyI*
then show *?case* **by** (*simp add*: *supp-Nil*)
next
case (*wfTh-consI* Θ *lst*)
then show *?case* **using** *supp-Cons* **by** *fast*
next
case (*wfTD-simpleI* Θ *lst s*)
then have *supp* (*AF-typedef s lst*) = *supp lst* ∪ *supp s* **using** *type-def.supp* **by** *auto*
then show *?case* **using** *wfTD-simpleI pure-supp*
by (*simp add*: *pure-supp supp-Cons supp-at-base*)
next
case (*wfTD-poly* Θ *bv lst s*)
then have *supp* (*AF-typedef-poly s bv lst*) = *supp lst* − { *atom bv* } ∪ *supp s* **using** *type-def.supp*
by *auto*
then show *?case* **using** *wfTD-poly pure-supp*
by (*simp add*: *pure-supp supp-Cons supp-at-base*)
next
case (*wfTs-nil* Θ $\mathcal{B}$ Γ)
then show *?case* **using** *supp-Nil* **by** *auto*
next
case (*wfTs-cons* Θ $\mathcal{B}$ Γ τ *dc ts*)
then show *?case* **using** *supp-Cons supp-Pair pure-supp*[*of dc*] **by** *blast*
next
case (*wfCE-valI* Θ $\mathcal{B}$ Γ *v b*)
thus *?case* **using** *ce.supp wfCE-elims* **by** *simp*
next
case (*wfCE-plusI* Θ $\mathcal{B}$ Γ *v1 v2*)
hence *supp* (*CE-op Plus v1 v2*) ⊆ *atom-dom* Γ ∪ *supp* $\mathcal{B}$ **using** *ce.supp pure-supp*
by (*simp add*: *wfCE-plusI opp.supp*)
then show *?case* **using** *ce.supp wfCE-elims UnCI subsetCE subsetI x-not-in-b-set* **by** *auto*
next
case (*wfCE-leqI* Θ $\mathcal{B}$ Γ *v1 v2*)
hence *supp* (*CE-op LEq v1 v2*) ⊆ *atom-dom* Γ ∪ *supp* $\mathcal{B}$ **using** *ce.supp pure-supp*
by (*simp add*: *wfCE-plusI opp.supp*)
then show *?case* **using** *ce.supp wfE-elims UnCI subsetCE subsetI x-not-in-b-set* **by** *auto*
next
case (*wfCE-fstI* Θ $\mathcal{B}$ Γ *v1 b1 b2*)
thus *?case* **using** *ce.supp wfCE-elims* **by** *simp*
next
case (*wfCE-sndI* Θ $\mathcal{B}$ Γ *v1 b1 b2*)
thus *?case* **using** *ce.supp wfCE-elims* **by** *simp*
next
case (*wfCE-concatI* Θ $\mathcal{B}$ Γ *v1 v2*)
thus *?case* **using** *ce.supp wfCE-elims* **by** *simp*
next
case (*wfCE-lenI* Θ $\mathcal{B}$ Γ *v1*)
thus *?case* **using** *ce.supp wfCE-elims* **by** *simp*
next
case (*wfTI z* Θ $\mathcal{B}$ Γ *b c*)
hence *supp c* ⊆ *supp z* ∪ *atom-dom* Γ ∪ *supp* $\mathcal{B}$ **using** *supp-at-base dom-cons* **by** *metis*
moreover have *supp b* ⊆ *supp* $\mathcal{B}$ **using** *wfTI* **by** *auto*

ultimately have *supp* $\{\!|\ z : b\ \mid c\ |\!\} \subseteq$ *atom-dom* $\Gamma \cup$ *supp* $\mathcal{B}$ **using** τ*.supp supp-at-base* **by** *force*
thus *?case* **by** *auto*
qed(*auto*)

lemma *wf-supp2*:
fixes Γ::Γ **and** Γ'::Γ **and** *v*::*v* **and** *e*::*e* **and** *c*::*c* **and** τ::τ **and**
ts::(*string*$*\tau$) *list* **and** Δ::Δ **and** *s*::*s* **and** *b*::*b* **and** *ftq*::*fun-typ-q* **and**
ft::*fun-typ* **and** *ce*::*ce* **and** *td*::*type-def* **and** *cs*::*branch-s* **and** *css* ::*branch-list*
shows
wfE-supp: Θ ; Φ ; $\mathcal{B}$; Γ ; $\Delta \vdash_{wf} e : b \Longrightarrow$ (*supp e* $\subseteq$ *atom-dom* $\Gamma \cup$ *supp* $\mathcal{B} \cup$ *atom* ' *fst* ' *setD* Δ) **and**
wfS-supp: Θ ; Φ ; $\mathcal{B}$; Γ ; $\Delta \vdash_{wf} s : b \Longrightarrow$ *supp s* $\subseteq$ *atom-dom* $\Gamma \cup$ *atom* ' *fst* ' *setD* $\Delta \cup$ *supp* $\mathcal{B}$ **and**
Θ ; Φ ; $\mathcal{B}$; Γ ; Δ ; *tid* ; *dc* ; $t \vdash_{wf}$ *cs* : $b \Longrightarrow$ *supp cs* $\subseteq$ *atom-dom* $\Gamma \cup$ *atom* ' *fst* ' *setD* $\Delta \cup$ *supp* $\mathcal{B}$ **and**
Θ ; Φ ; $\mathcal{B}$; Γ ; Δ ; *tid* ; *dclist* $\vdash_{wf}$ *css* : $b \Longrightarrow$ *supp css* $\subseteq$ *atom-dom* $\Gamma \cup$ *atom* ' *fst* ' *setD* $\Delta \cup$ *supp* $\mathcal{B}$ **and**
wfPhi-supp: $\Theta \vdash_{wf}$ (Φ::Φ) $\Longrightarrow$ *supp* $\Phi = \{\}$ **and**
wfD-supp: Θ ; $\mathcal{B}$; $\Gamma \vdash_{wf} \Delta \Longrightarrow$ *supp* $\Delta \subseteq$ *atom*'*fst*'(*setD* Δ) $\cup$ *atom-dom* $\Gamma \cup$ *supp* $\mathcal{B}$ **and**
Θ ; Φ $\vdash_{wf}$ *ftq* $\Longrightarrow$ *supp ftq* $= \{\}$ **and**
Θ ; Φ ; $\mathcal{B} \vdash_{wf}$ *ft* $\Longrightarrow$ *supp ft* $\subseteq$ *supp* $\mathcal{B}$
proof(*induct* *rule:wfE-wfS-wfCS-wfCSS-wfPhi-wfD-wfFTQ-wfFT.inducts*)
case (*wfE-valI* Θ Φ $\mathcal{B}$ Γ Δ *v b*)
hence *supp* (*AE-val v*) $\subseteq$ *atom-dom* $\Gamma \cup$ *supp* $\mathcal{B}$ **using** *e.supp wf-supp1* **by** *simp*
then show *?case* **using** *e.supp wfE-elims UnCI subsetCE subsetI x-not-in-b-set* **by** *metis*
next
case (*wfE-plusI* Θ Φ $\mathcal{B}$ Γ Δ *v1 v2*)
hence *supp* (*AE-op Plus v1 v2*) $\subseteq$ *atom-dom* $\Gamma \cup$ *supp* $\mathcal{B}$
using *wfE-plusI opp.supp wf-supp1 e.supp pure-supp Un-least*
by (*metis sup-bot.left-neutral*)

then show *?case* **using** *e.supp wfE-elims UnCI subsetCE subsetI x-not-in-b-set* **by** *auto*
next
case (*wfE-leqI* Θ Φ $\mathcal{B}$ Γ Δ *v1 v2*)
hence *supp* (*AE-op LEq v1 v2*) $\subseteq$ *atom-dom* $\Gamma \cup$ *supp* $\mathcal{B}$ **using** *e.supp pure-supp Un-least sup-bot.left-neutral* **using** *opp.supp wf-supp1* **by** *auto*
then show *?case* **using** *e.supp wfE-elims UnCI subsetCE subsetI x-not-in-b-set* **by** *auto*
next
case (*wfE-fstI* Θ Φ $\mathcal{B}$ Γ Δ *v1 b1 b2*)
hence *supp* (*AE-fst v1*) $\subseteq$ *atom-dom* $\Gamma \cup$ *supp* $\mathcal{B}$ **using** *e.supp pure-supp sup-bot.left-neutral* **using** *opp.supp wf-supp1* **by** *auto*
then show *?case* **using** *e.supp wfE-elims UnCI subsetCE subsetI x-not-in-b-set* **by** *auto*
next
case (*wfE-sndI* Θ Φ $\mathcal{B}$ Γ Δ *v1 b1 b2*)
hence *supp* (*AE-snd v1*) $\subseteq$ *atom-dom* $\Gamma \cup$ *supp* $\mathcal{B}$ **using** *e.supp pure-supp wfE-plusI opp.supp wf-supp1* **by** (*metis Un-least*)
then show *?case* **using** *e.supp wfE-elims UnCI subsetCE subsetI x-not-in-b-set* **by** *auto*
next
case (*wfE-concatI* Θ Φ $\mathcal{B}$ Γ Δ *v1 v2*)
hence *supp* (*AE-concat v1 v2*) $\subseteq$ *atom-dom* $\Gamma \cup$ *supp* $\mathcal{B}$ **using** *e.supp pure-supp wfE-plusI opp.supp wf-supp1* **by** (*metis Un-least*)
then show *?case* **using** *e.supp wfE-elims UnCI subsetCE subsetI x-not-in-b-set* **by** *auto*

next
case (*wfE-splitI* Θ Φ $\mathcal{B}$ Γ Δ *v1 v2*)
hence *supp* (*AE-split v1 v2*) ⊆ *atom-dom* Γ ∪ *supp* $\mathcal{B}$ **using** *e.supp pure-supp wfE-plusI opp.supp wf-supp1* **by** (*metis Un-least*)
then show *?case* **using** *e.supp wfE-elims UnCI subsetCE subsetI x-not-in-b-set* **by** *auto*
next
case (*wfE-lenI* Θ Φ $\mathcal{B}$ Γ Δ *v1*)
hence *supp* (*AE-len v1*) ⊆ *atom-dom* Γ ∪ *supp* $\mathcal{B}$ **using** *e.supp pure-supp*
using *e.supp pure-supp sup-bot.left-neutral* **using** *opp.supp wf-supp1* **by** *auto*
then show *?case* **using** *e.supp wfE-elims UnCI subsetCE subsetI x-not-in-b-set* **by** *auto*
next
case (*wfE-appI* Θ Φ $\mathcal{B}$ Γ Δ *f x b c τ s v*)
then obtain *b* **where** Θ ; $\mathcal{B}$; Γ ⊢$_{wf}$ *v* : *b* **using** *wfE-elims* **by** *metis*
hence *supp v* ⊆ *atom-dom* Γ ∪ *supp* $\mathcal{B}$ **using** *wfE-appI wf-supp1* **by** *metis*
hence *supp* (*AE-app f v*) ⊆ *atom-dom* Γ ∪ *supp* $\mathcal{B}$ **using** *e.supp pure-supp* **by** *fast*
then show *?case* **using** *e.supp*(*2*) *UnCI subsetCE subsetI wfE-appI* **using** *b.supp*(*3*) *pure-supp x-not-in-b-set* **by** *auto*
next
case (*wfE-appPI* Θ Φ $\mathcal{B}$ Γ Δ *b′ bv v τ f xa ba ca s*)
then obtain *b* **where** Θ ; $\mathcal{B}$; Γ ⊢$_{wf}$ *v* : (*b*[*bv*::=*b′*]$_b$) **using** *wfE-elims* **by** *metis*
hence *supp v* ⊆ *atom-dom* Γ ∪ *supp* $\mathcal{B}$ **using** *wfE-appPI wf-supp1* **by** *auto*
moreover have *supp b′* ⊆ *supp* $\mathcal{B}$ **using** *wf-supp1*(*7*) *wfE-appPI* **by** *simp*
ultimately show *?case* **unfolding** *e.supp* **using** *wfE-appPI pure-supp* **by** *fast*
next
case (*wfE-mvarI* Θ Φ $\mathcal{B}$ Γ Δ *u τ*)
then obtain *τ* **where** (*u*,*τ*) ∈ *setD* Δ **using** *wfE-elims*(*10*) **by** *metis*
hence *atom u* ∈ *atom‘fst‘setD* Δ **by** *force*
hence *supp* (*AE-mvar u*) ⊆ *atom‘fst‘setD* Δ **using** *e.supp*
by (*simp add*: *supp-at-base*)
thus *?case* **using** *UnCI subsetCE subsetI e.supp wfE-mvarI supp-at-base subsetCE supp-at-base u-not-in-b-set*

by (*simp add*: *supp-at-base*)
next
case (*wfS-valI* Θ Φ $\mathcal{B}$ Γ *v b* Δ)
then show *?case* **using** *wf-supp1*
by (*metis s-branch-s-branch-list.supp*(*1*) *sup.coboundedI2 sup-assoc sup-commute*)
next
case (*wfS-letI* Θ Φ $\mathcal{B}$ Γ Δ *e b′ x s b*)
then show *?case* **by** *auto*
next
case (*wfS-let2I* Θ Φ $\mathcal{B}$ Γ Δ *s1 τ x s2 b*)
then show *?case* **unfolding** *s-branch-s-branch-list.supp* (*3*) **using** *wf-supp1*(*4*)[*OF wfS-let2I*(*3*)] **by** *auto*
next
case (*wfS-ifI* Θ $\mathcal{B}$ Γ *v* Φ Δ *s1 b s2*)
then show *?case* **using** *wf-supp1*(*1*)[*OF wfS-ifI*(*1*)] **by** *auto*
next
case (*wfS-varI* Θ $\mathcal{B}$ Γ *τ v u* Δ Φ *s b*)
then show *?case* **using** *wf-supp1*(*1*)[*OF wfS-varI*(*2*)] *wf-supp1*(*4*)[*OF wfS-varI*(*1*)] **by** *auto*
next
next
case (*wfS-assignI u τ* Δ Θ $\mathcal{B}$ Γ Φ *v*)

hence *supp u* ⊆ *atom ‘ fst ‘ setD* Δ **proof**(*induct* Δ *rule*:Δ-*induct*)
case *DNil*
then show *?case* **by** *auto*
next
case (*DCons u′ t′* Δ′)
show *?case* **proof**(*cases u*=*u′*)
case *True*
then show *?thesis* **using** *setG.simps DCons supp-at-base* **by** *fastforce*
next
case *False*
then show *?thesis* **using** *setG.simps DCons supp-at-base wfS-assignI*
by (*metis empty-subsetI fstI image-eqI insert-subset*)
qed
qed
then show *?case* **using** *s-branch-s-branch-list.supp*(*8*) *wfS-assignI wf-supp1*(*1*)[*OF wfS-assignI*(*6*)]
by *auto*
next
case (*wfS-matchI* Θ $\mathcal{B}$ Γ *v tid dclist* Δ Φ *cs b*)
then show *?case* **using** *wf-supp1*(*1*)[*OF wfS-matchI*(*1*)] **by** *auto*
next
case (*wfS-branchI* Θ Φ $\mathcal{B}$ *x* τ Γ Δ *s b tid dc*)
moreover have *supp s* ⊆ *supp x* ∪ *atom-dom* Γ ∪ *atom ‘ fst ‘ setD* Δ ∪ *supp* $\mathcal{B}$
using *dom-cons supp-at-base wfS-branchI* **by** *auto*
moreover hence *supp s* − *set* [*atom x*] ⊆ *atom-dom* Γ ∪ *atom ‘ fst ‘ setD* Δ ∪ *supp* $\mathcal{B}$ **using**
supp-at-base **by** *force*
ultimately have
(*supp s* − *set* [*atom x*]) ∪ (*supp dc*) ⊆ *atom-dom* Γ ∪ *atom ‘ fst ‘ setD* Δ ∪ *supp* $\mathcal{B}$
by (*simp add*: *pure-supp*)
thus *?case* **using** *s-branch-s-branch-list.supp*(*2*) **by** *auto*
next
case (*wfD-emptyI* Θ $\mathcal{B}$ Γ)
then show *?case* **using** *supp-DNil* **by** *auto*
next
case (*wfD-cons* Θ $\mathcal{B}$ Γ Δ τ *u*)
have *supp* ((*u*, τ) $\#_{\Delta}$ Δ) = *supp u* ∪ *supp* τ ∪ *supp* Δ **using** *supp-DCons supp-Pair* **by** *metis*
also have ... ⊆ *supp u* ∪ *atom ‘ fst ‘ setD* Δ ∪ *atom-dom* Γ ∪ *supp* $\mathcal{B}$
using *wfD-cons wf-supp1*(*4*)[*OF wfD-cons*(*3*)] **by** *auto*
also have ... ⊆ *atom ‘ fst ‘ setD* ((*u*, τ) $\#_{\Delta}$ Δ) ∪ *atom-dom* Γ ∪ *supp* $\mathcal{B}$ **using** *supp-at-base* **by** *auto*
finally show *?case* **by** *auto*
next
case (*wfPhi-emptyI* Θ)
then show *?case* **using** *supp-Nil* **by** *auto*
next
case (*wfPhi-consI f* Θ Φ *ft*)
then show *?case* **using** *fun-def.supp*
by (*simp add*: *pure-supp supp-Cons*)
next
case (*wfFTI* Θ *B′ b* Φ *x c s* τ)
have *supp* (*AF-fun-typ x b c* τ *s*) = *supp c* ∪ (*supp* τ ∪ *supp s*) − *set* [*atom x*] ∪ *supp b* **using**
fun-typ.supp **by** *auto*
thus *?case* **using** *wfFTI wf-supp1*
proof −

have *f1*: *supp* $\tau \subseteq \{atom\ x\} \cup$ *atom-dom GNil* $\cup$ *supp* B'
using *dom-cons wfFTI.hyps*(6) *wf-supp1*(4) **by** *blast*
have *supp* $b \subseteq$ *supp* B'
using *wfFTI.hyps*(1) *wf-supp1*(7) **by** *blast*
then show *?thesis*
using *f1* ‹*supp* (*AF-fun-typ x b c* τ *s*) = *supp c* $\cup$ (*supp* τ $\cup$ *supp s*) $-$ *set* [*atom x*] $\cup$ *supp b*›
wfFTI.hyps(4) *wfFTI.hyps*(5) **by** *auto*
qed
next
case (*wfFTNone* Θ Φ *ft*)
then show *?case* **by** (*simp add*: *fun-typ-q.supp*(2))
next
case (*wfFTSome* Θ Φ *bv ft*)
then show *?case* **using** *fun-typ-q.supp*
by (*simp add*: *supp-at-base*)
next
case (*wfS-assertI* Θ Φ $\mathcal{B}$ *x c* Γ Δ *s b*)
then have *supp c* $\subseteq$ *atom-dom* Γ $\cup$ *atom* ‘ *fst* ‘ *setD* Δ $\cup$ *supp* $\mathcal{B}$ **using** *wf-supp1*
by (*metis Un-assoc Un-commute le-supI2*)
moreover have *supp s* $\subseteq$ *atom-dom* Γ $\cup$ *atom* ‘ *fst* ‘ *setD* Δ $\cup$ *supp* $\mathcal{B}$ **proof**
fix *z*
assume $*$:*z* $\in$ *supp s*
have $**$:*atom x* $\notin$ *supp s* **using** *wfS-assertI fresh-prodN fresh-def* **by** *metis*
have *z* $\in$ *atom-dom* ((*x*, *B-bool*, *c*) $\#_\Gamma$ Γ) $\cup$ *atom* ‘ *fst* ‘ *setD* Δ $\cup$ *supp* $\mathcal{B}$ **using** *wfS-assertI* $*$ **by**
blast
have *z* $\in$ *atom-dom* ((*x*, *B-bool*, *c*) $\#_\Gamma$ Γ) $\Longrightarrow$ *z* $\in$ *atom-dom* Γ **using** $*$ $**$ **by** *auto*
thus *z* $\in$ *atom-dom* Γ $\cup$ *atom* ‘ *fst* ‘ *setD* Δ $\cup$ *supp* $\mathcal{B}$ **using** $*$ $**$
using ‹*z* $\in$ *atom-dom* ((*x*, *B-bool*, *c*) $\#_\Gamma$ Γ) $\cup$ *atom* ‘ *fst* ‘ *setD* Δ $\cup$ *supp* $\mathcal{B}$› **by** *blast*
qed
ultimately show *?case* **by** *auto*
qed(*auto*)

lemmas *wf-supp* = *wf-supp1 wf-supp2*

lemma *wfV-supp-nil*:
fixes *v*::*v*
assumes *P* ; {||} ; *GNil* $\vdash_{wf}$ *v* : *b*
shows *supp v* = {}
using *wfV-supp*[*of P* {||} *GNil v b*] *dom.simps setG.simps*
using *assms* **by** *auto*

lemma *wfT-TRUE-aux*:
assumes *wfG P* $\mathcal{B}$ Γ **and** *atom z* $\sharp$ (*P*, $\mathcal{B}$, Γ) **and** *wfB P* $\mathcal{B}$ *b*
shows *wfT P* $\mathcal{B}$ Γ ({| *z* : *b* | *TRUE* |})
proof (*rule*)
show ‹ *atom z* $\sharp$ (*P*, $\mathcal{B}$, Γ)› **using** *assms* **by** *auto*
show ‹ *P* ; $\mathcal{B}$ $\vdash_{wf}$ *b* › **using** *assms* **by** *auto*
show ‹ *P* ;$\mathcal{B}$; (*z*, *b*, *TRUE*) $\#_\Gamma$ Γ $\vdash_{wf}$ *TRUE* › **using** *wfG-cons2I wfC-trueI assms* **by** *auto*
qed

lemma *wfT-TRUE*:
assumes *wfG P* $\mathcal{B}$ Γ **and** *wfB P* $\mathcal{B}$ *b*

shows *wfT P* $\mathcal{B}$ Γ ($\{\!|$ *z* : *b* | *TRUE* $|\!\}$)
proof −
obtain *z′*::*x* **where** ∗:*atom z′* ♯ (*P*, $\mathcal{B}$, Γ) **using** *obtain-fresh* **by** *metis*
hence $\{\!|$ *z* : *b* | *TRUE* $|\!\}$ = $\{\!|$ *z′* : *b* | *TRUE* $|\!\}$ **by** *auto*
thus *?thesis* **using** *wfT-TRUE-aux assms* ∗ **by** *metis*
qed

lemma *phi-flip-eq*:
assumes *wfPhi T P*
shows (*x* ↔ *xa*) ∙ *P* = *P*
using *wfPhi-supp*[*OF assms*] *flip-fresh-fresh fresh-def* **by** *blast*

lemma *wfC-supp-cons*:
fixes *c′*::*c* **and** *G*::Γ
assumes *P* ; $\mathcal{B}$; (*x′*, *b′* , *TRUE*) #$_\Gamma$ *G* ⊢$_{wf}$ *c′*
shows *supp c′* ⊆ *atom-dom G* ∪ *supp x′* ∪ *supp* $\mathcal{B}$ **and** *supp c′* ⊆ *supp G* ∪ *supp x′* ∪ *supp* $\mathcal{B}$
proof −
show *supp c′* ⊆ *atom-dom G* ∪ *supp x′* ∪ *supp* $\mathcal{B}$
using *wfC-supp*[*OF assms*] *dom-cons supp-at-base* **by** *blast*
moreover have *atom-dom G* ⊆ *supp G*
by (*meson assms wfC-wf wfG-cons wfG-supp*)
ultimately show *supp c′* ⊆ *supp G* ∪ *supp x′* ∪ *supp* $\mathcal{B}$ **using** *wfG-supp assms wfG-cons wfC-wf* **by** *fast*
qed

lemma *wfG-dom-supp*:
fixes *x*::*x*
assumes *wfG P* $\mathcal{B}$ *G*
shows *atom x* ∈ *atom-dom G* ⟷ *atom x* ∈ *supp G*
using *assms* **proof**(*induct G rule*: Γ-*induct*)
case *GNil*
then show *?case* **using** *dom.simps supp-of-atom-list*
using *supp-GNil* **by** *auto*
next
case (*GCons x′ b′ c′ G*)
thm *wfG-cons*

show *?case* **proof**(*cases x′* = *x*)
case *True*
then show *?thesis* **using** *dom.simps supp-of-atom-list supp-at-base*
using *supp-GCons* **by** *auto*
next
case *False*
have (*atom x* ∈ *atom-dom* ((*x′*, *b′*, *c′*) #$_\Gamma$ *G*)) = (*atom x* ∈ *atom-dom G*) **using** *atom-dom.simps False* **by** *simp*
also have ... = (*atom x* ∈ *supp G*) **using** *GCons wfG-elims* **by** *metis*
also have ... = (*atom x* ∈ (*supp* (*x′*, *b′*, *c′*) ∪ *supp G*)) **proof**
show *atom x* ∈ *supp G* ⟹ *atom x* ∈ *supp* (*x′*, *b′*, *c′*) ∪ *supp G* **by** *auto*
assume *atom x* ∈ *supp* (*x′*, *b′*, *c′*) ∪ *supp G*
then consider *atom x* ∈ *supp* (*x′*, *b′*, *c′*) | *atom x* ∈ *supp G* **by** *auto*
then show *atom x* ∈ *supp G* **proof**(*cases*)
case *1*

assume *atom x* ∈ *supp* (*x'*, *b'*, *c'*)
hence *atom x* ∈ *supp c'* **using** *supp-triple False supp-b-empty supp-at-base* **by** *force*

moreover have *P* ; *B* ; (*x'*, *b'* , *TRUE*) #Γ *G* ⊢*wf* *c'* **using** *wfG-elim2 GCons* **by** *simp*
moreover hence *supp c'* ⊆ *supp G* ∪ *supp x'* ∪ *supp B* **using** *wfC-supp-cons* **by** *auto*
ultimately have *atom x* ∈ *supp G* ∪ *supp x'* **using** *x-not-in-b-set* **by** *auto*
then show *?thesis* **using** *False supp-at-base* **by** (*simp add*: *supp-at-base*)
next
case *2*
then show *?thesis* **by** *simp*
qed
qed

also have ... = (*atom x* ∈ *supp* ((*x'*, *b'*, *c'*) #Γ *G*)) **using** *supp-at-base False supp-GCons* **by** *simp*
finally show *?thesis* **by** *simp*
qed
qed

lemma *wfG-atoms-supp-eq* :
fixes *x*::*x*
assumes *wfG P B G*
shows *atom x* ∈ *atom-dom G* ⟷ *atom x* ∈ *supp G*
using *wfG-dom-supp assms* **by** *auto*

lemma *beta-flip-eq*:
fixes *x*::*x* **and** *xa*::*x* **and** *B*::*B*
shows (*x* ↔ *xa*) • *B* = *B*
proof −
thm *x-not-in-b-set*
have *atom x* ♯ *B* ∧ *atom xa* ♯ *B* **using** *x-not-in-b-set fresh-def supp-set* **by** *metis*
thus *?thesis* **by** (*simp add*: *flip-fresh-fresh fresh-def*)
qed

lemma *theta-flip-eq2*:
assumes ⊢*wf* Θ
shows (*z* ↔ *za*) • Θ = Θ
proof −
have *supp* Θ = {} **using** *wfTh-supp assms* **by** *simp*
thus *?thesis*
by (*simp add*: *flip-fresh-fresh fresh-def*)
qed

lemma *theta-flip-eq*:
assumes *wfTh* Θ
shows (*x* ↔ *xa*) • Θ = Θ
using *wfTh-supp flip-fresh-fresh fresh-def*
by (*simp add*: *assms theta-flip-eq2*)

lemma *wfT-wfC*:
fixes *c*::*c*
assumes Θ ; *B* ; Γ ⊢*wf* {| *z* : *b* | *c* |} **and** *atom z* ♯ Γ

```
  shows Θ ; B ; (z,b,TRUE)  #_Γ Γ ⊢_wf c
proof −

  obtain za ba ca where ∗:{| z : b | c |} = {| za : ba | ca |} ∧ atom za ♯ (Θ,B,Γ) ∧  Θ ; B ; (za, ba,
TRUE)  #_Γ Γ ⊢_wf ca
    using wfT-elims[OF assms(1)] by metis
  hence c1: [[atom z]]lst. c = [[atom za]]lst. ca using τ.eq-iff by meson
  show ?thesis proof(cases z=za)
    case True
    hence ca = c using  c1  by (simp add: Abs1-eq-iff(3))
    then show ?thesis using ∗ True by simp
  next
    case False
    have ⊢_wf Θ using wfT-wf wfG-wf assms by metis
    moreover have atom za ♯ Γ using ∗ fresh-prodN by auto
    ultimately have  Θ ; B ; (z ↔ za ) · (za, ba, TRUE)  #_Γ Γ ⊢_wf (z ↔ za ) · ca
      using wfC.eqvt theta-flip-eq2  beta-flip-eq ∗ GCons-eqvt assms flip-fresh-fresh  by metis
    moreover have atom z ♯ ca
    proof −
     have supp ca ⊆ atom-dom Γ ∪ { atom za } ∪ supp B using ∗ wfC-supp atom-dom.simps setG.simps
by fastforce
      moreover have atom z ∉ atom-dom Γ  using assms fresh-def wfT-wf  wfG-dom-supp wfC-supp
by metis
      moreover hence  atom z ∉ atom-dom Γ ∪ { atom za } using False by simp
      moreover have atom z ∉ supp B using x-not-in-b-set by simp
      ultimately show ?thesis using fresh-def False by fast
    qed
    moreover hence  (z ↔ za ) · ca = c using type-eq-subst-eq1(3)  ∗ by metis
    ultimately show  ?thesis using assms G-cons-flip-fresh ∗ by auto
  qed
qed

lemma u-not-in-dom-g:
  fixes u::u
  shows  atom u ∉ atom-dom  G
  using setG.simps atom-dom.simps u-not-in-x-atoms by auto

lemma bv-not-in-dom-g:
  fixes bv::bv
  shows  atom bv ∉ atom-dom  G
  using setG.simps atom-dom.simps u-not-in-x-atoms by auto
```

An important lemma that confirms that Γ does not rely on mutable variables

```
lemma u-not-in-g:
  fixes u::u
  assumes wfG Θ B G
  shows  atom u ∉ supp G
using assms proof(induct G rule: Γ-induct)
case GNil
  then show ?case using supp-GNil fresh-def
    using fresh-set-empty by fastforce
next
```

case (*GCons x b c* Γ′)
moreover hence *atom u* ∉ *supp b* **using**
wfB-supp wfC-supp u-not-in-x-atoms wfG-elims wfX-wfY **by** *auto*
moreover hence *atom u* ∉ *supp x* **using** *u-not-in-x-atoms supp-at-base* **by** *blast*
moreover hence *atom u* ∉ *supp c* **proof** −
have Θ ; *B* ; (*x*, *b*, *TRUE*) #Γ Γ′ ⊢*wf c* **using** *wfG-cons-wfC GCons* **by** *simp*
hence *supp c* ⊆ *atom-dom* ((*x*, *b*, *TRUE*) #Γ Γ′) ∪ *supp B* **using** *wfC-supp* **by** *blast*
thus *?thesis* **using** *u-not-in-dom-g u-not-in-b-atoms*
using *u-not-in-b-set* **by** *auto*
qed
ultimately have *atom u* ∉ *supp* (*x*,*b*,*c*) **using** *supp-Pair* **by** *simp*
thus *?case* **using** *supp-GCons GCons wfG-elims* **by** *blast*
qed

lemma *u-not-in-t*:
fixes *u*::*u*
assumes *wfT* Θ *B G* τ
shows *atom u* ∉ *supp* τ
proof −
have *supp* τ ⊆ *atom-dom G* ∪ *supp B* **using** *wfT-supp assms* **by** *auto*
thus *?thesis* **using** *u-not-in-dom-g u-not-in-b-set* **by** *blast*
qed

lemma *bv-not-in-bset-supp*:
fixes *bv*::*bv*
assumes *bv* |∉| *B*
shows *atom bv* ∉ *supp B*
proof −
have ∗:*supp B* = *fset* (*fimage atom B*)
by (*metis fimage.rep-eq finite-fset supp-finite-set-at-base supp-fset*)
thus *?thesis* **using** *assms*
using *notin-fset* **by** *fastforce*
qed

lemma *wfT-supp-c*:
fixes $\mathcal{B}$::$\mathcal{B}$ **and** *z*::*x*
assumes *wfT P* $\mathcal{B}$ Γ ({|* *z* : *b* | *c* *|})
shows *supp c* − { *atom z* } ⊆ *atom-dom* Γ ∪ *supp* $\mathcal{B}$
using *wf-supp* τ.*supp assms*
by (*metis Un-subset-iff empty-set list.simps*(*15*))

lemma *wfG-wfC*[*ms-wb*]:
assumes *wfG P* $\mathcal{B}$ ((*x*,*b*,*c*) #Γ Γ)
shows *wfC P* $\mathcal{B}$ ((*x*,*b*,*TRUE*) #Γ Γ) *c*
using *assms* **proof**(*cases c* ∈ {*TRUE*,*FALSE*})
case *True*
have *atom x* ♯ Γ ∧ *wfG P* $\mathcal{B}$ Γ ∧ *wfB P* $\mathcal{B}$ *b* **using** *wfG-cons assms* **by** *auto*
hence *wfG P* $\mathcal{B}$ ((*x*,*b*,*TRUE*) #Γ Γ) **using** *wfG-cons2I* **by** *auto*
then show *?thesis* **using** *wfC-trueI wfC-falseI True* **by** *auto*
next

case *False*
then show *?thesis* **using** *wfG-elims assms* **by** *blast*
qed

lemma *wfT-wf-cons*:
assumes *wfT P* $\mathcal{B}$ Γ $\{\!|$ *z* : *b* | *c* $|\!\}$ **and** *atom z* $\sharp$ Γ
shows *wfG P* $\mathcal{B}$ (*z*,*b*,*c*) $\#_{\Gamma}\Gamma$)
using *assms* **proof**(*cases c* $\in$ { *TRUE*,*FALSE* })
case *True*
then show *?thesis* **using** *wfT-wfC wfC-wf wfG-wfB wfG-cons2I assms wfT-wf* **by** *fastforce*
next
case *False*
then show *?thesis* **using** *wfT-wfC wfC-wf wfG-wfB wfG-cons1I wfT-wf wfT-wfC assms* **by** *fastforce*
qed

lemma *wfV-b-fresh*:
fixes *b*::*b* **and** *v*::*v* **and** *bv*::*bv*
assumes Θ ; $\mathcal{B}$; $\Gamma \vdash_{wf} v$: *b* **and** *bv* $|\notin|$ $\mathcal{B}$
shows *atom bv* $\sharp$ *v*
using *wfV-supp bv-not-in-dom-g fresh-def assms bv-not-in-bset-supp* **by** *blast*

lemma *wfCE-b-fresh*:
fixes *b*::*b* **and** *ce*::*ce* **and** *bv*::*bv*
assumes Θ ; $\mathcal{B}$; $\Gamma \vdash_{wf}$ *ce*: *b* **and** *bv* $|\notin|$ $\mathcal{B}$
shows *atom bv* $\sharp$ *ce*
using *bv-not-in-dom-g fresh-def assms bv-not-in-bset-supp wf-supp1*(*8*) **by** *fast*

8.7 Freshness

lemma *wfG-fresh-x*:
fixes Γ::Γ **and** *z*::*x*
assumes Θ ; $\mathcal{B} \vdash_{wf} \Gamma$ **and** *atom z* $\sharp$ Γ
shows *atom z* $\sharp$ (Θ,$\mathcal{B}$, Γ)
unfolding *fresh-prodN* **apply**(*intro conjI*)
using *wf-supp1 wfX-wfY assms fresh-def x-not-in-b-set* **by**(*metis empty-iff*)+

lemma *wfG-wfT*:
assumes *wfG P* $\mathcal{B}$ ((*x*, *b*, *c*[*z*::=*V-var x*]$_{cv}$) $\#_{\Gamma}$ *G*) **and** *atom x* $\sharp$ *c*
shows *P* ; $\mathcal{B}$; *G* $\vdash_{wf}$ $\{\!|$ *z* : *b* | *c* $|\!\}$
proof −
have *P* ; $\mathcal{B}$; (*x*, *b*, *TRUE*) $\#_{\Gamma}$ *G* $\vdash_{wf}$ *c*[*z*::=*V-var x*]$_{cv}$ $\wedge$ *wfB P* $\mathcal{B}$ *b* **using** *assms*
using *wfG-elim2* **by** *auto*
moreover have *atom x* $\sharp$ (*P* ,$\mathcal{B}$,*G*) **using** *wfG-elims assms wfG-fresh-x* **by** *metis*
ultimately have *wfT P* $\mathcal{B}$ *G* $\{\!|$ *x* : *b* | *c*[*z*::=*V-var x*]$_{cv}$ $|\!\}$ **using** *wfTI assms* **by** *metis*
moreover have $\{\!|$ *x* : *b* | *c*[*z*::=*V-var x*]$_{cv}$ $|\!\}$ = $\{\!|$ *z* : *b* | *c* $|\!\}$ **using** *type-eq-subst* ‹*atom x* $\sharp$ *c*› **by** *auto*
ultimately show *?thesis* **by** *auto*
qed

lemma *wfT-wfT-if*:
assumes *wfT* Θ $\mathcal{B}$ Γ ($\{\!|$ *z2* : *b* | *CE-val v* == *CE-val* (*V-lit L-false*) *IMP* *c*[*z*::=*V-var z2*]$_{cv}$ $|\!\}$)

and *atom z2* $\sharp$ (*c*,Γ)
shows *wfT* Θ $\mathcal{B}$ Γ $\{\!|$ *z* : *b* | *c* $|\!\}$
proof −
have ∗: *atom z2* $\sharp$ (Θ, $\mathcal{B}$, Γ) **using** *wfG-fresh-x wfX-wfY assms fresh-Pair* **by** *metis*
have *wfB* Θ $\mathcal{B}$ *b* **using** *assms wfT-elims* **by** *metis*
have Θ ; $\mathcal{B}$; (*GCons* (*z2*,*b*,*TRUE*) Γ) $\vdash_{wf}$ (*CE-val v* == *CE-val* (*V-lit L-false*) *IMP* *c*[*z*::=*V-var z2*]$_{cv}$) **using** *wfT-wfC assms fresh-Pair* **by** *auto*
hence Θ ; $\mathcal{B}$; ((*z2*,*b*,*TRUE*) #$_\Gamma\Gamma$) $\vdash_{wf}$ *c*[*z*::=*V-var z2*]$_{cv}$ **using** *wfC-elims* **by** *metis*
hence *wfT* Θ $\mathcal{B}$ Γ ($\{\!|$ *z2* : *b* | *c*[*z*::=*V-var z2*]$_{cv}$ $|\!\}$) **using** *assms fresh-Pair wfTI* ‹*wfB* Θ $\mathcal{B}$ *b*› ∗ **by** *auto*
moreover have $\{\!|$ *z* : *b* | *c* $|\!\}$ = $\{\!|$ *z2* : *b* | *c*[*z*::=*V-var z2*]$_{cv}$ $|\!\}$ **using** *type-eq-subst assms fresh-Pair* **by** *auto*
ultimately show *?thesis* **using** *wfTI assms* **by** *argo*
qed

lemma *wfT-fresh-c*:
fixes *x*::*x*
assumes *wfT P* $\mathcal{B}$ Γ $\{\!|$ *z* : *b* | *c* $|\!\}$ **and** *atom x* $\sharp$ Γ **and** *x* ≠ *z*
shows *atom x* $\sharp$ *c*
proof(*rule ccontr*)
assume ¬ *atom x* $\sharp$ *c*
hence ∗:*atom x* ∈ *supp c* **using** *fresh-def* **by** *auto*
moreover have *supp c* − *set* [*atom z*] ∪ *supp b* ⊆ *atom-dom* Γ ∪ *supp* $\mathcal{B}$
using *assms wfT-supp* τ*.supp* **by** *blast*
moreover hence *atom x* ∈ *supp c* − *set* [*atom z*] **using** *assms* ∗ **by** *auto*
ultimately have *atom x* ∈ *atom-dom* Γ **using** *x-not-in-b-set* **by** *auto*
thus *False* **using** *assms wfG-atoms-supp-eq wfT-wf fresh-def* **by** *metis*
qed

lemma *wfG-x-fresh* [*simp*]:
fixes *x*::*x*
assumes *wfG P* $\mathcal{B}$ *G*
shows *atom x* ∉ *atom-dom G* ⟷ *atom x* $\sharp$ *G*
using *wfG-atoms-supp-eq assms fresh-def* **by** *metis*

lemma *wfD-x-fresh*:
fixes *x*::*x*
assumes *atom x* $\sharp$ Γ **and** *wfD P B* Γ Δ
shows *atom x* $\sharp$ Δ
using *assms* **proof**(*induct* Δ *rule*: Δ*-induct*)
case *DNil*
then show *?case* **using** *supp-DNil fresh-def* **by** *auto*
next
case (*DCons u' t'* Δ')
have *wfg*: *wfG P B* Γ **using** *wfD-wf DCons* **by** *blast*
hence *wfd*: *wfD P B* Γ Δ' **using** *wfD-elims DCons* **by** *blast*
have *supp t'* ⊆ *atom-dom* Γ ∪ *supp B* **using** *wfT-supp DCons wfD-elims* **by** *metis*
moreover have *atom x* ∉ *atom-dom* Γ **using** *DCons*(*2*) *fresh-def wfG-supp wfg* **by** *blast*
ultimately have *atom x* $\sharp$ *t'* **using** *fresh-def DCons wfG-supp wfg x-not-in-b-set* **by** *blast*
moreover have *atom x* $\sharp$ *u'* **using** *supp-at-base fresh-def* **by** *fastforce*
ultimately have *atom x* $\sharp$ (*u'*,*t'*) **using** *supp-Pair* **by** *fastforce*

thus *?case* **using** *DCons fresh-DCons wfd* **by** *fast*
qed

thm *wf-supp2*

lemma *wfG-fresh-x2*:
fixes Γ::Γ **and** z::x **and** Δ::Δ **and** Φ::Φ
assumes Θ ; $\mathcal{B}$; $\Gamma \vdash_{wf} \Delta$ **and** $\Theta \vdash_{wf} \Phi$ **and** *atom* $z \sharp \Gamma$
shows *atom* $z \sharp (\Theta,\Phi,\mathcal{B}, \Gamma,\Delta)$
unfolding *fresh-prodN* **apply**(*intro conjI*)
using *wfG-fresh-x assms fresh-prod3 wfX-wfY* **apply** *metis*
using *wf-supp2*(5) *assms fresh-def* **apply** *blast*
using *assms wfG-fresh-x wfX-wfY fresh-prod3* **apply** *metis*
using *assms wfG-fresh-x wfX-wfY fresh-prod3* **apply** *metis*
using *wf-supp2*(6) *assms fresh-def wfD-x-fresh* **by** *metis*

lemma *wfV-x-fresh*:
fixes v::v **and** b::b **and** Γ::Γ **and** x::x
assumes Θ ; $\mathcal{B}$; $\Gamma \vdash_{wf} v : b$ **and** *atom* $x \sharp \Gamma$
shows *atom* $x \sharp v$
proof −
have *supp* $v \subseteq$ *atom-dom* $\Gamma \cup$ *supp* $\mathcal{B}$ **using** *assms wfV-supp* **by** *auto*
moreover have *atom* $x \notin$ *atom-dom* Γ **using** *fresh-def assms*
dom.simps subsetCE wfG-elims wfG-supp **by** (*metis dom-supp-g*)
moreover have *atom* $x \notin$ *supp* $\mathcal{B}$ **using** *x-not-in-b-set* **by** *auto*
ultimately show *?thesis* **using** *fresh-def* **by** *fast*
qed

lemma *wfE-x-fresh*:
fixes e::e **and** b::b **and** Γ::Γ **and** Δ::Δ **and** Φ::Φ **and** x::x
assumes Θ ; Φ ; $\mathcal{B}$; Γ ; $\Delta \vdash_{wf} e : b$ **and** *atom* $x \sharp \Gamma$
shows *atom* $x \sharp e$
proof −
have *wfG* Θ $\mathcal{B}$ Γ **using** *assms wfE-wf* **by** *auto*
hence *supp* $e \subseteq$ *atom-dom* $\Gamma \cup$ *supp* $\mathcal{B} \cup$ *atom`fst`setD* Δ **using** *wfE-supp dom.simps assms* **by** *auto*
moreover have *atom* $x \notin$ *atom-dom* Γ **using** *fresh-def assms*
dom.simps subsetCE ‹*wfG* Θ $\mathcal{B}$ Γ› *wfG-supp* **by** (*metis dom-supp-g*)
moreover have *atom* $x \notin$ *atom`fst`setD* Δ **by** *auto*
ultimately show *?thesis* **using** *fresh-def x-not-in-b-set* **by** *fast*
qed

lemma *wfT-x-fresh*:
fixes τ::τ **and** Γ::Γ **and** x::x
assumes Θ ; $\mathcal{B}$; $\Gamma \vdash_{wf} \tau$ **and** *atom* $x \sharp \Gamma$
shows *atom* $x \sharp \tau$
proof −
have *wfG* Θ $\mathcal{B}$ Γ **using** *assms wfX-wfY* **by** *auto*
hence *supp* $\tau \subseteq$ *atom-dom* $\Gamma \cup$ *supp* $\mathcal{B}$ **using** *wfT-supp dom.simps assms* **by** *auto*
moreover have *atom* $x \notin$ *atom-dom* Γ **using** *fresh-def assms*
dom.simps subsetCE ‹*wfG* Θ $\mathcal{B}$ Γ› *wfG-supp* **by** (*metis dom-supp-g*)

moreover have *atom x* $\sharp$ *supp* $\mathcal{B}$ **using** *x-not-in-b-set* **by** *simp*
ultimately show *?thesis* **using** *fresh-def* **by** *fast*
qed

lemma *wfS-x-fresh*:
fixes *s*::*s* **and** Δ::Δ **and** *x*::*x*
assumes Θ ; Φ ; $\mathcal{B}$; Γ ; Δ $\vdash_{wf}$ *s* : *b* **and** *atom x* $\sharp$ Γ
shows *atom x* $\sharp$ *s*
proof −
have *supp s* $\subseteq$ *atom-dom* Γ $\cup$ *atom ' fst ' setD* Δ $\cup$ *supp* $\mathcal{B}$ **using** *wf-supp assms* **by** *metis*
moreover have *atom x* $\notin$ *atom ' fst ' setD* Δ **by** *auto*
moreover have *atom x* $\notin$ *atom-dom* Γ **using** *assms fresh-def wfG-dom-supp wfX-wfY* **by** *metis*
moreover have *atom x* $\notin$ *supp* $\mathcal{B}$ **using** *supp-b-empty supp-fset*
by (*simp add*: *x-not-in-b-set*)
ultimately show *?thesis* **using** *fresh-def* **by** *fast*
qed

lemma *wfTh-fresh*:
fixes *x*
assumes *wfTh T*
shows *atom x* $\sharp$ *T*
using *wf-supp1 assms fresh-def* **by** *fastforce*

lemmas *wfTh-x-fresh* = *wfTh-fresh*

lemma *wfPhi-fresh*:
fixes *x*
assumes *wfPhi T P*
shows *atom x* $\sharp$ *P*
using *wf-supp assms fresh-def* **by** *fastforce*

lemmas *wfPhi-x-fresh* = *wfPhi-fresh*
lemmas *wb-x-fresh* = *wfTh-x-fresh wfPhi-x-fresh wfD-x-fresh wfT-x-fresh wfV-x-fresh*

lemma *wfG-inside-fresh*[*ms-fresh*]:
fixes Γ::Γ **and** *x*::*x*
assumes *wfG P* $\mathcal{B}$ (Γ'@((*x*,*b*,*c*) $\#_\Gamma\Gamma$))
shows *atom x* $\notin$ *atom-dom* Γ'
using *assms* **proof**(*induct* Γ' *rule*: Γ-*induct*)
case *GNil*
then show *?case* **by** *auto*
next
case (*GCons x1 b1 c1* Γ*1*)
moreover hence *atom x* $\notin$ *atom ' fst '*({(*x1*,*b1*,*c1*)}) **proof** −
have ∗: *P* ; $\mathcal{B}$ $\vdash_{wf}$ (Γ*1* @ (*x*, *b*, *c*) $\#_\Gamma$ Γ) **using** *wfG-elims append-g.simps GCons* **by** *metis*
have *atom x1* $\sharp$ (Γ*1* @ (*x*, *b*, *c*) $\#_\Gamma$ Γ) **using** *GCons wfG-elims append-g.simps* **by** *metis*
hence *atom x1* $\notin$ *atom-dom* (Γ*1* @ (*x*, *b*, *c*) $\#_\Gamma$ Γ) **using** *wfG-dom-supp fresh-def* ∗ **by** *metis*
thus *?thesis* **by** *auto*
qed
ultimately show *?case* **using** *append-g.simps atom-dom.simps setG.simps wfG-elims*
by (*metis image-insert insert-iff insert-is-Un*)
qed

lemma *wfG-inside-x-in-atom-dom*:
fixes $c::c$ **and** $x::x$ **and** $\Gamma::\Gamma$
shows *atom* $x \in$ *atom-dom* (Γ'@ $(x,\ b,\ c[z::=V\text{-}var\ x]_{cv})$ $\#_\Gamma\ \Gamma$)
by(*induct* Γ' *rule*: Γ-*induct*, (*simp add*: *setG.simps atom-dom.simps*)+)

lemma *wfG-inside-x-neq*:
fixes $c::c$ **and** $x::x$ **and** $\Gamma::\Gamma$ **and** $G::\Gamma$ **and** $xa::x$
assumes G=(Γ'@ $(x,\ b,\ c[z::=V\text{-}var\ x]_{cv})$ $\#_\Gamma\ \Gamma$) **and** *atom* $xa \sharp G$ **and** $\Theta\ ;\ \mathcal{B} \vdash_{wf} G$
shows $xa \neq x$
proof −
have *atom* $xa \notin$ *atom-dom* G **using** *fresh-def wfG-atoms-supp-eq assms* **by** *metis*
moreover have *atom* $x \in$ *atom-dom* G **using** *wfG-inside-x-in-atom-dom assms* **by** *simp*
ultimately show *?thesis* **by** *auto*
qed

lemma *wfG-inside-x-fresh*:
fixes $c::c$ **and** $x::x$ **and** $\Gamma::\Gamma$ **and** $G::\Gamma$ **and** $xa::x$
assumes G=(Γ'@ $(x,\ b,\ c[z::=V\text{-}var\ x]_{cv})$ $\#_\Gamma\ \Gamma$) **and** *atom* $xa \sharp G$ **and** $\Theta\ ;\ \mathcal{B} \vdash_{wf} G$
shows *atom* $xa \sharp x$
using *fresh-def supp-at-base wfG-inside-x-neq assms* **by** *auto*

lemma *wfT-nil-supp*:
fixes $t::\tau$
assumes $\Theta\ ;\ \{||\}\ ;\ GNil \vdash_{wf} t$
shows *supp* $t = \{\}$
using *wfT-supp atom-dom.simps assms setG.simps* **by** *force*

8.8 Misc

lemma *wfG-cons-append*:
fixes $b'::b$
assumes $\Theta\ ;\ \mathcal{B} \vdash_{wf} ((x',\ b',\ c')\ \ \#_\Gamma\ \Gamma')\ @\ (x,\ b,\ c)\ \ \#_\Gamma\ \Gamma$
shows $\Theta\ ;\ \mathcal{B} \vdash_{wf} (\Gamma'\ @\ (x,\ b,\ c)\ \ \#_\Gamma\ \Gamma)\ \wedge$ *atom* $x' \sharp (\Gamma'\ @\ (x,\ b,\ c)\ \ \#_\Gamma\ \Gamma) \wedge \Theta\ ;\ \mathcal{B} \vdash_{wf} b' \wedge x' \neq x$
proof −
have $((x',\ b',\ c')\ \ \#_\Gamma\ \Gamma')\ @\ (x,\ b,\ c)\ \ \#_\Gamma\ \Gamma = (x',\ b',\ c')\ \ \#_\Gamma\ (\Gamma'\ @\ (x,\ b,\ c)\ \ \#_\Gamma\ \Gamma)$ **using** *append-g.simps* **by** *auto*
hence $*$:$\Theta\ ;\ \mathcal{B} \vdash_{wf} (\Gamma'\ @\ (x,\ b,\ c)\ \ \#_\Gamma\ \Gamma)\ \wedge$ *atom* $x' \sharp (\Gamma'\ @\ (x,\ b,\ c)\ \ \#_\Gamma\ \Gamma) \wedge \Theta\ ;\ \mathcal{B} \vdash_{wf} b'$ **using** *assms wfG-cons* **by** *metis*
moreover have *atom* $x' \sharp x$ **proof**(*rule wfG-inside-x-fresh*[*of* $(\Gamma'\ @\ (x,\ b,\ c)\ \ \#_\Gamma\ \Gamma)$])
show $\Gamma'\ @\ (x,\ b,\ c)\ \ \#_\Gamma\ \Gamma = \Gamma'\ @\ (x,\ b,\ c[x::=V\text{-}var\ x]_{cv})\ \ \#_\Gamma\ \Gamma$ **by** *simp*
show *atom* $x' \sharp \Gamma'\ @\ (x,\ b,\ c)\ \ \#_\Gamma\ \Gamma$ **using** $*$ **by** *auto*
show $\Theta\ ;\ \mathcal{B}\ \vdash_{wf} \Gamma'\ @\ (x,\ b,\ c)\ \ \#_\Gamma\ \Gamma$ **using** $*$ **by** *auto*
qed
ultimately show *?thesis* **by** *auto*
qed

lemma *flip-u-eq*:
fixes $u::u$ **and** $u'::u$ **and** $\Theta::\Theta$ **and** $\tau::\tau$

assumes Θ ; $\mathcal{B}$; $\Gamma \vdash_{wf} \tau$
shows $(u \leftrightarrow u') \cdot \tau = \tau$ **and** $(u \leftrightarrow u') \cdot \Gamma = \Gamma$ **and** $(u \leftrightarrow u') \cdot \Theta = \Theta$ **and** $(u \leftrightarrow u') \cdot \mathcal{B} = \mathcal{B}$
proof −
show $(u \leftrightarrow u') \cdot \tau = \tau$ **using** *wfT-supp flip-fresh-fresh*
by (*metis assms(1) fresh-def u-not-in-t*)
show $(u \leftrightarrow u') \cdot \Gamma = \Gamma$ **using** *u-not-in-g wfX-wfY assms flip-fresh-fresh fresh-def* **by** *metis*
show $(u \leftrightarrow u') \cdot \Theta = \Theta$ **using** *theta-flip-eq assms wfX-wfY* **by** *metis*
show $(u \leftrightarrow u') \cdot \mathcal{B} = \mathcal{B}$ **using** *u-not-in-b-set flip-fresh-fresh fresh-def* **by** *metis*
qed

lemma *wfT-wf-cons-flip*:
fixes *c*::*c* **and** *x*::*x*
assumes *wfT P* $\mathcal{B}$ Γ $\{\!|\ z : b\ |\ c\ |\!\}$ **and** *atom x* $\sharp$ (*c*,Γ)
shows *wfG P* $\mathcal{B}$ ((*x*,*b*,*c*[*z*::=*V-var x*]$_{cv}$) $\#_\Gamma\Gamma$)
proof −
have $\{\!|\ x : b \mid c[z::=V\text{-}var\ x]_{cv}\ |\!\} = \{\!|\ z : b\ |\ c\ |\!\}$ **using** *assms freshers type-eq-subst* **by** *metis*
hence ∗:*wfT P* $\mathcal{B}$ Γ $\{\!|\ x : b \mid c[z::=V\text{-}var\ x]_{cv}\ |\!\}$ **using** *assms* **by** *metis*
show *?thesis* **proof**(*rule wfG-consI*)
show ‹ *P* ; $\mathcal{B}$ $\vdash_{wf}$ Γ › **using** *assms wfT-wf* **by** *auto*
show ‹*atom x* $\sharp$ Γ› **using** *assms* **by** *auto*
show ‹ *P* ; $\mathcal{B}$ $\vdash_{wf}$ *b* › **using** *assms wfX-wfY b-of.simps* **by** *metis*
show ‹ *P* ; $\mathcal{B}$; (*x*, *b*, *TRUE*) $\#_\Gamma$ Γ $\vdash_{wf}$ *c*[*z*::=*V-var x*]$_{cv}$ › **using** *wfT-wfC* ∗ *assms fresh-Pair*
by *metis*
qed
qed

8.9 Context Strengthening

Can remove an entry for a variable from the context if the variable doesn't appear in the term and the variable is not used later in the context or any other context

lemma *fresh-restrict*:
fixes *y*::′*a*::*at-base* **and** Γ::Γ
assumes *atom y* $\sharp$ (Γ' @ (*x*, *b*, *c*) $\#_\Gamma$ Γ)
shows *atom y* $\sharp$ (Γ'@Γ)
using *assms* **proof**(*induct* Γ' *rule*: Γ-*induct*)
case *GNil*
then show *?case* **using** *fresh-GCons fresh-GNil* **by** *auto*
next
case (*GCons x*′ *b*′ *c*′ Γ'')
then show *?case* **using** *fresh-GCons fresh-GNil* **by** *auto*
qed

lemma *wf-restrict1*:
fixes Γ::Γ **and** Γ'::Γ **and** *v*::*v* **and** *e*::*e* **and** *c*::*c* **and** τ::τ **and** *ts*::(*string*∗τ) *list* **and** Δ::Δ **and** *s*::*s* **and** *b*::*b* **and** *ftq*::*fun-typ-q* **and** *ft*::*fun-typ* **and** *ce*::*ce* **and** *td*::*type-def*
and *cs*::*branch-s* **and** *css*::*branch-list*
shows Θ ; $\mathcal{B}$; Γ $\vdash_{wf}$ *v* : *b* $\Longrightarrow$ Γ=Γ_1@((*x*,*b*′,*c*′) $\#_\Gamma\Gamma_2$) $\Longrightarrow$ *atom x* $\sharp$ *v* $\Longrightarrow$ *atom x* $\sharp$ Γ_1 $\Longrightarrow$ Θ ; $\mathcal{B}$; Γ_1@Γ_2 $\vdash_{wf}$ *v* : *b* **and**

Θ ; $\mathcal{B}$; Γ $\vdash_{wf}$ *c* $\Longrightarrow$ Γ=Γ_1@((*x*,*b*′,*c*′) $\#_\Gamma\Gamma_2$) $\Longrightarrow$ *atom x* $\sharp$ *c*$\Longrightarrow$ *atom x* $\sharp$ Γ_1 $\Longrightarrow$ Θ ; $\mathcal{B}$; Γ_1@Γ_2 $\vdash_{wf}$ *c* **and**

Θ ; $\mathcal{B} \vdash_{wf} \Gamma \Longrightarrow \Gamma{=}\Gamma_1@((x,b',c') \ \#_\Gamma \Gamma_2) \Longrightarrow \ atom\ x \sharp \Gamma_1 \Longrightarrow \Theta$; $\mathcal{B} \vdash_{wf} \Gamma_1@\Gamma_2$ **and**

Θ ; $\mathcal{B}$; $\Gamma \ \vdash_{wf} \tau \Longrightarrow \Gamma{=}\Gamma_1@((x,b',c') \ \#_\Gamma \Gamma_2) \Longrightarrow atom\ x \sharp \tau \Longrightarrow atom\ x \sharp \Gamma_1 \Longrightarrow \Theta$; $\mathcal{B}$; $\Gamma_1@\Gamma_2 \vdash_{wf} \tau$ **and**

Θ ; $\mathcal{B}$; $\Gamma \ \vdash_{wf} ts \Longrightarrow True$ **and**

$\vdash_{wf} \Theta \Longrightarrow True$ **and**

Θ ; $\mathcal{B} \vdash_{wf} b \Longrightarrow True$ **and**

Θ ; $\mathcal{B}$; $\Gamma \ \vdash_{wf} ce : b \Longrightarrow \Gamma{=}\Gamma_1@((x,b',c') \ \#_\Gamma \Gamma_2) \Longrightarrow atom\ x \sharp ce \Longrightarrow atom\ x \sharp \Gamma_1 \Longrightarrow \Theta$; $\mathcal{B}$; $\Gamma_1@\Gamma_2 \vdash_{wf} ce : b$ **and**

$\Theta \ \vdash_{wf} td \Longrightarrow True$

proof(*induct arbitrary*: Γ_1 **and** Γ_1 **and** Γ_1 **and** Γ_1 **and** Γ_1 **and** Γ_1 **and** Γ_1 **and** Γ_1 **and** Γ_1 **and** Γ_1 **and** Γ_1 **and** Γ_1 **and** Γ_1 **and** Γ_1 *rule*:*wfV-wfC-wfG-wfT-wfTs-wfTh-wfB-wfCE-wfTD.inducts*)

case (*wfV-varI* Θ $\mathcal{B}$ Γ *b c y*)
hence $y{\neq}x$ **using** *v.fresh* **by** *auto*
hence *Some* (*b*, *c*) = *lookup* ($\Gamma_1@\Gamma_2$) *y* **using** *lookup-restrict wfV-varI* **by** *metis*
then show *?case* **using** *wfV-varI wf-intros* **by** *metis*
next
case (*wfV-litI* Θ Γ *l*)
then show *?case* **using** *e.fresh wf-intros* **by** *metis*
next
case (*wfV-pairI* Θ $\mathcal{B}$ Γ *v1 b1 v2 b2*)
show *?case* **proof**
show Θ ; $\mathcal{B}$; Γ_1 @ $\Gamma_2 \vdash_{wf} v1 : b1$ **using** *wfV-pairI* **by** *auto*
show Θ ; $\mathcal{B}$; Γ_1 @ $\Gamma_2 \vdash_{wf} v2 : b2$ **using** *wfV-pairI* **by** *auto*
qed
next
case (*wfV-consI s dclist* Θ *dc x b c* $\mathcal{B}$ Γ *v*)
show *?case* **proof**
show *AF-typedef s dclist* $\in$ *set* Θ **using** *wfV-consI* **by** *auto*
show (*dc*, $\{\!|\ x : b \mid c\ |\!\}$) $\in$ *set dclist* **using** *wfV-consI* **by** *auto*
show Θ ; $\mathcal{B}$; Γ_1 @ $\Gamma_2 \vdash_{wf} v : b$ **using** *wfV-consI* **by** *auto*
qed
next
case (*wfV-conspI s bv dclist* Θ *dc x b' c* $\mathcal{B}$ *b* Γ *v*)
show *?case* **proof**
show *AF-typedef-poly s bv dclist* $\in$ *set* Θ **using** *wfV-conspI* **by** *auto*
show (*dc*, $\{\!|\ x : b' \mid c\ |\!\}$) $\in$ *set dclist* **using** *wfV-conspI* **by** *auto*
show Θ ; $\mathcal{B} \ \vdash_{wf} b$ **using** *wfV-conspI* **by** *auto*
show Θ ; $\mathcal{B}$; Γ_1 @ $\Gamma_2 \vdash_{wf} v : b'[bv::=b]_{bb}$ **using** *wfV-conspI* **by** *auto*
show *atom bv* $\sharp$ (Θ, $\mathcal{B}$, Γ_1 @ Γ_2, *b*, *v*) **unfolding** *fresh-prodN fresh-append-g* **using** *wfV-conspI fresh-prodN fresh-GCons fresh-append-g* **by** *metis*
qed
next
case (*wfCE-valI* Θ $\mathcal{B}$ Γ *v b*)
then show *?case* **using** *ce.fresh wf-intros* **by** *metis*
next
case (*wfCE-plusI* Θ $\mathcal{B}$ Γ *v1 v2*)
then show *?case* **using** *ce.fresh wf-intros* **by** *metis*
next

case (*wfCE-leqI* Θ $\mathcal{B}$ Γ *v1 v2*)
then show *?case* **using** *ce.fresh wf-intros* **by** *metis*
next
case (*wfCE-fstI* Θ $\mathcal{B}$ Γ *v1 b1 b2*)
then show *?case* **using** *ce.fresh wf-intros* **by** *metis*
next
case (*wfCE-sndI* Θ $\mathcal{B}$ Γ *v1 b1 b2*)
then show *?case* **using** *ce.fresh wf-intros* **by** *metis*
next
case (*wfCE-concatI* Θ $\mathcal{B}$ Γ *v1 v2*)
then show *?case* **using** *ce.fresh wf-intros* **by** *metis*
next
case (*wfCE-lenI* Θ $\mathcal{B}$ Γ *v1*)
then show *?case* **using** *ce.fresh wf-intros* **by** *metis*
next
case (*wfTI z* Θ $\mathcal{B}$ Γ *b c*)
hence $x \neq z$ **using** *wfTI*
fresh-GCons fresh-prodN fresh-PairD(1) fresh-gamma-append not-self-fresh **by** *metis*
show *?case* **proof**
show ‹*atom z* $\sharp$ (Θ, $\mathcal{B}$, Γ_1 @ Γ_2)› **using** *wfTI fresh-restrict*[*of z*] **using** *wfG-fresh-x wfX-wfY wfTI fresh-prodN* **by** *metis*
show ‹ Θ ; $\mathcal{B}$ $\vdash_{wf}$ *b* › **using** *wfTI* **by** *auto*
have Θ ; $\mathcal{B}$; (*z*, *b*, *TRUE*) $\#_\Gamma$ Γ_1) @ Γ_2 $\vdash_{wf}$ *c* **proof**(*rule wfTI*(5)[*of* (*z*, *b*, *TRUE*) $\#_\Gamma$ Γ_1])
show ‹(*z*, *b*, *TRUE*) $\#_\Gamma$ Γ = ((*z*, *b*, *TRUE*) $\#_\Gamma$ Γ_1) @ (*x*, *b'*, *c'*) $\#_\Gamma$ Γ_2› **using** *wfTI* **by** *auto*
show ‹*atom x* $\sharp$ *c*› **using** *wfTI* τ.*fresh* ‹$x \neq z$› **by** *auto*
show ‹*atom x* $\sharp$ (*z*, *b*, *TRUE*) $\#_\Gamma$ Γ_1› **using** *wfTI* ‹$x \neq z$› *fresh-GCons* **by** *simp*
qed
thus ‹ Θ ; $\mathcal{B}$; (*z*, *b*, *TRUE*) $\#_\Gamma$ Γ_1 @ Γ_2 $\vdash_{wf}$ *c* › **by** *auto*
qed
next
case (*wfC-eqI* Θ $\mathcal{B}$ Γ *e1 b e2*)
show *?case* **proof**
show Θ ; $\mathcal{B}$; Γ_1 @ Γ_2 $\vdash_{wf}$ *e1* : *b* **using** *wfC-eqI c.fresh fresh-Nil* **by** *auto*
show Θ ; $\mathcal{B}$; Γ_1 @ Γ_2 $\vdash_{wf}$ *e2* : *b* **using** *wfC-eqI c.fresh fresh-Nil* **by** *auto*
qed
next
case (*wfC-trueI* Θ Γ)
then show *?case* **using** *c.fresh wf-intros* **by** *metis*
next
case (*wfC-falseI* Θ Γ)
then show *?case* **using** *c.fresh wf-intros* **by** *metis*
next
case (*wfC-conjI* Θ Γ *c1 c2*)
then show *?case* **using** *c.fresh wf-intros* **by** *metis*
next
case (*wfC-disjI* Θ Γ *c1 c2*)
then show *?case* **using** *c.fresh wf-intros* **by** *metis*
next
case (*wfC-notI* Θ Γ *c1*)
then show *?case* **using** *c.fresh wf-intros* **by** *metis*
next

case (*wfC-impI* Θ Γ *c1 c2*)
then show *?case* **using** *c.fresh wf-intros* **by** *metis*
next
case (*wfG-nilI* Θ)
then show *?case* **using** *wfV-varI wf-intros*
by (*meson GNil-append* Γ.*simps*(*3*))
next
case (*wfG-cons1I c1* Θ $\mathcal{B}$ *G x1 b1*)
show *?case* **proof**(*cases* Γ_1=*GNil*)
case *True*
then show *?thesis* **using** *wfG-cons1I wfG-consI* **by** *auto*
next
case *False*
then obtain G'::Γ **where** $*$:(*x1*, *b1*, *c1*) $\#_\Gamma$ $G' = \Gamma_1$ **using** *GCons-eq-append-conv wfG-cons1I*
by *auto*
hence $**$:G=G' @ (*x*, *b'*, *c'*) $\#_\Gamma$ Γ_2 **using** *wfG-cons1I* **by** *auto*

have Θ ; $\mathcal{B} \vdash_{wf}$ (*x1*, *b1*, *c1*) $\#_\Gamma$ (G' @ Γ_2) **proof**(*rule Wellformed.wfG-cons1I*)
show ‹*c1* $\notin$ {*TRUE*, *FALSE*}› **using** *wfG-cons1I* **by** *auto*
show ‹*atom x1* $\sharp$ G' @ Γ_2› **using** *wfG-cons1I*(*4*) $**$ *fresh-restrict* **by** *metis*
have *atom x* $\sharp$ G' **using** *wfG-cons1I* $*$ **using** *fresh-GCons* **by** *blast*
thus ‹ Θ ; $\mathcal{B} \vdash_{wf}$ G' @ Γ_2 › **using** *wfG-cons1I*(*3*)[*of G'*] $**$ **by** *auto*
have *atom x* $\sharp$ *c1* $\wedge$ *atom x* $\sharp$ (*x1*, *b1*, *TRUE*) $\#_\Gamma$ G' **using** *fresh-GCons* ‹*atom x* $\sharp$ Γ_1› $*$ **by** *auto*
thus ‹ Θ ; $\mathcal{B}$; (*x1*, *b1*, *TRUE*) $\#_\Gamma$ G' @ Γ_2 $\vdash_{wf}$ *c1* › **using** *wfG-cons1I*(*6*)[*of* (*x1*, *b1*, *TRUE*)
$\#_\Gamma$ G'] $**$ $*$ *wfG-cons1I* **by** *auto*
show ‹ Θ ; $\mathcal{B} \vdash_{wf}$ *b1* › **using** *wfG-cons1I* **by** *auto*
qed
thus *?thesis* **using** $*$ **by** *auto*
qed
next
case (*wfG-cons2I c1* Θ $\mathcal{B}$ *G x1 b1*)
show *?case* **proof**(*cases* Γ_1=*GNil*)
case *True*
then show *?thesis* **using** *wfG-cons2I wfG-consI* **by** *auto*
next
case *False*
then obtain G'::Γ **where** $*$:(*x1*, *b1*, *c1*) $\#_\Gamma$ $G' = \Gamma_1$ **using** *GCons-eq-append-conv wfG-cons2I*
by *auto*
hence $**$:G=G' @ (*x*, *b'*, *c'*) $\#_\Gamma$ Γ_2 **using** *wfG-cons2I* **by** *auto*

have Θ ; $\mathcal{B} \vdash_{wf}$ (*x1*, *b1*, *c1*) $\#_\Gamma$ (G' @ Γ_2) **proof**(*rule Wellformed.wfG-cons2I*)
show ‹*c1* $\in$ {*TRUE*, *FALSE*}› **using** *wfG-cons2I* **by** *auto*
show ‹*atom x1* $\sharp$ G' @ Γ_2› **using** *wfG-cons2I* $**$ *fresh-restrict* **by** *metis*
have *atom x* $\sharp$ G' **using** *wfG-cons2I* $*$ **using** *fresh-GCons* **by** *blast*
thus ‹ Θ ; $\mathcal{B} \vdash_{wf}$ G' @ Γ_2 › **using** *wfG-cons2I* $**$ **by** *auto*
show ‹ Θ ; $\mathcal{B}$ $\vdash_{wf}$ *b1* › **using** *wfG-cons2I* **by** *auto*
qed
thus *?thesis* **using** $*$ **by** *auto*
qed
qed(*auto*)+

lemma *wf-restrict2*:

fixes Γ::Γ **and** Γ'::Γ **and** v::v **and** e::e **and** c::c **and** τ::τ **and** ts::(*string*$*\tau$) *list* **and** Δ::Δ **and** s::s **and** b::b **and** *ftq*::*fun-typ-q* **and** *ft*::*fun-typ* **and** *ce*::*ce* **and** *td*::*type-def*
and *cs*::*branch-s* **and** *css*::*branch-list*
shows Θ ; Φ ; $\mathcal{B}$; Γ ; $\Delta \vdash_{wf} e : b \implies \Gamma{=}\Gamma_1@((x,b',c') \#_\Gamma \Gamma_2) \implies$ *atom* $x \sharp e \implies$ *atom* $x \sharp \Gamma_1 \implies$ *atom* $x \sharp \Delta \implies \Theta$; Φ ; $\mathcal{B}$; $\Gamma_1@\Gamma_2$; $\Delta \vdash_{wf} e : b$ **and**
Θ ; Φ ; $\mathcal{B}$; Γ ; $\Delta \vdash_{wf} s : b \implies$ *True* **and**
Θ ; Φ ; $\mathcal{B}$; Γ ; Δ ; *tid* ; *dc* ; $t \vdash_{wf}$ *cs* : $b \implies$ *True* **and**
Θ ; Φ ; $\mathcal{B}$; Γ ; Δ ; *tid* ; *dclist* $\vdash_{wf}$ *css* : $b \implies$ *True* **and**
$\Theta \vdash_{wf} (\Phi{::}\Phi) \implies$ *True* **and**
Θ ; $\mathcal{B}$; $\Gamma \vdash_{wf} \Delta \implies \Gamma{=}\Gamma_1@((x,b',c') \#_\Gamma \Gamma_2) \implies$ *atom* $x \sharp \Gamma_1 \implies$ *atom* $x \sharp \Delta \implies \Theta$; $\mathcal{B}$; $\Gamma_1@\Gamma_2 \vdash_{wf} \Delta$ **and**
Θ ; $\Phi \vdash_{wf}$ *ftq* $\implies$ *True* **and**
Θ ; Φ ; $\mathcal{B} \vdash_{wf}$ *ft* $\implies$ *True*

proof(*induct arbitrary*: Γ_1 **and** Γ_1 **and** Γ_1 **and** Γ_1 **and** Γ_1 **and** Γ_1 **and** Γ_1 **and** Γ_1 **and** Γ_1 **and** Γ_1 **and** Γ_1 **and** Γ_1 **and** Γ_1
rule:*wfE-wfS-wfCS-wfCSS-wfPhi-wfD-wfFTQ-wfFT.inducts*)

case (*wfE-valI* Θ Φ Γ Δ *v b*)
then show *?case* **using** *e.fresh wf-intros wf-restrict1* **by** *metis*
next
case (*wfE-plusI* Θ Φ Γ Δ *v1 v2*)
then show *?case* **using** *e.fresh wf-intros wf-restrict1* **by** *metis*
next
case (*wfE-leqI* Θ Φ Γ Δ *v1 v2*)
then show *?case* **using** *e.fresh wf-intros wf-restrict1* **by** *metis*
next
case (*wfE-fstI* Θ Φ Γ Δ *v1 b1 b2*)
then show *?case* **using** *e.fresh wf-intros wf-restrict1* **by** *metis*
next
case (*wfE-sndI* Θ Φ Γ Δ *v1 b1 b2*)
then show *?case* **using** *e.fresh wf-intros wf-restrict1* **by** *metis*
next
case (*wfE-concatI* Θ Φ Γ Δ *v1 v2*)
then show *?case* **using** *e.fresh wf-intros wf-restrict1* **by** *metis*
next
case (*wfE-splitI* Θ Φ Γ Δ *v1 v2*)
then show *?case* **using** *e.fresh wf-intros wf-restrict1* **by** *metis*
next
case (*wfE-lenI* Θ Φ Γ Δ *v1*)
then show *?case* **using** *e.fresh wf-intros wf-restrict1* **by** *metis*
next
case (*wfE-appI* Θ Φ Γ Δ *f x b c* τ *s'* *v*)
then show *?case* **using** *e.fresh wf-intros wf-restrict1* **by** *metis*
next
case (*wfE-appPI* Θ Φ $\mathcal{B}$ Γ Δ *b'* *bv v* τ *f x b c s*)
show *?case* **proof**
show ‹ $\Theta \vdash_{wf} \Phi$ › **using** *wfE-appPI* **by** *auto*
show ‹ Θ ; $\mathcal{B}$; Γ_1 @ $\Gamma_2 \vdash_{wf} \Delta$ › **using** *wfE-appPI* **by** *auto*
show ‹ Θ ; $\mathcal{B} \vdash_{wf} b'$ › **using** *wfE-appPI* **by** *auto*

have *atom bv* $\sharp$ Γ_1 @ Γ_2 **using** *wfE-appPI fresh-prodN fresh-restrict* **by** *metis*

thus ‹*atom bv* ♯ (Φ, Θ, $\mathcal{B}$, Γ_1 @ Γ_2, Δ, *b′*, *v*, (*b-of* τ)[*bv*::=*b′*]$_b$)›
using *wfE-appPI fresh-prodN* **by** *auto*

show ‹*Some* (*AF-fundef f* (*AF-fun-typ-some bv* (*AF-fun-typ x b c* τ *s*))) = *lookup-fun* Φ *f*› **using** *wfE-appPI* **by** *auto*
show ‹ Θ ; $\mathcal{B}$; Γ_1 @ Γ_2 ⊢$_{wf}$ *v* : *b*[*bv*::=*b′*]$_b$ › **using** *wfE-appPI wf-restrict1* **by** *auto*
qed

next
case (*wfE-mvarI* Θ Φ Γ Δ *u* τ)
then show *?case* **using** *e.fresh wf-intros* **by** *metis*
next

case (*wfD-emptyI* Θ Γ)
then show *?case* **using** *c.fresh wf-intros wf-restrict1* **by** *metis*
next
case (*wfD-cons* Θ $\mathcal{B}$ Γ Δ τ *u*)
show *?case* **proof**
show Θ ; $\mathcal{B}$; Γ_1 @ Γ_2 ⊢$_{wf}$ Δ **using** *wfD-cons fresh-DCons* **by** *metis*
show Θ ; $\mathcal{B}$; Γ_1 @ Γ_2 ⊢$_{wf}$ τ **using** *wfD-cons fresh-DCons fresh-Pair wf-restrict1* **by** *metis*
show *u* ∉ *fst* ‘ *setD* Δ **using** *wfD-cons* **by** *auto*
qed
next
case (*wfFTNone* Θ *ft*)
then show *?case* **by** *auto*
next
case (*wfFTSome* Θ *bv ft*)
then show *?case* **by** *auto*
next
case (*wfFTI* Θ *B b* Φ *x c s* τ)
then show *?case* **by** *auto*
qed(*auto*)+

lemmas *wf-restrict=wf-restrict1 wf-restrict2*

lemma *wfG-intros2*:
assumes *wfC P* $\mathcal{B}$ ((*x*,*b*,*TRUE*) #$_\Gamma$Γ) *c*
shows *wfG P* $\mathcal{B}$ ((*x*,*b*,*c*) #$_\Gamma$Γ)
proof −
have *wfG P* $\mathcal{B}$ ((*x*,*b*,*TRUE*) #$_\Gamma$Γ) **using** *wfC-wf assms* **by** *auto*
hence ∗:*wfG P* $\mathcal{B}$ Γ ∧ *atom x* ♯ Γ ∧ *wfB P* $\mathcal{B}$ *b* **using** *wfG-elims* **by** *metis*
show *?thesis* **using** *assms* **proof**(*cases c* ∈ {*TRUE*,*FALSE*})
case *True*
then show *?thesis* **using** *wfG-cons2I* ∗ **by** *auto*
next
case *False*
then show *?thesis* **using** *wfG-cons1I* ∗ *assms* **by** *auto*
qed
qed

8.10 Type Definitions

lemma *wf-theta-weakening1*:
fixes Γ::Γ **and** Γ'::Γ **and** v::v **and** e::e **and** c::c **and** τ::τ **and** ts::(*string*$*\tau$) *list* **and** Δ::Δ **and** s::s **and** b::b **and** $\mathcal{B}$ :: $\mathcal{B}$ **and** *ftq*::*fun-typ-q* **and** *ft*::*fun-typ* **and** *ce*::*ce* **and** *td*::*type-def* **and** *cs*::*branch-s* **and** *css*::*branch-list* **and** t::τ

shows $\Theta\ ;\ \mathcal{B}\ ;\ \Gamma\ \vdash_{wf} v : b \Longrightarrow \vdash_{wf} \Theta' \Longrightarrow set\ \Theta \subseteq set\ \Theta' \Longrightarrow \Theta'\ ;\ \mathcal{B}\ ;\ \Gamma \vdash_{wf} v : b$ **and**
$\Theta\ ;\ \mathcal{B}\ ;\ \Gamma\ \vdash_{wf} c \Longrightarrow \vdash_{wf} \Theta' \Longrightarrow set\ \Theta \subseteq set\ \Theta' \Longrightarrow \Theta'\ ;\ \mathcal{B}\ ;\ \Gamma \vdash_{wf} c$ **and**
$\Theta\ ;\ \mathcal{B} \vdash_{wf} \Gamma \Longrightarrow \vdash_{wf} \Theta' \Longrightarrow set\ \Theta \subseteq set\ \Theta' \Longrightarrow \Theta'\ ;\ \mathcal{B}\ \vdash_{wf} \Gamma$ **and**
$\Theta\ ;\ \mathcal{B}\ ;\ \Gamma\ \vdash_{wf} \tau \Longrightarrow \vdash_{wf} \Theta' \Longrightarrow set\ \Theta \subseteq set\ \Theta' \Longrightarrow \Theta'\ ;\ \mathcal{B}\ ;\ \Gamma \vdash_{wf} \tau$ **and**
$\Theta\ ;\ \mathcal{B}\ ;\ \Gamma\ \vdash_{wf} ts \Longrightarrow \vdash_{wf} \Theta' \Longrightarrow set\ \Theta \subseteq set\ \Theta' \Longrightarrow \Theta'\ ;\ \mathcal{B}\ ;\ \Gamma\ \vdash_{wf} ts$ **and**
$\vdash_{wf} P \Longrightarrow True$ **and**
$\Theta\ ;\ \mathcal{B} \vdash_{wf} b \Longrightarrow \vdash_{wf} \Theta' \Longrightarrow set\ \Theta \subseteq set\ \Theta' \Longrightarrow \Theta'\ ;\ \mathcal{B}\ \vdash_{wf} b$ **and**
$\Theta\ ;\ \mathcal{B}\ ;\ \Gamma\ \vdash_{wf} ce : b \Longrightarrow \vdash_{wf} \Theta' \Longrightarrow set\ \Theta \subseteq set\ \Theta' \Longrightarrow \Theta'\ ;\ \mathcal{B}\ ;\ \Gamma \vdash_{wf} ce : b$ **and**
$\Theta\ \vdash_{wf} td \Longrightarrow \vdash_{wf} \Theta' \Longrightarrow set\ \Theta \subseteq set\ \Theta' \Longrightarrow \Theta'\ \vdash_{wf} td$
proof(*nominal-induct* b **and** c **and** Γ **and** τ **and** ts **and** P **and** b **and** b **and** td
avoiding: Θ'
rule:*wfV-wfC-wfG-wfT-wfTs-wfTh-wfB-wfCE-wfTD.strong-induct*)
case (*wfV-consI s dclist* Θ *dc x b c* $\mathcal{B}$ Γ *v*)
show *?case* **proof**
show ‹*AF-typedef s dclist* $\in$ *set* Θ'› **using** *wfV-consI* **by** *auto*
show ‹(*dc*, $\{\!|$ *x* : *b* | *c* $|\!\}$) $\in$ *set dclist*› **using** *wfV-consI* **by** *auto*
show ‹ $\Theta'\ ;\ \mathcal{B}\ ;\ \Gamma \vdash_{wf} v : b$ › **using** *wfV-consI* **by** *auto*
qed
next
case (*wfV-conspI s bv dclist* Θ *dc x b' c* $\mathcal{B}$ *b* Γ *v*)
show *?case* **proof**
show ‹*AF-typedef-poly s bv dclist* $\in$ *set* Θ'› **using** *wfV-conspI* **by** *auto*
show ‹(*dc*, $\{\!|$ *x* : *b'* | *c* $|\!\}$) $\in$ *set dclist*› **using** *wfV-conspI* **by** *auto*
show ‹$\Theta'\ ;\ \mathcal{B}\ ;\ \Gamma \vdash_{wf} v : b'[bv::=b]_{bb}$ › **using** *wfV-conspI* **by** *auto*
show $\Theta'\ ;\ \mathcal{B}\ \vdash_{wf} b$ **using** *wfV-conspI* **by** *auto*
show *atom bv* $\sharp$ (Θ', $\mathcal{B}$, Γ, *b*, *v*) **using** *wfV-conspI fresh-prodN* **by** *auto*
qed
next
case (*wfTI z* Θ $\mathcal{B}$ Γ *b c*)
thus *?case* **using** *Wellformed.wfTI* **by** *auto*
next
case (*wfB-consI* Θ *s dclist*)
show *?case* **proof**
show ‹ $\vdash_{wf} \Theta'$ › **using** *wfB-consI* **by** *auto*
show ‹*AF-typedef s dclist* $\in$ *set* Θ'› **using** *wfB-consI* **by** *auto*
qed
next
case (*wfB-appI* Θ $\mathcal{B}$ *b s bv dclist*)
show *?case* **proof**
show ‹ $\vdash_{wf} \Theta'$ › **using** *wfB-appI* **by** *auto*
show ‹*AF-typedef-poly s bv dclist* $\in$ *set* Θ'› **using** *wfB-appI* **by** *auto*
show $\Theta'\ ;\ \mathcal{B}\ \vdash_{wf} b$ **using** *wfB-appI* **by** *simp*
qed
qed(*metis wf-intros*)+

lemma *wf-theta-weakening2*:

fixes Γ::Γ **and** Γ'::Γ **and** v::v **and** e::e **and** c::c **and** τ::τ **and** ts::($string*\tau$) *list* **and** Δ::Δ **and** s::s **and** b::b **and** $\mathcal{B}$ :: $\mathcal{B}$ **and** *ftq*::*fun-typ-q* **and** *ft*::*fun-typ* **and** *ce*::*ce* **and** *td*::*type-def* **and** *cs*::*branch-s* **and** *css*::*branch-list* **and** t::τ

shows
Θ ; Φ ; $\mathcal{B}$; Γ ; $\Delta \vdash_{wf} e : b \Longrightarrow \vdash_{wf} \Theta' \Longrightarrow set\ \Theta \subseteq set\ \Theta' \Longrightarrow \Theta'$; Φ ; $\mathcal{B}$; Γ ; $\Delta \vdash_{wf} e : b$
and
Θ ; Φ ; $\mathcal{B}$; Γ ; $\Delta \vdash_{wf} s : b \Longrightarrow \vdash_{wf} \Theta' \Longrightarrow set\ \Theta \subseteq set\ \Theta' \Longrightarrow \Theta'$; Φ ; $\mathcal{B}$; Γ ; $\Delta \vdash_{wf} s : b$ **and**
Θ ; Φ ; $\mathcal{B}$; Γ ; Δ ; *tid* ; *dc* ; $t \vdash_{wf} cs : b \Longrightarrow \vdash_{wf} \Theta' \Longrightarrow set\ \Theta \subseteq set\ \Theta' \Longrightarrow \Theta'$; Φ ; $\mathcal{B}$; Γ ; Δ ; *tid* ; *dc* ; $t \vdash_{wf} cs : b$ **and**
Θ ; Φ ; $\mathcal{B}$; Γ ; Δ ; *tid* ; *dclist* $\vdash_{wf} css : b \Longrightarrow \vdash_{wf} \Theta' \Longrightarrow set\ \Theta \subseteq set\ \Theta' \Longrightarrow \Theta'$; Φ ; $\mathcal{B}$; Γ ; Δ ; *tid* ; *dclist* $\vdash_{wf} css : b$ **and**
$\Theta \vdash_{wf} (\Phi::\Phi) \Longrightarrow \vdash_{wf} \Theta' \Longrightarrow set\ \Theta \subseteq set\ \Theta' \Longrightarrow \Theta' \vdash_{wf} (\Phi::\Phi)$ **and**
Θ ; $\mathcal{B}$; $\Gamma \vdash_{wf} \Delta \Longrightarrow \vdash_{wf} \Theta' \Longrightarrow set\ \Theta \subseteq set\ \Theta' \Longrightarrow \Theta'$; $\mathcal{B}$; $\Gamma \vdash_{wf} \Delta$ **and**
Θ ; $\Phi \vdash_{wf} ftq \Longrightarrow \vdash_{wf} \Theta' \Longrightarrow set\ \Theta \subseteq set\ \Theta' \Longrightarrow \Theta'$; $\Phi \vdash_{wf} ftq$ **and**
Θ ; Φ ; $\mathcal{B} \vdash_{wf} ft \Longrightarrow \vdash_{wf} \Theta' \Longrightarrow set\ \Theta \subseteq set\ \Theta' \Longrightarrow \Theta'$; Φ ; $\mathcal{B} \vdash_{wf} ft$

proof(*nominal-induct* b **and** b **and** b **and** b **and** Φ **and** Δ **and** *ftq* **and** *ft*
avoiding: Θ'
rule:*wfE-wfS-wfCS-wfCSS-wfPhi-wfD-wfFTQ-wfFT.strong-induct*)
case (*wfE-appPI* Θ Φ $\mathcal{B}$ Γ Δ b' *bv* v τ f x b c s)
show *?case* **proof**
show ‹ $\Theta' \vdash_{wf} \Phi$ › **using** *wfE-appPI* **by** *auto*
show ‹ Θ' ; $\mathcal{B}$; $\Gamma \vdash_{wf} \Delta$ › **using** *wfE-appPI* **by** *auto*
show ‹ Θ' ; $\mathcal{B} \vdash_{wf} b'$ › **using** *wfE-appPI wf-theta-weakening1* **by** *auto*
show ‹*atom bv* ♯ (Φ, Θ', $\mathcal{B}$, Γ, Δ, b', v, (*b-of* τ)[*bv*::=b']$_b$)› **using** *wfE-appPI* **by** *auto*
show ‹*Some* (*AF-fundef* f (*AF-fun-typ-some bv* (*AF-fun-typ* x b c τ s))) = *lookup-fun* Φ f› **using** *wfE-appPI* **by** *auto*
show ‹ Θ' ; $\mathcal{B}$; $\Gamma \vdash_{wf} v : b$[*bv*::=b']$_b$ › **using** *wfE-appPI wf-theta-weakening1* **by** *auto*
qed

next
case (*wfS-matchI* Θ $\mathcal{B}$ Γ v *tid* *dclist* Δ Φ *cs* b)
show *?case* **proof**
show ‹ Θ' ; $\mathcal{B}$; $\Gamma \vdash_{wf} v$: *B-id tid* › **using** *wfS-matchI wf-theta-weakening1* **by** *auto*
show ‹*AF-typedef tid dclist* ∈ *set* Θ'› **using** *wfS-matchI* **by** *auto*
show ‹ Θ' ; $\mathcal{B}$; $\Gamma \vdash_{wf} \Delta$ › **using** *wfS-matchI* **by** *auto*
show ‹ $\Theta' \vdash_{wf} \Phi$ › **using** *wfS-matchI* **by** *auto*
show ‹ Θ' ; Φ ; $\mathcal{B}$; Γ ; Δ ; *tid* ; *dclist* $\vdash_{wf} cs : b$ › **using** *wfS-matchI* **by** *auto*
qed
next
case (*wfS-varI* Θ $\mathcal{B}$ Γ τ v u Φ Δ b s)
show *?case* **proof**
show ‹ Θ' ; $\mathcal{B}$; $\Gamma \vdash_{wf} \tau$ › **using** *wfS-varI wf-theta-weakening1* **by** *auto*
show ‹ Θ' ; $\mathcal{B}$; $\Gamma \vdash_{wf} v$: *b-of* τ › **using** *wfS-varI wf-theta-weakening1* **by** *auto*
show ‹*atom* u ♯ (Φ, Θ', $\mathcal{B}$, Γ, Δ, τ, v, b)› **using** *wfS-varI* **by** *auto*
show ‹ Θ' ; Φ ; $\mathcal{B}$; Γ ; (u, τ) $\#_\Delta$ $\Delta \vdash_{wf} s : b$ › **using** *wfS-varI* **by** *auto*
qed
next
case (*wfS-letI* Θ Φ $\mathcal{B}$ Γ Δ e b' x s b)
show *?case* **proof**
show ‹ Θ' ; Φ ; $\mathcal{B}$; Γ ; $\Delta \vdash_{wf} e : b'$ › **using** *wfS-letI* **by** *auto*

show ‹ Θ' ; Φ ; $\mathcal{B}$; $(x,\ b',\ TRUE)$ $\#_\Gamma$ Γ ; $\Delta \vdash_{wf} s : b$ › **using** *wfS-letI* **by** *auto*
show ‹ Θ' ; $\mathcal{B}$; $\Gamma \vdash_{wf} \Delta$ › **using** *wfS-letI* **by** *auto*
show ‹*atom* $x \sharp (\Phi,\ \Theta',\ \mathcal{B},\ \Gamma,\ \Delta,\ e,\ b)$› **using** *wfS-letI* **by** *auto*
qed
next
case (*wfS-let2I* Θ Φ $\mathcal{B}$ Γ Δ *s1* τ *x s2 b*)
show *?case* **proof**
show ‹ Θ' ; Φ ; $\mathcal{B}$; Γ ; $\Delta \vdash_{wf}$ *s1* : *b-of* τ › **using** *wfS-let2I* **by** *auto*
show ‹ Θ' ; $\mathcal{B}$; $\Gamma \vdash_{wf} \tau$ › **using** *wfS-let2I wf-theta-weakening1* **by** *auto*
show ‹ Θ' ; Φ ; $\mathcal{B}$; $(x,\ b\text{-}of\ \tau,\ TRUE)$ $\#_\Gamma$ Γ ; $\Delta \vdash_{wf}$ *s2* : *b* › **using** *wfS-let2I* **by** *auto*
show ‹*atom* $x \sharp (\Phi,\ \Theta',\ \mathcal{B},\ \Gamma,\ \Delta,$ *s1*, *b*, $\tau)$› **using** *wfS-let2I* **by** *auto*
qed
next
case (*wfS-branchI* Θ Φ $\mathcal{B}$ *x* τ Γ Δ *s b tid dc*)
show *?case* **proof**
show ‹ Θ' ; Φ ; $\mathcal{B}$; $(x,\ b\text{-}of\ \tau,\ TRUE)$ $\#_\Gamma$ Γ ; $\Delta \vdash_{wf} s : b$ › **using** *wfS-branchI* **by** *auto*
show ‹*atom* $x \sharp (\Phi,\ \Theta',\ \mathcal{B},\ \Gamma,\ \Delta,\ \Gamma,\ \tau)$› **using** *wfS-branchI* **by** *auto*
show ‹ Θ' ; $\mathcal{B}$; $\Gamma \vdash_{wf} \Delta$ › **using** *wfS-branchI* **by** *auto*
qed
next
case (*wfPhi-consI f* Φ Θ *ft*)
show *?case* **proof**
show $f \notin$ *name-of-fun* ‘ *set* Φ **using** *wfPhi-consI* **by** *auto*
show Θ' ; $\Phi \vdash_{wf}$ *ft* **using** *wfPhi-consI* **by** *auto*
show $\Theta' \vdash_{wf} \Phi$ **using** *wfPhi-consI* **by** *auto*
qed
next
case (*wfFTNone* Θ *ft*)
then show *?case* **using** *wf-intros* **by** *metis*
next
case (*wfFTSome* Θ *bv ft*)
then show *?case* **using** *wf-intros* **by** *metis*
next
case (*wfFTI* Θ *B b* Φ *x c s* τ)
thus *?case* **using** *Wellformed.wfFTI wf-theta-weakening1* **by** *simp*
next
case (*wfS-assertI* Θ Φ $\mathcal{B}$ *x c* Γ Δ *s b*)
show *?case* **proof**
show ‹ Θ' ; Φ ; $\mathcal{B}$; $(x,\ B\text{-}bool,\ c)$ $\#_\Gamma$ Γ ; $\Delta \vdash_{wf} s : b$ › **using** *wfS-assertI wf-theta-weakening1* **by** *auto*
show ‹ Θ' ; $\mathcal{B}$; $\Gamma \vdash_{wf} c$ › **using** *wfS-assertI wf-theta-weakening1* **by** *auto*
show ‹ Θ' ; $\mathcal{B}$; $\Gamma \vdash_{wf} \Delta$ › **using** *wfS-assertI wf-theta-weakening1* **by** *auto*
have *atom x* $\sharp$ Θ' **using** *wf-supp*(*6*)[*OF* ‹$\vdash_{wf} \Theta'$›] *fresh-def* **by** *auto*
thus ‹*atom* $x \sharp (\Phi,\ \Theta',\ \mathcal{B},\ \Gamma,\ \Delta,\ c,\ b,\ s)$› **using** *wfS-assertI fresh-prodN fresh-def* **by** *simp*
qed
qed(*metis wf-intros wf-theta-weakening1*)+

lemmas *wf-theta-weakening* = *wf-theta-weakening1 wf-theta-weakening2*

lemma *lookup-wfTD*:
fixes *td*::*type-def*
assumes *td* $\in$ *set* Θ **and** $\vdash_{wf} \Theta$

shows $\Theta \vdash_{wf} td$
using *assms* **proof**(*induct* Θ)
case *Nil*
then show *?case* **by** *auto*
next
case (*Cons td′* Θ')
then consider $td = td' \mid td \in set\ \Theta'$ **by** *auto*
then have $\Theta' \vdash_{wf} td$ **proof**(*cases*)
case *1*
then show *?thesis* **using** *Cons* **using** *wfTh-elims* **by** *auto*
next
case *2*
then show *?thesis* **using** *Cons* **using** *wfTh-elims* **by** *auto*
qed
then show *?case* **using** *wf-theta-weakening Cons* **by** (*meson set-subset-Cons*)
qed

8.10.1 Simple

lemma *wfTh-dclist-unique*:
assumes *wfTh* Θ **and** *AF-typedef tid dclist1* $\in$ *set* Θ **and** *AF-typedef tid dclist2* $\in$ *set* Θ
shows *dclist1* = *dclist2*
using *assms* **proof**(*induct* Θ *rule*: Θ-*induct*)
case *TNil*
then show *?case* **by** *auto*
next
case (*AF-typedef tid′ dclist′* Θ')
then show *?case* **using** *wfTh-elims*
by (*metis image-eqI name-of-type.simps*(*1*) *set-ConsD type-def.eq-iff*(*1*))
next
case (*AF-typedef-poly tid bv dclist* Θ')
then show *?case* **using** *wfTh-elims* **by** *auto*
qed

lemma *wfTs-ctor-unique*:
fixes *dclist*::(*string*$*\tau$) *list*
assumes Θ ; {||} ; *GNil* $\vdash_{wf}$ *dclist* **and** (*c*, *t1*) $\in$ *set dclist* **and** (*c*,*t2*) $\in$ *set dclist*
shows *t1* = *t2*
using *assms* **proof**(*induct dclist rule*: *list.inducts*)
case *Nil*
then show *?case* **by** *auto*
next
case (*Cons x1 x2*)
consider $x1 = (c,t1) \mid x1 = (c,t2) \mid x1 \neq (c,t1) \wedge x1 \neq (c,t2)$ **by** *auto*
thus *?case* **proof**(*cases*)
case *1*
then show *?thesis* **using** *Cons wfTs-elims set-ConsD*
by (*metis fst-conv image-eqI prod.inject*)
next
case *2*
then show *?thesis* **using** *Cons wfTs-elims set-ConsD*
by (*metis fst-conv image-eqI prod.inject*)
next

```
    case 3
    then show ?thesis using Cons wfTs-elims by (metis set-ConsD)
  qed
qed

lemma wfTD-ctor-unique:
  assumes  Θ ⊢wf (AF-typedef tid dclist) and (c, t1) ∈ set dclist  and (c,t2) ∈ set dclist
  shows t1 = t2
  using wfTD-elims wfTs-elims assms  wfTs-ctor-unique by metis

lemma wfTh-ctor-unique:
  assumes   wfTh Θ and AF-typedef tid dclist ∈ set Θ and (c, t1) ∈ set dclist  and (c,t2) ∈ set dclist

  shows t1 = t2
  using lookup-wfTD wfTD-ctor-unique assms by metis

lemma wfTs-supp-t:
  fixes dclist::(string*τ) list
  assumes (c,t) ∈ set dclist and Θ ; B ; GNil ⊢wf dclist
  shows supp t ⊆ supp B
using assms proof(induct dclist arbitrary: c t rule:list.induct)
  case Nil
  then show ?case by auto
next
  case (Cons ct dclist')
  then consider ct = (c,t) | (c,t) ∈ set dclist' by auto
  then show ?case proof(cases)
    case 1
    then have Θ ; B ; GNil ⊢wf t using Cons wfTs-elims by blast
    thus ?thesis using wfT-supp atom-dom.simps by force
  next
    case 2
    then show ?thesis using Cons wfTs-elims by metis
  qed
qed

lemma wfTh-lookup-supp-empty:
  fixes t::τ
  assumes  AF-typedef tid dclist ∈ set Θ and (c,t) ∈ set dclist and ⊢wf Θ
  shows supp t = {}
proof −
  have Θ ; {||} ; GNil ⊢wf dclist using assms lookup-wfTD  wfTD-elims by metis
  thus ?thesis using wfTs-supp-t assms by force
qed

lemma wfTh-supp-b:
  assumes  AF-typedef tid dclist ∈ set Θ and (dc,{| z : b | c |} ) ∈ set dclist and ⊢wf Θ
  shows supp b = {}
  using assms wfTh-lookup-supp-empty τ.supp by blast


lemma wfTh-b-eq-iff:
```

fixes *bva1*::*bv* **and** *bva2*::*bv* **and** *dc*::*string*
assumes (*dc*, {| *x1* : *b1* | *c1* |}) ∈ *set dclist1* **and** (*dc*, {| *x2* : *b2* | *c2* |}) ∈ *set dclist2* **and**
wfTs P {|*bva1*|} *GNil dclist1* **and** *wfTs P* {|*bva2*|} *GNil dclist2*
[[*atom bva1*]]*lst.dclist1* = [[*atom bva2*]]*lst.dclist2*
shows [[*atom bva1*]]*lst.* (*dc*,{| *x1* : *b1* | *c1* |}) = [[*atom bva2*]]*lst.* (*dc*,{| *x2* : *b2* | *c2* |})
using *assms* **proof**(*induct dclist1 arbitrary*: *dclist2*)
case *Nil*
then show *?case* **by** *auto*
next
case (*Cons dct1′ dclist1′*)
show *?case* **proof**(*cases dclist2* = [])
case *True*
then show *?thesis* **using** *Cons* **by** *auto*
next
case *False*
then obtain *dct2′* **and** *dclist2′* **where** *cons*:*dct2′* # *dclist2′* = *dclist2* **using** *list.exhaust* **by** *metis*
hence ∗:[[*atom bva1*]]*lst. dclist1′* = [[*atom bva2*]]*lst. dclist2′* ∧ [[*atom bva1*]]*lst. dct1′* = [[*atom bva2*]]*lst. dct2′*
using *Cons lst-head-cons Cons cons* **by** *metis*
hence ∗∗: *fst dct1′* = *fst dct2′* **using** *lst-fst*[*THEN lst-pure*]
by (*metis* (*no-types*) ‹[[*atom bva1*]]*lst. dclist1′* = [[*atom bva2*]]*lst. dclist2′* ∧ [[*atom bva1*]]*lst. dct1′* = [[*atom bva2*]]*lst. dct2′*›
‹⋀*x2 x1 t2′ t2a t2 t1*. [[*atom x1*]]*lst.* (*t1*, *t2a*) = [[*atom x2*]]*lst.* (*t2*, *t2′*) ⟹ *t1* = *t2*› *fst-conv surj-pair*)

show *?thesis* **proof**(*cases fst dct1′* = *dc*)
case *True*
have *dc* ∉ *fst* ‘ *set dclist1′* **using** *wfTs-elims Cons* **by** (*metis True fstI*)
hence *1*:(*dc*, {| *x1* : *b1* | *c1* |}) = *dct1′* **using** *Cons* **by** (*metis fstI image-iff set-ConsD*)
have *dc* ∉ *fst* ‘ *set dclist2′* **using** *wfTs-elims Cons cons*
by (*metis* ∗∗ *True fstI*)
hence *2*:(*dc*, {| *x2* : *b2* | *c2* |}) = *dct2′* **using** *Cons cons* **by** (*metis fst-conv image-eqI set-ConsD*)
then show *?thesis* **using** *Cons* ∗ *1 2* **by** *blast*
next
case *False*
hence *fst dct2′* ≠ *dc* **using** ∗∗ **by** *auto*
hence (*dc*, {| *x1* : *b1* | *c1* |}) ∈ *set dclist1′* ∧ (*dc*, {| *x2* : *b2* | *c2* |}) ∈ *set dclist2′* **using** *Cons cons False*
by (*metis fstI set-ConsD*)
moreover have [[*atom bva1*]]*lst. dclist1′* = [[*atom bva2*]]*lst. dclist2′* **using** ∗ *False* **by** *metis*
ultimately show *?thesis* **using** *Cons* ∗∗ ∗
using *cons wfTs-elims*(*2*) **by** *blast*
qed
qed
qed

8.10.2 Polymorphic

lemma *wfTh-wfTs-poly*:
fixes *dclist*::(*string* ∗ *τ*) *list*
assumes *AF-typedef-poly tyid bva dclist* ∈ *set P* **and** ⊢$_{wf}$ *P*
shows *P* ; {|*bva*|} ; *GNil* ⊢$_{wf}$ *dclist*
proof −

have $*$:$P \vdash_{wf}$ *AF-typedef-poly tyid bva dclist* **using** *lookup-wfTD assms* **by** *simp*

obtain *bv lst* **where** $*$:P ; $\{|bv|\}$; *GNil* $\vdash_{wf}$ *lst* $\wedge$
$(\forall\, c.\ atom\ c \sharp (dclist, lst) \longrightarrow atom\ c \sharp (bva, bv, dclist, lst) \longrightarrow (bva \leftrightarrow c) \cdot dclist = (bv \leftrightarrow c) \cdot lst)$
using *wfTD-elims*(2)[*OF* $*$] **by** *metis*

obtain *c*::*bv* **where** $**$:*atom c* $\sharp$ ((*dclist*, *lst*),(*bva*, *bv*, *dclist*, *lst*)) **using** *obtain-fresh* **by** *metis*
have P ; $\{|bv|\}$; *GNil* $\vdash_{wf}$ *lst* **using** $*$ **by** *metis*
hence *wfTs* $((bv \leftrightarrow c) \cdot P)$ $((bv \leftrightarrow c) \cdot \{|bv|\})$ $((bv \leftrightarrow c) \cdot GNil)$ $((bv \leftrightarrow c) \cdot lst)$ **using** $**$ *wfTs.eqvt* **by** *metis*
hence *wfTs P* $\{|c|\}$ *GNil* $((bva \leftrightarrow c) \cdot dclist)$ **using** $*$ *theta-flip-eq fresh-GNil assms*
proof −
have $\forall\, b\ ba.\ (ba::bv \leftrightarrow b) \cdot P = P$ **by** (*metis* ‹$\vdash_{wf}$ *P*› *theta-flip-eq*)
then show *?thesis*
using $*$ $**$ ‹$(bv \leftrightarrow c) \cdot P$; $(bv \leftrightarrow c) \cdot \{|bv|\}$; $(bv \leftrightarrow c) \cdot GNil \vdash_{wf} (bv \leftrightarrow c) \cdot lst$› **by** *fastforce*
qed
hence *wfTs* $((bva \leftrightarrow c) \cdot P)$ $((bva \leftrightarrow c) \cdot \{|bva|\})$ $((bva \leftrightarrow c) \cdot GNil)$ $((bva \leftrightarrow c) \cdot dclist)$
using *wfTs.eqvt fresh-GNil*
by (*simp add*: *assms*(2) *theta-flip-eq2*)

thus *?thesis* **using** *wfTs.eqvt permute-flip-cancel* **by** *metis*
qed

lemma *wfTh-dclist-poly-unique*:
assumes *wfTh* Θ **and** *AF-typedef-poly tid bva dclist1* $\in$ *set* Θ **and** *AF-typedef-poly tid bva2 dclist2* $\in$ *set* Θ
shows $[[atom\ bva]]lst.\ dclist1 = [[atom\ bva2]]lst.dclist2$
using *assms* **proof**(*induct* Θ *rule*: Θ-*induct*)
case *TNil*
then show *?case* **by** *auto*
next
case (*AF-typedef tid' dclist'* Θ')
then show *?case* **using** *wfTh-elims* **by** *auto*
next
case (*AF-typedef-poly tid bv dclist* Θ')
then show *?case* **using** *wfTh-elims image-eqI name-of-type.simps set-ConsD type-def.eq-iff*
by (*metis Abs1-eq*(3))
qed

lemma *wfTh-poly-lookup-supp*:
fixes *t*::τ
assumes *AF-typedef-poly tid bv dclist* $\in$ *set* Θ **and** (*c*,*t*) $\in$ *set dclist* **and** $\vdash_{wf}$ Θ
shows *supp t* $\subseteq$ $\{atom\ bv\}$
proof −
have *supp dclist* $\subseteq$ $\{atom\ bv\}$ **using** *assms lookup-wfTD wf-supp1 type-def.supp*
by (*metis Diff-single-insert Un-subset-iff list.simps*(15) *supp-Nil supp-of-atom-list*)
then show *?thesis* **using** *assms*(2) **proof**(*induct dclist*)
case *Nil*
then show *?case* **by** *auto*
next
case (*Cons a dclist*)

then show *?case* **using** *supp-Pair supp-Cons*
by (*metis* (*mono-tags*, *hide-lams*) *Un-empty-left Un-empty-right pure-supp subset-Un-eq subset-singletonD supp-list-member*)
qed
qed

lemma *wfTh-poly-supp-b*:
assumes *AF-typedef-poly tid bv dclist* ∈ *set* Θ **and** (*dc*,⟦ *z* : *b* | *c* ⟧) ∈ *set dclist* **and** ⊢$_{wf}$ Θ
shows *supp b* ⊆ {*atom bv*}
using *assms wfTh-poly-lookup-supp τ.supp* **by** *force*

lemma *subst-g-inside*:
fixes *x*::*x* **and** *c*::*c* **and** Γ::Γ **and** Γ′::Γ
assumes *wfG P B* (Γ′ @ (*x*, *b*, *c*[*z*::=*V-var x*]$_{cv}$) #$_\Gamma$ Γ)
shows (Γ′ @ (*x*, *b*, *c*[*z*::=*V-var x*]$_{cv}$) #$_\Gamma$ Γ)[*x*::=*v*]$_{\Gamma v}$ = (Γ′[*x*::=*v*]$_{\Gamma v}$@Γ)
using *assms* **proof**(*induct* Γ′ *rule*: Γ-*induct*)
case *GNil*
then show *?case* **using** *subst-gb.simps* **by** *simp*
next
case (*GCons x′ b′ c′ G*)

hence *wfg*:*wfG P B* (*G* @ (*x*, *b*, *c*[*z*::=*V-var x*]$_{cv}$) #$_\Gamma$ Γ) ∧ *atom x′* ♯ (*G* @ (*x*, *b*, *c*[*z*::=*V-var x*]$_{cv}$) #$_\Gamma$ Γ) **using** *wfG-elims*(*2*)
using *GCons.prems append-g.simps* **by** *metis*
hence *atom x* ∉ *atom-dom* ((*x′*, *b′*, *c′*) #$_\Gamma$ *G*) **using** *GCons wfG-inside-fresh* **by** *fast*
hence *x*≠*x′*
using *GCons append-Cons wfG-inside-fresh atom-dom.simps setG.simps* **by** *simp*
hence ((*GCons* (*x′*, *b′*, *c′*) *G*) @ (*GCons* (*x*, *b*, *c*[*z*::=*V-var x*]$_{cv}$) Γ))[*x*::=*v*]$_{\Gamma v}$ =
(*GCons* (*x′*, *b′*, *c′*) (*G* @ (*GCons* (*x*, *b*, *c*[*z*::=*V-var x*]$_{cv}$) Γ)))[*x*::=*v*]$_{\Gamma v}$ **by** *auto*
also have ... = *GCons* (*x′*, *b′*, *c′*[*x*::=*v*]$_{cv}$) ((*G* @ (*GCons* (*x*, *b*, *c*[*z*::=*V-var x*]$_{cv}$) Γ))[*x*::=*v*]$_{\Gamma v}$)
using *subst-gv.simps* ‹*x*≠*x′*› **by** *simp*
also have ... = (*x′*, *b′*, *c′*[*x*::=*v*]$_{cv}$) #$_\Gamma$ (*G*[*x*::=*v*]$_{\Gamma v}$ @ Γ) **using** *GCons wfg* **by** *blast*
also have ... = ((*x′*, *b′*, *c′*) #$_\Gamma$ *G*)[*x*::=*v*]$_{\Gamma v}$ @ Γ **using** *subst-gv.simps* ‹*x*≠*x′*› **by** *simp*
finally show *?case* **by** *auto*
qed

lemma *wfTh-td-eq*:
assumes *td1* ∈ *set* (*td2* # *P*) **and** *wfTh* (*td2* # *P*) **and** *name-of-type td1* = *name-of-type td2*
shows *td1* = *td2*
proof(*rule ccontr*)
assume *as*: *td1* ≠ *td2*
have *name-of-type td2* ∉ *name-of-type* ‘ *set P* **using** *wfTh-elims*(*2*)[*OF assms*(*2*)] **by** *metis*
moreover have *td1* ∈ *set P* **using** *assms as* **by** *simp*
ultimately have *name-of-type td1* ≠ *name-of-type td2*
by (*metis rev-image-eqI*)
thus *False* **using** *assms* **by** *auto*
qed

lemma *wfTh-td-unique*:
assumes *td1* ∈ *set P* **and** *td2* ∈ *set P* **and** *wfTh P* **and** *name-of-type td1* = *name-of-type td2*
shows *td1* = *td2*

using *assms* **proof**(*induct P rule*: *list.induct*)
case *Nil*
then show *?case* **by** *auto*
next
case (*Cons td* Θ′)
consider *td* = *td1* | *td* = *td2* | *td* ≠ *td1* ∧ *td* ≠ *td2* **by** *auto*
then show *?case* **proof**(*cases*)
case *1*
then show *?thesis* **using** *Cons wfTh-elims wfTh-td-eq* **by** *metis*
next
case *2*
then show *?thesis* **using** *Cons wfTh-elims wfTh-td-eq* **by** *metis*
next
case *3*
then show *?thesis* **using** *Cons wfTh-elims* **by** *auto*
qed
qed

lemma *wfTs-distinct*:
fixes *dclist*::(*string* ∗ τ) *list*
assumes Θ ; *B* ; *GNil* $\vdash_{wf}$ *dclist*
shows *distinct* (*map fst dclist*)
using *assms* **proof**(*induct dclist rule*: *list.induct*)
case *Nil*
then show *?case* **by** *auto*
next
case (*Cons x1 x2*)
then show *?case*
by (*metis Cons.hyps Cons.prems distinct.simps*(2) *fst-conv list.set-map list.simps*(9) *wfTs-elims*(2))

qed

lemma *wfTh-dclist-distinct*:
assumes *AF-typedef s dclist* ∈ *set P* **and** *wfTh P*
shows *distinct* (*map fst dclist*)
proof −
have *wfTD P* (*AF-typedef s dclist*) **using** *assms lookup-wfTD* **by** *auto*
hence *wfTs P* {||} *GNil dclist* **using** *wfTD-elims* **by** *metis*
thus *?thesis* **using** *wfTs-distinct* **by** *metis*
qed

lemma *wfTh-dc-t-unique*:
assumes *AF-typedef s dclist′* ∈ *set P* **and** (*dc*, {| *x′* : *b′* | *c′* |}) ∈ *set dclist′* **and** *AF-typedef s dclist* ∈ *set P* **and** *wfTh P* **and**
(*dc*, {| *x* : *b* | *c* |}) ∈ *set dclist*
shows {| *x′* : *b′* | *c′* |}= {| *x* : *b* | *c* |}
proof −
have *dclist* = *dclist′* **using** *assms wfTh-td-unique name-of-type.simps* **by** *force*
moreover have *distinct* (*map fst dclist*) **using** *wfTh-dclist-distinct assms* **by** *auto*
ultimately show *?thesis* **using** *assms*

by (*meson eq-key-imp-eq-value*)
qed

lemma *wfTs-wfT*:
fixes *dclist*::(*string* ∗τ) *list* **and** *t*::τ
assumes Θ ; $\mathcal{B}$; *GNil* $\vdash_{wf}$ *dclist* **and** (*dc*,*t*) ∈ *set dclist*
shows Θ ; $\mathcal{B}$; *GNil* $\vdash_{wf}$ *t*
using *assms* **proof**(*induct dclist rule*:*list.induct*)
case *Nil*
then show *?case* **by** *auto*
next
case (*Cons x1 x2*)
thus *?case* **using** *wfTs-elims*(*2*)[*OF Cons*(*2*)] **by** *auto*
qed

lemma *wfTh-wfT*:
fixes *t*::τ
assumes *wfTh P* **and** *AF-typedef tid dclist* ∈ *set P* **and** (*dc*,*t*) ∈ *set dclist*
shows *P* ; {||} ; *GNil* $\vdash_{wf}$ *t*
proof −
have *P* $\vdash_{wf}$ *AF-typedef tid dclist* **using** *lookup-wfTD assms* **by** *auto*
hence *P* ; {||} ; *GNil* $\vdash_{wf}$ *dclist* **using** *wfTD-elims* **by** *auto*
thus *?thesis* **using** *wfTs-wfT assms* **by** *auto*
qed

lemma *td-lookup-eq-iff*:
fixes *dc* :: *string* **and** *bva1*::*bv* **and** *bva2*::*bv*
assumes [[*atom bva1*]]*lst*. *dclist1* = [[*atom bva2*]]*lst*. *dclist2* **and** (*dc*, {| *x* : *b* | *c* |}) ∈ *set dclist1*
shows ∃ *x2 b2 c2*. (*dc*, {| *x2* : *b2* | *c2* |}) ∈ *set dclist2*
using *assms* **proof**(*induct dclist1 arbitrary*: *dclist2*)
case *Nil*
then show *?case* **by** *auto*
next
case (*Cons dct1′ dclist1′*)
then obtain *dct2′* **and** *dclist2′* **where** *cons*:*dct2′* # *dclist2′* = *dclist2* **using** *lst-head-cons-neq-nil*[*OF Cons*(*2*)] *list.exhaust* **by** *metis*
hence ∗:[[*atom bva1*]]*lst*. *dclist1′* = [[*atom bva2*]]*lst*. *dclist2′* ∧ [[*atom bva1*]]*lst*. *dct1′* = [[*atom bva2*]]*lst*. *dct2′*
using *Cons lst-head-cons Cons cons* **by** *metis*
show *?case* **proof**(*cases dc*=*fst dct1′*)
case *True*
hence *dc* = *fst dct2′* **using** ∗ *lst-fst*[*THEN lst-pure*]
proof −
show *?thesis*
by (*metis* (*no-types*) *local.*∗ *True* ‹⋀*x2 x1 t2′ t2a t2 t1*. [[*atom x1*]]*lst*. (*t1*, *t2a*) = [[*atom x2*]]*lst*. (*t2*, *t2′*) ⟹ *t1* = *t2*› *prod.exhaust-sel*)
qed
obtain *x2 b2* **and** *c2* **where** *snd dct2′* = {| *x2* : *b2* | *c2* |} **using** *obtain-fresh-z* **by** *metis*
hence (*dc*, {| *x2* : *b2* | *c2* |}) = *dct2′* **using** ‹*dc* = *fst dct2′*›
by (*metis prod.exhaust-sel*)

then show *?thesis* **using** *cons* **by** *force*
next
case *False*
hence (*dc*, {| *x* : *b* | *c* |}) ∈ *set dclist1'* **using** *Cons* **by** *auto*
then show *?thesis* **using** *Cons*
by (*metis local.∗ cons list.set-intros*(*2*))
qed

qed

lemma *lst-t-b-eq-iff*:
fixes *bva1*::*bv* **and** *bva2*::*bv*
assumes [[*atom bva1*]]*lst*. {| *x1* : *b1* | *c1* |} = [[*atom bva2*]]*lst*. {| *x2* : *b2* | *c2* |}
shows [[*atom bva1*]]*lst*. *b1* = [[*atom bva2*]]*lst*.*b2*
proof(*subst Abs1-eq-iff-all*(*3*)[*of bva1 b1 bva2 b2*],*rule*,*rule*,*rule*)
fix *c*::*bv*
assume *atom c* ♯ ({| *x1* : *b1* | *c1* |} , {| *x2* : *b2* | *c2* |}) **and** *atom c* ♯ (*bva1*, *bva2*, *b1*, *b2*)

show (*bva1* ↔ *c*) ∙ *b1* = (*bva2* ↔ *c*) ∙ *b2* **using** *assms Abs1-eq-iff*(*3*) *assms*
by (*metis Abs1-eq-iff-fresh*(*3*) ‹*atom c* ♯ (*bva1*, *bva2*, *b1*, *b2*)› *τ.fresh τ.perm-simps type-eq-subst-eq2*(*2*))
qed

lemma *wfTh-typedef-poly-b-eq-iff*:
assumes *AF-typedef-poly tyid bva1 dclist1* ∈ *set P* **and** (*dc*, {| *x1* : *b1* | *c1* |}) ∈ *set dclist1*
and *AF-typedef-poly tyid bva2 dclist2* ∈ *set P* **and** (*dc*, {| *x2* : *b2* | *c2* |}) ∈ *set dclist2* **and** ⊢$_{wf}$ *P*
shows *b1*[*bva1*::=*b*]$_{bb}$ = *b2*[*bva2*::=*b*]$_{bb}$
proof −
have [[*atom bva1*]]*lst*. *dclist1* = [[*atom bva2*]]*lst*.*dclist2* **using** *assms wfTh-dclist-poly-unique* **by** *metis*
hence [[*atom bva1*]]*lst*. (*dc*,{| *x1* : *b1* | *c1* |}) = [[*atom bva2*]]*lst*. (*dc*,{| *x2* : *b2* | *c2* |}) **using** *wfTh-b-eq-iff assms wfTh-wfTs-poly* **by** *metis*
hence [[*atom bva1*]]*lst*. {| *x1* : *b1* | *c1* |} = [[*atom bva2*]]*lst*. {| *x2* : *b2* | *c2* |} **using** *lst-snd* **by** *metis*
hence [[*atom bva1*]]*lst*. *b1* = [[*atom bva2*]]*lst*.*b2* **using** *lst-t-b-eq-iff* **by** *metis*
thus *?thesis* **using** *subst-b-flip-eq-two subst-b-b-def* **by** *metis*
qed

8.11 Equivariance Lemmas

lemma *x-not-in-u-set*[*simp*]:
fixes *x*::*x* **and** *us*::*u fset*
shows *atom x* ∉ *supp us*
by(*induct us*,*auto*, *simp add*: *supp-finsert supp-at-base*)

lemma *wfS-flip-eq*:
fixes *s1*::*s* **and** *x1*::*x* **and** *s2*::*s* **and** *x2*::*x* **and** Δ::Δ
assumes [[*atom x1*]]*lst*. *s1* = [[*atom x2*]]*lst*. *s2* **and** [[*atom x1*]]*lst*. *t1* = [[*atom x2*]]*lst*. *t2* **and** [[*atom x1*]]*lst*. *c1* = [[*atom x2*]]*lst*. *c2* **and** *atom x2* ♯ Γ **and**
Θ ; $\mathcal{B}$; Γ ⊢$_{wf}$ Δ **and**
Θ ; Φ ; $\mathcal{B}$; (*x1*, *b*, *c1*) #$_{\Gamma}$ Γ ; Δ ⊢$_{wf}$ *s1* : *b-of t1*
shows Θ ; Φ ; $\mathcal{B}$; (*x2*, *b*, *c2*) #$_{\Gamma}$ Γ ; Δ ⊢$_{wf}$ *s2* : *b-of t2*
proof(*cases x1*=*x2*)

case *True*
hence $s1 = s2 \wedge t1 = t2 \wedge c1 = c2$ **using** *assms Abs1-eq-iff* **by** *metis*
then show *?thesis* **using** *assms True* **by** *simp*
next
case *False*
thm *wfD-x-fresh*
have $\vdash_{wf} \Theta \wedge \Theta \vdash_{wf} \Phi \wedge \ \Theta ; \mathcal{B} ; \Gamma \vdash_{wf} \Delta$ **using** *wfX-wfY assms* **by** *metis*
moreover have *atom x1* $\sharp$ Γ **using** *wfX-wfY wfG-elims assms* **by** *metis*
moreover hence *atom x1* $\sharp$ $\Delta \wedge$ *atom x2* $\sharp$ Δ **using** *wfD-x-fresh assms* **by** *auto*
ultimately have $\Theta ; \Phi \ ; \mathcal{B} ; (x2 \leftrightarrow x1) \cdot ((x1, b, c1) \ \#_\Gamma \ \Gamma) ; \Delta \ \vdash_{wf} (x2 \leftrightarrow x1) \cdot s1 : (x2 \leftrightarrow x1) \cdot b\text{-}of\ t1$
using *wfS.eqvt theta-flip-eq phi-flip-eq assms flip-base-eq beta-flip-eq flip-fresh-fresh supp-b-empty* **by** *metis*
hence $\Theta ; \Phi \ ; \mathcal{B} ; ((x2, b, (x2 \leftrightarrow x1) \cdot c1) \ \#_\Gamma \ ((x2 \leftrightarrow x1) \cdot \Gamma)) ; \Delta \ \vdash_{wf} (x2 \leftrightarrow x1) \cdot s1 : b\text{-}of\ ((x2 \leftrightarrow x2) \cdot t1)$ **by** *fastforce*
thus *?thesis* **using** *assms Abs1-eq-iff*
proof −
have *f1*: $x2 = x1 \wedge t2 = t1 \vee x2 \neq x1 \wedge t2 = (x2 \leftrightarrow x1) \cdot t1 \wedge$ *atom x2* $\sharp$ *t1*
by (*metis* (*full-types*) *Abs1-eq-iff*(*3*) ‹[[*atom x1*]]*lst. t1* = [[*atom x2*]]*lst. t2*›)
then have $x2 \neq x1 \wedge s2 = (x2 \leftrightarrow x1) \cdot s1 \wedge$ *atom x2* $\sharp$ *s1* $\longrightarrow$ *b-of t2* = $(x2 \leftrightarrow x1) \cdot$ *b-of t1*
by (*metis b-of.eqvt*)
then show *?thesis*
using *f1* **by** (*metis* (*no-types*) *Abs1-eq-iff*(*3*) *G-cons-flip-fresh3* ‹[[*atom x1*]]*lst. c1* = [[*atom x2*]]*lst. c2*› ‹[[*atom x1*]]*lst. s1* = [[*atom x2*]]*lst. s2*› ‹$\Theta ; \Phi \ ; \mathcal{B} ; (x1, b, c1) \ \#_\Gamma \ \Gamma ; \Delta \vdash_{wf} s1 : b\text{-}of\ t1$› ‹$\Theta ; \Phi \ ; \mathcal{B} ; (x2 \leftrightarrow x1) \cdot ((x1, b, c1) \ \#_\Gamma \ \Gamma) ; \Delta \vdash_{wf} (x2 \leftrightarrow x1) \cdot s1 : (x2 \leftrightarrow x1) \cdot b\text{-}of\ t1$› ‹*atom x1* $\sharp$ Γ› ‹*atom x2* $\sharp$ Γ›)
qed
qed

8.12 Lookup

lemma *wf-not-in-prefix*:
assumes $\Theta ; B \vdash_{wf} (\Gamma'@(x,b1,c1)) \ \#_\Gamma \Gamma)$
shows $x \notin$ *fst* ‘ *setG* Γ'
using *assms* **proof**(*induct* Γ' *rule*: Γ.*induct*)
case *GNil*
then show *?case* **by** *simp*
next
case (*GCons xbc* Γ')
then obtain x' **and** b' **and** c'::c **where** *xbc*: *xbc*=(x',b',c')
using *prod-cases3* **by** *blast*
hence $*$:(*xbc* $\#_\Gamma \Gamma'$) @ (x, *b1*, *c1*) $\#_\Gamma \Gamma$ = ((x',b',c') $\#_\Gamma$(Γ'@ ((x, *b1*, *c1*) $\#_\Gamma \Gamma$))) **by** *simp*
hence *atom* $x' \sharp (\Gamma'@(x,b1,c1) \ \#_\Gamma \Gamma)$ **using** *wfG-elims*(*2*) *GCons* **by** *metis*

moreover have $\Theta ; B \vdash_{wf} (\Gamma' @ (x, b1, c1) \ \#_\Gamma \Gamma)$ **using** *GCons wfG-elims* $*$ **by** *metis*
ultimately have *atom* $x' \notin$ *atom-dom* $(\Gamma'@(x,b1,c1) \ \#_\Gamma \Gamma)$ **using** *wfG-dom-supp GCons append-g.simps xbc fresh-def* **by** *fast*
hence $x' \neq x$ **using** *GCons fresh-GCons xbc* **by** *fastforce*
then show *?case* **using** *GCons xbc setG.simps*
using *Un-commute* ‹$\Theta ; B \vdash_{wf} \Gamma' @ (x, b1, c1) \ \#_\Gamma \Gamma$› *atom-dom.simps* **by** *auto*
qed

lemma *lookup-inside-wf*[*simp*]:
assumes Θ ; $B \vdash_{wf} (\Gamma'@(x,b1,c1)) \#_\Gamma \Gamma)$
shows *Some* $(b1,c1)$ = *lookup* $(\Gamma'@(x,b1,c1)) \#_\Gamma \Gamma)$ x
using *wf-not-in-prefix lookup-inside assms* **by** *fast*

lemma *lookup-weakening*:
fixes Θ::Θ **and** Γ::Γ **and** Γ'::Γ
assumes *Some* (b,c) = *lookup* Γ x **and** *setG* $\Gamma \subseteq$ *setG* Γ' **and** Θ ; $\mathcal{B} \vdash_{wf} \Gamma'$ **and** Θ ; $\mathcal{B} \vdash_{wf} \Gamma$
shows *Some* (b,c) = *lookup* Γ' x
proof −
have $(x,b,c) \in$ *setG* $\Gamma \wedge (\forall\, b'\ c'.\ (x,b',c') \in$ *setG* $\Gamma \longrightarrow b'=b \wedge c'=c)$ **using** *assms lookup-iff setG.simps* **by** *force*
hence $(x,b,c) \in$ *setG* Γ' **using** *assms* **by** *auto*
moreover have $(\forall\, b'\ c'.\ (x,b',c') \in$ *setG* $\Gamma' \longrightarrow b'=b \wedge c'=c)$ **using** *assms wf-g-unique*
using *calculation* **by** *auto*
ultimately show *?thesis* **using** *lookup-iff*
using *assms*(*3*) **by** *blast*
qed

lemma *wfPhi-lookup-fun-unique*:
fixes Φ::Φ
assumes $\Theta \vdash_{wf} \Phi$ **and** *AF-fundef f fd* $\in$ *set* Φ
shows *Some* (*AF-fundef f fd*) = *lookup-fun* Φ *f*
using *assms* **proof**(*induct* Φ *rule*: *list.induct*)
case *Nil*
then show *?case* **using** *lookup-fun.simps* **by** *simp*
next
case (*Cons a* Φ')
then obtain f' **and** fd' **where** *a*:*a* = *AF-fundef f' fd'* **using** *fun-def.exhaust* **by** *auto*
have *wf*: $\Theta \vdash_{wf} \Phi' \wedge f' \notin$ *name-of-fun* ' *set* Φ' **using** *wfPhi-elims Cons a* **by** *metis*
then show *?case* **using** *Cons lookup-fun.simps* **using** *Cons lookup-fun.simps wf a*
by (*metis image-eqI name-of-fun.simps set-ConsD*)
qed

lemma *lookup-fun-weakening*:
fixes Φ'::Φ
assumes *Some fd* = *lookup-fun* Φ *f* **and** *set* $\Phi \subseteq$ *set* Φ' **and** $\Theta \vdash_{wf} \Phi'$
shows *Some fd* = *lookup-fun* Φ' *f*
using *assms* **proof**(*induct* Φ)
case *Nil*
then show *?case* **using** *lookup-fun.simps* **by** *simp*
next
case (*Cons a* Φ'')
then obtain f' **and** fd' **where** *a*: *a* = *AF-fundef f' fd'* **using** *fun-def.exhaust* **by** *auto*
then show *?case* **proof**(*cases f*=*f'*)
case *True*
then show *?thesis* **using** *lookup-fun.simps Cons wfPhi-lookup-fun-unique a*
by (*metis lookup-fun-member subset-iff*)
next
case *False*
then show *?thesis* **using** *lookup-fun.simps Cons*
using ‹*a* = *AF-fundef f' fd'*› **by** *auto*

qed
qed

lemma *fundef-poly-fresh-bv*:
assumes *atom bv2* $\sharp$ (*bv1*,*b1*,*c1*,τ*1*,*s1*)
shows $*$: (*AF-fun-typ-some bv2* (*AF-fun-typ x1* ((*bv1*$\leftrightarrow$*bv2*) $\cdot$ *b1*) ((*bv1*$\leftrightarrow$*bv2*) $\cdot$*c1*) ((*bv1*$\leftrightarrow$*bv2*) $\cdot$ τ*1*) ((*bv1*$\leftrightarrow$*bv2*) $\cdot$ *s1*)) = (*AF-fun-typ-some bv1* (*AF-fun-typ x1 b1 c1* τ*1 s1*)))
(**is** (*AF-fun-typ-some ?bv ?fun-typ* = *AF-fun-typ-some ?bva ?fun-typa*))
proof −
have *1*:*atom bv2* $\notin$ *set* [*atom x1*] **using** *bv-not-in-x-atoms* **by** *simp*
have *2*:*bv1* $\neq$ *bv2* **using** *assms* **by** *auto*
have *3*:(*bv2* $\leftrightarrow$ *bv1*) $\cdot$ *x1* = *x1* **using** *pure-fresh flip-fresh-fresh*
by (*simp add*: *flip-fresh-fresh*)
have *AF-fun-typ x1* ((*bv1* $\leftrightarrow$ *bv2*) $\cdot$ *b1*) ((*bv1* $\leftrightarrow$ *bv2*) $\cdot$ *c1*) ((*bv1* $\leftrightarrow$ *bv2*) $\cdot$ τ*1*) ((*bv1* $\leftrightarrow$ *bv2*) $\cdot$ *s1*) = (*bv2* $\leftrightarrow$ *bv1*) $\cdot$ *AF-fun-typ x1 b1 c1* τ*1 s1*
using *1 2 3 assms* **by** (*simp add*: *flip-commute*)
moreover have (*atom bv2* $\sharp$ *c1* $\wedge$ *atom bv2* $\sharp$ τ*1* $\wedge$ *atom bv2* $\sharp$ *s1* $\vee$ *atom bv2* $\in$ *set* [*atom x1*]) $\wedge$ *atom bv2* $\sharp$ *b1*
using *1 2 3 assms fresh-prod5* **by** *metis*
ultimately show *?thesis* **unfolding** *fun-typ-q.eq-iff Abs1-eq-iff*(*3*) *fun-typ.fresh 1 2* **by** *fastforce*
qed

lemma *wb-b-weakening1*:
fixes Γ::Γ **and** Γ'::Γ **and** *v*::*v* **and** *e*::*e* **and** *c*::*c* **and** τ::τ **and** *ts*::(*string*$*\tau$) *list* **and** Δ::Δ **and** *s*::*s* **and** $\mathcal{B}$::$\mathcal{B}$ **and** *ftq*::*fun-typ-q* **and** *ft*::*fun-typ* **and** *ce*::*ce* **and** *td*::*type-def*
and *cs*::*branch-s* **and** *css*::*branch-list*

shows Θ ; $\mathcal{B}$; Γ $\vdash_{wf}$ *v* : *b* $\Longrightarrow$ $\mathcal{B}$ $|\subseteq|$ $\mathcal{B}'$ $\Longrightarrow$ Θ ; $\mathcal{B}'$; Γ $\vdash_{wf}$ *v* : *b* **and**
Θ ; $\mathcal{B}$; Γ $\vdash_{wf}$ *c* $\Longrightarrow$$\mathcal{B}$ $|\subseteq|$ $\mathcal{B}'$ $\Longrightarrow$ Θ ; $\mathcal{B}'$; Γ $\vdash_{wf}$ *c* **and**
Θ ; $\mathcal{B}$ $\vdash_{wf}$ Γ $\Longrightarrow$$\mathcal{B}$ $|\subseteq|$ $\mathcal{B}'$ $\Longrightarrow$ Θ ; $\mathcal{B}'$ $\vdash_{wf}$ Γ **and**
Θ ; $\mathcal{B}$; Γ $\vdash_{wf}$ τ $\Longrightarrow$ $\mathcal{B}$ $|\subseteq|$ $\mathcal{B}'$ $\Longrightarrow$ Θ ; $\mathcal{B}'$; Γ $\vdash_{wf}$ τ **and**
Θ ; $\mathcal{B}$; Γ $\vdash_{wf}$ *ts* $\Longrightarrow$ $\mathcal{B}$ $|\subseteq|$ $\mathcal{B}'$ $\Longrightarrow$ Θ ; $\mathcal{B}'$; Γ $\vdash_{wf}$ *ts* **and**
$\vdash_{wf}$ *P* $\Longrightarrow$ *True* **and**
wfB Θ $\mathcal{B}$ *b* $\Longrightarrow$ $\mathcal{B}$ $|\subseteq|$ $\mathcal{B}'$ $\Longrightarrow$ *wfB* Θ $\mathcal{B}'$ *b* **and**
Θ ; $\mathcal{B}$; Γ $\vdash_{wf}$ *ce* : *b* $\Longrightarrow$ $\mathcal{B}$ $|\subseteq|$ $\mathcal{B}'$ $\Longrightarrow$ Θ ; $\mathcal{B}'$; Γ $\vdash_{wf}$ *ce* : *b* **and**
Θ $\vdash_{wf}$ *td* $\Longrightarrow$ *True*
proof(*nominal-induct b* **and** *c* **and** Γ **and** τ **and** *ts* **and** *P* **and** *b* **and** *b* **and** *td*
avoiding: $\mathcal{B}'$

rule:*wfV-wfC-wfG-wfT-wfTs-wfTh-wfB-wfCE-wfTD.strong-induct*)
case (*wfV-conspI s bv dclist* Θ *dc x b' c* $\mathcal{B}$ *b* Γ *v*)
show *?case* **proof**
show ‹*AF-typedef-poly s bv dclist* $\in$ *set* Θ› **using** *wfV-conspI* **by** *metis*
show ‹(*dc*, {| *x* : *b'* | *c* |}) $\in$ *set dclist*› **using** *wfV-conspI* **by** *auto*
show ‹ Θ ; $\mathcal{B}'$ $\vdash_{wf}$ *b* › **using** *wfV-conspI* **by** *auto*
show ‹*atom bv* $\sharp$ (Θ, $\mathcal{B}'$, Γ, *b*, *v*)› **using** *fresh-prodN wfV-conspI* **by** *auto*
thus ‹ Θ ; $\mathcal{B}'$; Γ $\vdash_{wf}$ *v* : *b'*[*bv*::=*b*]$_{bb}$ › **using** *wfV-conspI* **by** *simp*
qed
next

case (*wfTI z* Θ $\mathcal{B}$ Γ *b c*)
show *?case* **proof**
show *atom z* $\sharp$ (Θ, $\mathcal{B}'$, Γ) **using** *wfTI* **by** *auto*
show Θ ; $\mathcal{B}'$ $\vdash_{wf}$ *b* **using** *wfTI* **by** *auto*
show Θ ; $\mathcal{B}'$; (*z*, *b*, *TRUE*) $\#_\Gamma$ Γ $\vdash_{wf}$ *c* **using** *wfTI* **by** *auto*
qed
qed((*auto simp add*: *wf-intros* | *metis wf-intros*)+)

lemma *wb-b-weakening2*:
fixes Γ::Γ **and** Γ'::Γ **and** *v*::*v* **and** *e*::*e* **and** *c*::*c* **and** τ::τ **and** *ts*::(*string*$*\tau$) *list* **and** Δ::Δ **and** *s*::*s* **and** $\mathcal{B}$::$\mathcal{B}$ **and** *ftq*::*fun-typ-q* **and** *ft*::*fun-typ* **and** *ce*::*ce* **and** *td*::*type-def*
and *cs*::*branch-s* **and** *css*::*branch-list*

shows
Θ ; Φ ; $\mathcal{B}$; Γ ; Δ $\vdash_{wf}$ *e* : *b* $\Longrightarrow$ $\mathcal{B}$ $|\subseteq|$ $\mathcal{B}'$ $\Longrightarrow$ Θ ; Φ ; $\mathcal{B}'$; Γ ; Δ $\vdash_{wf}$ *e* : *b* **and**
Θ ; Φ ; $\mathcal{B}$; Γ ; Δ $\vdash_{wf}$ *s* : *b* $\Longrightarrow$ $\mathcal{B}$ $|\subseteq|$ $\mathcal{B}'$ $\Longrightarrow$ Θ ; Φ ; $\mathcal{B}'$; Γ ; Δ $\vdash_{wf}$ *s* : *b* **and**
Θ ; Φ ; $\mathcal{B}$; Γ ; Δ ; *tid* ; *dc* ; *t* $\vdash_{wf}$ *cs* : *b* $\Longrightarrow$ $\mathcal{B}$ $|\subseteq|$ $\mathcal{B}'$ $\Longrightarrow$ Θ ; Φ ; $\mathcal{B}'$; Γ ; Δ ; *tid* ; *dc* ; *t* $\vdash_{wf}$ *cs* : *b* **and**
Θ ; Φ ; $\mathcal{B}$; Γ ; Δ ; *tid* ; *dclist* $\vdash_{wf}$ *css* : *b* $\Longrightarrow$ $\mathcal{B}$ $|\subseteq|$ $\mathcal{B}'$ $\Longrightarrow$ Θ ; Φ ; $\mathcal{B}'$; Γ ; Δ ; *tid* ; *dclist* $\vdash_{wf}$ *css* : *b* **and**
Θ $\vdash_{wf}$ (Φ::Φ) $\Longrightarrow$ *True* **and**
Θ ; $\mathcal{B}$; Γ $\vdash_{wf}$ Δ $\Longrightarrow$ $\mathcal{B}$ $|\subseteq|$ $\mathcal{B}'$ $\Longrightarrow$ Θ ; $\mathcal{B}'$; Γ $\vdash_{wf}$ Δ **and**
Θ ; Φ $\vdash_{wf}$ *ftq* $\Longrightarrow$ *True* **and**
Θ ; Φ ; $\mathcal{B}$ $\vdash_{wf}$ *ft* $\Longrightarrow$ $\mathcal{B}$ $|\subseteq|$ $\mathcal{B}'$ $\Longrightarrow$ Θ ; Φ ; $\mathcal{B}'$ $\vdash_{wf}$ *ft*
proof(*nominal-induct b* **and** *b* **and** *b* **and** *b* **and** Φ **and** Δ **and** *ftq* **and** *ft*
avoiding: $\mathcal{B}'$

rule:*wfE-wfS-wfCS-wfCSS-wfPhi-wfD-wfFTQ-wfFT.strong-induct*)

case (*wfE-valI* Θ Φ $\mathcal{B}$ Γ Δ *v b*)
then show *?case* **using** *wf-intros wb-b-weakening1* **by** *metis*
next
case (*wfE-plusI* Θ Φ $\mathcal{B}$ Γ Δ *v1 v2*)
then show *?case* **using** *wf-intros wb-b-weakening1* **by** *metis*
next
case (*wfE-leqI* Θ Φ $\mathcal{B}$ Γ Δ *v1 v2*)
then show *?case* **using** *wf-intros wb-b-weakening1* **by** *metis*
next
case (*wfE-fstI* Θ Φ $\mathcal{B}$ Γ Δ *v1 b1 b2*)
then show *?case* **using** *Wellformed.wfE-fstI wb-b-weakening1* **by** *metis*
next
case (*wfE-sndI* Θ Φ $\mathcal{B}$ Γ Δ *v1 b1 b2*)
then show *?case* **using** *wf-intros wb-b-weakening1* **by** *metis*
next
case (*wfE-concatI* Θ Φ $\mathcal{B}$ Γ Δ *v1 v2*)
then show *?case* **using** *wf-intros wb-b-weakening1* **by** *metis*
next
case (*wfE-splitI* Θ Φ $\mathcal{B}$ Γ Δ *v1 v2*)
then show *?case* **using** *wf-intros wb-b-weakening1* **by** *metis*
next
case (*wfE-lenI* Θ Φ $\mathcal{B}$ Γ Δ *v1*)
then show *?case* **using** *wf-intros wb-b-weakening1* **by** *metis*

next
case (*wfE-appI* Θ Φ $\mathcal{B}$ Γ Δ *f ft v*)
then show *?case* **using** *wf-intros* **using** *wb-b-weakening1* **by** *meson*
next
case (*wfE-appPI* Θ Φ $\mathcal{B}1$ Γ Δ b' *bv1 v1* $\tau 1$ *f1 x1 b1 c1 s1*)

have Θ ; Φ ; $\mathcal{B}'$; Γ ; $\Delta \vdash_{wf}$ *AE-appP f1 b' v1* : (*b-of* $\tau 1$)[*bv1*::=b']$_b$
proof
show $\Theta \vdash_{wf} \Phi$ **using** *wfE-appPI* **by** *auto*
show Θ ; $\mathcal{B}'$; $\Gamma \vdash_{wf} \Delta$ **using** *wfE-appPI* **by** *auto*
show Θ ; $\mathcal{B}' \vdash_{wf} b'$ **using** *wfE-appPI wb-b-weakening1* **by** *auto*
thus *atom bv1* $\sharp$ (Φ, Θ, $\mathcal{B}'$, Γ, Δ, b', *v1*, (*b-of* $\tau 1$)[*bv1*::=b']$_b$)
using *wfE-appPI fresh-prodN* **by** *auto*

show *Some* (*AF-fundef f1* (*AF-fun-typ-some bv1* (*AF-fun-typ x1 b1 c1* $\tau 1$ *s1*))) = *lookup-fun* Φ *f1*
using *wfE-appPI* **by** *auto*
show Θ ; $\mathcal{B}'$; $\Gamma \vdash_{wf}$ *v1* : *b1*[*bv1*::=b']$_b$ **using** *wfE-appPI wb-b-weakening1* **by** *auto*
qed
then show *?case* **by** *auto*
next
case (*wfE-mvarI* Θ Φ $\mathcal{B}$ Γ Δ *u* τ)
then show *?case* **using** *wf-intros wb-b-weakening1* **by** *metis*
next
case (*wfS-valI* Θ Φ $\mathcal{B}$ Γ *v b* Δ)
then show *?case* **using** *wf-intros wb-b-weakening1* **by** *metis*
next
case (*wfS-letI* Θ Φ $\mathcal{B}$ Γ Δ *e b' x s b*)
show *?case* **proof**
show ‹ Θ ; Φ ; $\mathcal{B}'$; Γ ; $\Delta \vdash_{wf}$ *e* : b' › **using** *wfS-letI* **by** *auto*
show ‹ Θ ; Φ ; $\mathcal{B}'$; (*x*, b', *TRUE*) $\#_\Gamma$ Γ ; $\Delta \vdash_{wf}$ *s* : *b* › **using** *wfS-letI* **by** *auto*
show ‹ Θ ; $\mathcal{B}'$; $\Gamma \vdash_{wf}$ Δ › **using** *wfS-letI* **by** *auto*
show ‹*atom x* $\sharp$ (Φ, Θ, $\mathcal{B}'$, Γ, Δ, *e*, *b*)› **using** *wfS-letI* **by** *auto*
qed
next
case (*wfS-let2I* Θ Φ $\mathcal{B}$ Γ Δ *s1* τ *x s2 b*)
then show *?case* **using** *wb-b-weakening1 Wellformed.wfS-let2I* **by** *simp*
next
case (*wfS-ifI* Θ $\mathcal{B}$ Γ *v* Φ Δ *s1 b s2*)
then show *?case* **using** *wb-b-weakening1 Wellformed.wfS-ifI* **by** *simp*
next
case (*wfS-varI* Θ $\mathcal{B}$ Γ τ *v u* Δ Φ *s b*)
then show *?case* **using** *wb-b-weakening1 Wellformed.wfS-varI* **by** *simp*
next
case (*wfS-assignI u* τ Δ Θ $\mathcal{B}$ Γ Φ *v*)
then show *?case* **using** *wb-b-weakening1 Wellformed.wfS-assignI* **by** *simp*
next
case (*wfS-whileI* Θ Φ $\mathcal{B}$ Γ Δ *s1 s2 b*)
then show *?case* **using** *wb-b-weakening1 Wellformed.wfS-whileI* **by** *simp*
next
case (*wfS-seqI* Θ Φ $\mathcal{B}$ Γ Δ *s1 s2 b*)
then show *?case* **using** *Wellformed.wfS-seqI* **by** *metis*
next

case (*wfS-matchI* Θ $\mathcal{B}$ Γ *v tid dclist* Δ Φ *cs b*)
then show *?case* **using** *wb-b-weakening1 Wellformed.wfS-matchI* **by** *metis*
next
case (*wfS-branchI* Θ Φ $\mathcal{B}$ *x* τ Γ Δ *s b tid dc*)
then show *?case* **using** *Wellformed.wfS-branchI* **by** *auto*
next
case (*wfS-finalI* Θ Φ $\mathcal{B}$ Γ Δ *tid dclist' cs b dclist*)
then show *?case* **using** *wf-intros* **by** *metis*
next
case (*wfS-cons* Θ Φ $\mathcal{B}$ Γ Δ *tid dclist' cs b css dclist*)
then show *?case* **using** *wf-intros* **by** *metis*
next
case (*wfD-emptyI* Θ $\mathcal{B}$ Γ)
then show *?case* **using** *wf-intros wb-b-weakening1* **by** *metis*
next
case (*wfD-cons* Θ $\mathcal{B}$ Γ Δ τ *u*)
then show *?case* **using** *wf-intros wb-b-weakening1* **by** *metis*
next
case (*wfPhi-emptyI* Θ)
then show *?case* **using** *wf-intros wb-b-weakening1* **by** *metis*
next
case (*wfPhi-consI f* Θ Φ *ft*)
then show *?case* **using** *wf-intros wb-b-weakening1* **by** *metis*
next
case (*wfFTSome* Θ *bv ft*)
then show *?case* **using** *wf-intros wb-b-weakening1* **by** *metis*
next
case (*wfFTI* Θ *B b* Φ *x c s* τ)
then show *?case* **using** *wb-b-weakening1 Wellformed.wfFTI* **by** *auto*
next
case (*wfS-assertI* Θ Φ $\mathcal{B}$ *x c* Γ Δ *s b*)
show *?case* **proof**
show ‹ Θ ; Φ ; $\mathcal{B}'$; (*x*, *B-bool*, *c*) $\#_\Gamma$ Γ ; Δ $\vdash_{wf}$ *s* : *b* › **using** *wb-b-weakening1 wfS-assertI* **by** *simp*
show ‹ Θ ; $\mathcal{B}'$; Γ $\vdash_{wf}$ *c* › **using** *wb-b-weakening1 wfS-assertI* **by** *simp*
show ‹ Θ ; $\mathcal{B}'$; Γ $\vdash_{wf}$ Δ › **using** *wb-b-weakening1 wfS-assertI* **by** *simp*
have *atom x* $\sharp$ $\mathcal{B}'$ **using** *x-not-in-b-set fresh-def* **by** *metis*
thus ‹*atom x* $\sharp$ (Φ, Θ, $\mathcal{B}'$, Γ, Δ, *c*, *b*, *s*)› **using** *wfS-assertI fresh-prodN* **by** *simp*
qed

qed(*auto*)

lemmas *wb-b-weakening* = *wb-b-weakening1 wb-b-weakening2*

lemma *wfG-b-weakening*:
fixes Γ::Γ
assumes $\mathcal{B}$ |$\subseteq$| $\mathcal{B}'$ **and** Θ ; $\mathcal{B}$ $\vdash_{wf}$ Γ
shows Θ ; $\mathcal{B}'$ $\vdash_{wf}$ Γ
using *wb-b-weakening assms* **by** *auto*

lemma *wfT-b-weakening*:
fixes Γ::Γ **and** Θ::Θ **and** τ::τ

assumes $\mathcal{B}$ $|\subseteq|$ $\mathcal{B}'$ **and** Θ ; $\mathcal{B}$; Γ $\vdash_{wf}$ τ
shows Θ ; $\mathcal{B}'$; Γ $\vdash_{wf}$ τ
using *wb-b-weakening assms* **by** *auto*

lemma *wfB-subst-wfB*:
fixes τ::τ **and** b'::b **and** b::b
assumes Θ ; $\{|bv|\}$ $\vdash_{wf}$ b **and** Θ ; $\mathcal{B}$ $\vdash_{wf}$ b'
shows Θ ; $\mathcal{B}$ $\vdash_{wf}$ $b[bv::=b']_{bb}$
using *assms* **proof**(*nominal-induct b rule*:*b.strong-induct*)
case *B-int*
hence Θ ; $\{||\}$ $\vdash_{wf}$ *B-int* **using** *wfB-intI wfX-wfY* **by** *fast*
then show *?case* **using** *subst-bb.simps wb-b-weakening* **by** *fastforce*
next
case *B-bool*
hence Θ ; $\{||\}$ $\vdash_{wf}$ *B-bool* **using** *wfB-boolI wfX-wfY* **by** *fast*
then show *?case* **using** *subst-bb.simps wb-b-weakening* **by** *fastforce*
next
case (*B-id x*)
hence Θ ; $\mathcal{B}$ $\vdash_{wf}$ (*B-id x*) **using** *wfB-consI wfB-elims wfX-wfY* **by** *metis*
then show *?case* **using** *subst-bb.simps*(*4*) **by** *auto*
next
case (*B-pair x1 x2*)
then show *?case* **using** *subst-bb.simps*
by (*metis wfB-elims*(*1*) *wfB-pairI*)
next
case *B-unit*
hence Θ ; $\{||\}$ $\vdash_{wf}$ *B-unit* **using** *wfB-unitI wfX-wfY* **by** *fast*
then show *?case* **using** *subst-bb.simps wb-b-weakening* **by** *fastforce*
next
case *B-bitvec*
hence Θ ; $\{||\}$ $\vdash_{wf}$ *B-bitvec* **using** *wfB-bitvecI wfX-wfY* **by** *fast*
then show *?case* **using** *subst-bb.simps wb-b-weakening* **by** *fastforce*
next
case (*B-var x*)
then show *?case*
proof −
have *False*
using *B-var.prems*(*1*) *wfB.cases* **by** *fastforce*
then show *?thesis* **by** *metis*
qed
next
case (*B-app s b*)
then obtain *bv' dclist* **where** ∗:*AF-typedef-poly s bv' dclist* ∈ *set* Θ ∧ Θ ; $\{|bv|\}$ $\vdash_{wf}$ b **using** *wfB-elims* **by** *metis*
thm *wfB-appI*
show *?case* **unfolding** *subst-b-simps* **proof**
show $\vdash_{wf}$ Θ **using** *B-app wfX-wfY* **by** *metis*
show Θ ; $\mathcal{B}$ $\vdash_{wf}$ $b[bv::=b']_{bb}$ **using** ∗ *B-app forget-subst wfB-supp fresh-def*
by (*metis ex-in-conv subset-empty subst-b-b-def supp-empty-fset*)
show *AF-typedef-poly s bv' dclist* ∈ *set* Θ **using** ∗ **by** *auto*
qed
qed

lemma *wfT-subst-wfB*:
fixes τ::τ **and** b'::b
assumes Θ ; $\{|bv|\}$; $(x,\ b,\ c)$ $\#_\Gamma$ *GNil* $\vdash_{wf}$ τ **and** Θ ; $\mathcal{B}$ $\vdash_{wf}$ b'
shows Θ ; $\mathcal{B}$ $\vdash_{wf}$ (*b-of* τ)$[bv::=b']_{bb}$
proof −
obtain b **where** Θ ; $\{|bv|\} \vdash_{wf} b \wedge$ *b-of* $\tau = b$ **using** *wfT-elims b-of.simps assms* **by** *metis*
thus *?thesis* **using** *wfB-subst-wfB assms* **by** *auto*
qed

lemma *wfG-cons-unique*:
assumes (*x1*,*b1*,*c1*) ∈ *setG* (((*x*,*b*,*c*) $\#_\Gamma\Gamma$)) **and** *wfG* Θ $\mathcal{B}$ (((*x*,*b*,*c*) $\#_\Gamma\Gamma$)) **and** *x* = *x1*
shows *b1* = *b* ∧ *c1* = *c*
proof −
have *x1* ∉ *fst* ‘ *setG* Γ
proof −
have *atom x1* ♯ Γ **using** *assms wfG-cons* **by** *metis*
then show *?thesis*
using *fresh-gamma-elem*
by (*metis assms*(*2*) *atom-dom.simps rev-image-eqI wfG-cons2 wfG-x-fresh*)
qed
thus *?thesis* **using** *assms* **by** *force*
qed

lemma *wfG-member-unique*:
assumes (*x1*,*b1*,*c1*) ∈ *setG* (Γ′@((*x*,*b*,*c*) $\#_\Gamma\Gamma$)) **and** *wfG* Θ $\mathcal{B}$ (Γ′@((*x*,*b*,*c*) $\#_\Gamma\Gamma$)) **and** *x* = *x1*
shows *b1* = *b* ∧ *c1* = *c*
using *assms* **proof**(*induct* Γ′ *rule*: Γ-*induct*)
case *GNil*
then show *?case* **using** *wfG-suffix wfG-cons-unique append-g.simps* **by** *metis*
next
case (*GCons x′ b′ c′* Γ′)
moreover hence (*x1*, *b1*, *c1*) ∈ *setG* (Γ′ @ (*x*, *b*, *c*) $\#_\Gamma$ Γ) **using** *wf-not-in-prefix* **by** *fastforce*
ultimately show *?case* **using** *wfG-cons* **by** *fastforce*
qed

8.13 Function Definitions

lemma *wb-phi-weakening*:
fixes Γ::Γ **and** Γ′::Γ **and** *v*::*v* **and** *e*::*e* **and** *c*::*c* **and** τ::τ **and** *ts*::(*string*∗τ) *list* **and** Δ::Δ **and** *s*::*s* **and** $\mathcal{B}$::$\mathcal{B}$ **and** *ftq*::*fun-typ-q* **and** *ft*::*fun-typ* **and** *ce*::*ce* **and** *td*::*type-def*
and *cs*::*branch-s* **and** *css*::*branch-list* **and** Φ::Φ
shows
Θ ; Φ ; $\mathcal{B}$; Γ ; $\Delta \vdash_{wf} e : b \Longrightarrow \Theta \vdash_{wf} \Phi' \Longrightarrow$ *set* $\Phi \subseteq$ *set* $\Phi' \Longrightarrow \Theta$; Φ' ; $\mathcal{B}$; Γ ; $\Delta \vdash_{wf} e : b$ **and**
Θ ; Φ ; $\mathcal{B}$; Γ ; $\Delta \vdash_{wf} s : b \Longrightarrow \Theta \vdash_{wf} \Phi' \Longrightarrow$ *set* $\Phi \subseteq$ *set* $\Phi' \Longrightarrow \Theta$; Φ' ; $\mathcal{B}$; Γ ; $\Delta \vdash_{wf} s : b$ **and**
Θ ; Φ ; $\mathcal{B}$; Γ ; Δ ; *tid* ; *dc* ; $t \vdash_{wf} cs : b \Longrightarrow \Theta \vdash_{wf} \Phi' \Longrightarrow$ *set* $\Phi \subseteq$ *set* $\Phi' \Longrightarrow \Theta$; Φ' ; $\mathcal{B}$; Γ ; Δ ; *tid* ; *dc* ; $t \vdash_{wf} cs : b$ **and**
Θ ; Φ ; $\mathcal{B}$; Γ ; Δ ; *tid* ; *dclist* $\vdash_{wf}$ *css* : $b \Longrightarrow \Theta \vdash_{wf} \Phi' \Longrightarrow$ *set* $\Phi \subseteq$ *set* $\Phi' \Longrightarrow \Theta$; Φ' ; $\mathcal{B}$; Γ ; Δ ; *tid* ; *dclist* $\vdash_{wf}$ *css* : b **and**
$\Theta \vdash_{wf} (\Phi::\Phi) \Longrightarrow$ *True* **and**

Θ ; $\mathcal{B}$; Γ $\vdash_{wf}$ Δ $\Longrightarrow$ *True* **and**
Θ ; Φ $\vdash_{wf}$ *ftq* $\Longrightarrow$ Θ $\vdash_{wf}$ Φ' $\Longrightarrow$ *set* Φ $\subseteq$ *set* Φ' $\Longrightarrow$ Θ ; Φ' $\vdash_{wf}$ *ftq* **and**
Θ ; Φ ; $\mathcal{B}$ $\vdash_{wf}$ *ft* $\Longrightarrow$ Θ $\vdash_{wf}$ Φ' $\Longrightarrow$ *set* Φ $\subseteq$ *set* Φ' $\Longrightarrow$ Θ ; Φ' ; $\mathcal{B}$ $\vdash_{wf}$ *ft*
proof(*nominal-induct*
b **and** *b* **and** *b* **and** *b* **and** Φ **and** Δ **and** *ftq* **and** *ft*
avoiding: Φ'
rule:*wfE-wfS-wfCS-wfCSS-wfPhi-wfD-wfFTQ-wfFT.strong-induct*)
case (*wfE-valI* Θ Φ $\mathcal{B}$ Γ Δ *v b*)
then show *?case* **using** *wf-intros* **by** *metis*
next
case (*wfE-plusI* Θ Φ $\mathcal{B}$ Γ Δ *v1 v2*)
then show *?case* **using** *wf-intros* **by** *metis*
next
case (*wfE-leqI* Θ Φ $\mathcal{B}$ Γ Δ *v1 v2*)
then show *?case* **using** *wf-intros* **by** *metis*
next
case (*wfE-fstI* Θ Φ $\mathcal{B}$ Γ Δ *v1 b1 b2*)
then show *?case* **using** *wf-intros* **by** *metis*
next
case (*wfE-sndI* Θ Φ $\mathcal{B}$ Γ Δ *v1 b1 b2*)
then show *?case* **using** *wf-intros* **by** *metis*
next
case (*wfE-concatI* Θ Φ $\mathcal{B}$ Γ Δ *v1 v2*)
then show *?case* **using** *wf-intros* **by** *metis*
next
case (*wfE-splitI* Θ Φ $\mathcal{B}$ Γ Δ *v1 v2*)
then show *?case* **using** *wf-intros* **by** *metis*
next
case (*wfE-lenI* Θ Φ $\mathcal{B}$ Γ Δ *v1*)
then show *?case* **using** *wf-intros* **by** *metis*
next
case (*wfE-appI* Θ Φ $\mathcal{B}$ Γ Δ *f x b c* τ *s v*)
then show *?case* **using** *wf-intros lookup-fun-weakening* **by** *metis*
next
case (*wfE-appPI* Θ Φ $\mathcal{B}$ Γ Δ *b'* *bv v* τ *f x b c s*)
show *?case* **proof**
show ‹ Θ $\vdash_{wf}$ Φ' › **using** *wfE-appPI* **by** *auto*
show ‹ Θ ; $\mathcal{B}$; Γ $\vdash_{wf}$ Δ › **using** *wfE-appPI* **by** *auto*
show ‹ Θ ; $\mathcal{B}$ $\vdash_{wf}$ *b'* › **using** *wfE-appPI* **by** *auto*
show ‹*atom bv* $\sharp$ (Φ', Θ, $\mathcal{B}$, Γ, Δ, *b'*, *v*, (*b-of* τ)[*bv*::=b']$_b$)› **using** *wfE-appPI* **by** *auto*
show ‹*Some* (*AF-fundef f* (*AF-fun-typ-some bv* (*AF-fun-typ x b c* τ *s*))) = *lookup-fun* Φ' *f*›
using *wfE-appPI lookup-fun-weakening* **by** *metis*
show ‹ Θ ; $\mathcal{B}$; Γ $\vdash_{wf}$ *v* : b[*bv*::=b']$_b$ › **using** *wfE-appPI* **by** *auto*
qed

next
case (*wfE-mvarI* Θ Φ $\mathcal{B}$ Γ Δ *u* τ)
then show *?case* **using** *wf-intros* **by** *metis*
next
case (*wfS-valI* Θ Φ $\mathcal{B}$ Γ *v b* Δ)
then show *?case* **using** *wf-intros* **by** *metis*
next

case (*wfS-letI* Θ Φ $\mathcal{B}$ Γ Δ *e b' x s b*)
then show *?case* **using** *Wellformed.wfS-letI* **by** *fastforce*
next
case (*wfS-let2I* Θ Φ $\mathcal{B}$ Γ Δ *s1 b' x s2 b*)
then show *?case* **using** *Wellformed.wfS-let2I* **by** *fastforce*
next
case (*wfS-ifI* Θ $\mathcal{B}$ Γ *v* Φ Δ *s1 b s2*)
then show *?case* **using** *wf-intros* **by** *metis*
next
case (*wfS-varI* Θ $\mathcal{B}$ Γ τ *v u* Φ Δ *b s*)
show *?case* **proof**
show ‹ Θ ; $\mathcal{B}$; Γ $\vdash_{wf}$ τ › **using** *wfS-varI* **by** *simp*
show ‹ Θ ; $\mathcal{B}$; $\Gamma \vdash_{wf}$ *v* : *b-of* τ › **using** *wfS-varI* **by** *simp*
show ‹*atom u* $\sharp$ (Φ', Θ, $\mathcal{B}$, Γ, Δ, τ, *v*, *b*)› **using** *wfS-varI* **by** *simp*
show ‹ Θ ; Φ' ; $\mathcal{B}$; Γ ; (*u*, τ) $\#_{\Delta}$ $\Delta \vdash_{wf}$ *s* : *b* › **using** *wfS-varI* **by** *simp*
qed
next
case (*wfS-assignI u* τ Δ Θ $\mathcal{B}$ Γ Φ *v*)
then show *?case* **using** *wf-intros* **by** *metis*
next
case (*wfS-whileI* Θ Φ $\mathcal{B}$ Γ Δ *s1 s2 b*)
then show *?case* **using** *wf-intros* **by** *metis*
next
case (*wfS-seqI* Θ Φ $\mathcal{B}$ Γ Δ *s1 s2 b*)
then show *?case* **using** *wf-intros* **by** *metis*
next
case (*wfS-matchI* Θ $\mathcal{B}$ Γ *v tid dclist* Δ Φ *cs b*)
then show *?case* **using** *wf-intros* **by** *metis*
next
case (*wfS-branchI* Θ Φ $\mathcal{B}$ *x* τ Γ Δ *s b tid dc*)
then show *?case* **using** *Wellformed.wfS-branchI* **by** *fastforce*
next
case (*wfS-assertI* Θ Φ $\mathcal{B}$ *x c* Γ Δ *s b*)
show *?case* **proof**

show ‹ Θ ; Φ' ; $\mathcal{B}$; (*x*, *B-bool*, *c*) $\#_{\Gamma}$ Γ ; $\Delta \vdash_{wf}$ *s* : *b* › **using** *wfS-assertI* **by** *auto*
next
show ‹ Θ ; $\mathcal{B}$; Γ $\vdash_{wf}$ *c* › **using** *wfS-assertI* **by** *auto*
next
show ‹ Θ ; $\mathcal{B}$; $\Gamma \vdash_{wf}$ Δ › **using** *wfS-assertI* **by** *auto*

have *atom x* $\sharp$ Φ' **using** *wfS-assertI wfPhi-supp fresh-def* **by** *blast*
thus ‹*atom x* $\sharp$ (Φ', Θ, $\mathcal{B}$, Γ, Δ, *c*, *b*, *s*)› **using** *fresh-prodN wfS-assertI wfPhi-supp fresh-def* **by** *auto*
qed
qed(*auto*|*metis wf-intros*)+

lemma *wfT-fun-return-t*:
fixes $\tau a'::\tau$ **and** $\tau'::\tau$
assumes Θ ; $\mathcal{B}$; (*xa*, *b*, *ca*) $\#_{\Gamma}$ *GNil* $\vdash_{wf}$ $\tau a'$ **and** (*AF-fun-typ x b c* τ' *s'*) = (*AF-fun-typ xa b ca* $\tau a'$ *sa'*)
shows Θ ; $\mathcal{B}$; (*x*, *b*, *c*) $\#_{\Gamma}$ *GNil* $\vdash_{wf}$ τ'

proof −
obtain *cb*::*x* **where** *xf*: *atom cb* ♯ (*c*, *τ′*, *s′*, *sa′*, *τa′*, *ca*, *x* , *xa*) **using** *obtain-fresh* **by** *blast*
hence *atom cb* ♯ (*c*, *τ′*, *s′*, *sa′*, *τa′*, *ca*) ∧ *atom cb* ♯ (*x*, *xa*, ((*c*, *τ′*), *s′*), (*ca*, *τa′*), *sa′*) **using** *fresh-prod6 fresh-prod4 fresh-prod8* **by** *auto*
hence ∗:*c*[*x*::=*V-var cb*]$_{cv}$ = *ca*[*xa*::=*V-var cb*]$_{cv}$ ∧ *τ′*[*x*::=*V-var cb*]$_{\tau v}$ = *τa′*[*xa*::=*V-var cb*]$_{\tau v}$ **using** *assms τ.eq-iff Abs1-eq-iff-all* **by** *auto*

have ∗∗: Θ ; $\mathcal{B}$; (*xa* ↔ *cb*) · ((*xa*, *b*, *ca*) #$_\Gamma$ *GNil*) ⊢$_{wf}$ (*xa* ↔ *cb*) · *τa′* **using** *assms True-eqvt beta-flip-eq theta-flip-eq wfG-wf*
by (*metis GCons-eqvt GNil-eqvt wfT.eqvt wfT-wf*)

have Θ ; $\mathcal{B}$; (*x* ↔ *cb*) · ((*x*, *b*, *c*) #$_\Gamma$ *GNil*) ⊢$_{wf}$ (*x* ↔ *cb*) · *τ′* **proof** −
have (*xa* ↔ *cb*) · *xa* = (*x* ↔ *cb*) · *x* **using** *xf* **by** *auto*
hence (*x* ↔ *cb*) · ((*x*, *b*, *c*) #$_\Gamma$ *GNil*) = (*xa* ↔ *cb*) · ((*xa*, *b*, *ca*) #$_\Gamma$ *GNil*) **using** ∗ ∗∗ *xf G-cons-flip fresh-GNil* **by** *simp*
thus *?thesis* **using** ∗∗ ∗ *xf* **by** *simp*
qed
thus *?thesis* **using** *beta-flip-eq theta-flip-eq wfT-wf wfG-wf* ∗ ∗∗ *True-eqvt wfT.eqvt permute-flip-cancel* **by** *metis*
qed

lemma *wfFT-wf-aux*:
fixes *τ*::*τ* **and** Θ::Θ **and** Φ::Φ **and** *ft* :: *fun-typ-q* **and** *s*::*s* **and** Δ::Δ
assumes Θ ; Φ ; *B* ⊢$_{wf}$ (*AF-fun-typ x b c τ s*)
shows Θ ; *B* ; (*x*,*b*,*c*) #$_\Gamma$ *GNil* ⊢$_{wf}$ *τ* ∧ Θ ; Φ ; *B* ; (*x*,*b*,*c*) #$_\Gamma$ *GNil* ; []$_\Delta$ ⊢$_{wf}$ *s* : *b-of τ*
proof −

obtain *xa* **and** *ca* **and** *sa* **and** *τ′* **where** ∗:Θ ; *B* ⊢$_{wf}$ *b* ∧ (Θ ; Φ ; *B* ; (*xa*, *b*, *ca*) #$_\Gamma$ *GNil* ; []$_\Delta$ ⊢$_{wf}$ *sa* : *b-of τ′*) ∧
supp sa ⊆ {*atom xa*} ∧ (Θ ; *B* ; (*xa*, *b*, *ca*) #$_\Gamma$ *GNil* ⊢$_{wf}$ *τ′*) ∧
AF-fun-typ x b c τ s = *AF-fun-typ xa b ca τ′ sa*
using *wfFT.simps*[*of* Θ Φ *B AF-fun-typ x b c τ s*] *assms* **by** *auto*

moreover hence (*AF-fun-typ x b c τ s*) = (*AF-fun-typ xa b ca τ′ sa*) **by** *simp*
ultimately have Θ ; *B* ; (*x*,*b*,*c*) #$_\Gamma$*GNil* ⊢$_{wf}$ *τ* **using** *wfT-fun-return-t* **by** *metis*
moreover have (Θ ; Φ ; *B* ; (*x*, *b*, *c*) #$_\Gamma$ *GNil* ; []$_\Delta$ ⊢$_{wf}$ *s* : *b-of τ*) **proof** −
have ∗∗:Θ ; Φ ; *B* ; (*xa*, *b*, *ca*) #$_\Gamma$ *GNil* ; []$_\Delta$ ⊢$_{wf}$ *sa* : *b-of τ′* **using** ∗ **by** *auto*
moreover have [[*atom xa*]]*lst*. *sa* = [[*atom x*]]*lst*. *s* ∧ [[*atom xa*]]*lst*. *τ′* = [[*atom x*]]*lst*. *τ* ∧ [[*atom xa*]]*lst*. *ca* = [[*atom x*]]*lst*. *c*
using ∗ *fun-typ.eq-iff lst-fst lst-snd* **by** *metis*
moreover have *atom x* ♯ *GNil* **by** *auto*
ultimately show *?thesis* **using** *assms wfS-flip-eq wfD-emptyI wfG-nilI wfX-wfY* ∗ **by** *metis*
qed
ultimately show *?thesis* **by** *auto*
qed

lemma *wfFT-simple-wf*:
fixes *τ*::*τ* **and** Θ::Θ **and** Φ::Φ **and** *ft* :: *fun-typ-q* **and** *s*::*s* **and** Δ::Δ
assumes Θ ; Φ ⊢$_{wf}$ (*AF-fun-typ-none* (*AF-fun-typ x b c τ s*))
shows Θ ; {||} ; (*x*,*b*,*c*) #$_\Gamma$*GNil* ⊢$_{wf}$ *τ* ∧ Θ ; Φ ; {||} ; (*x*,*b*,*c*) #$_\Gamma$*GNil* ; []$_\Delta$ ⊢$_{wf}$ *s* : *b-of τ*
proof −
have ∗:Θ ; Φ ; {||} ⊢$_{wf}$ (*AF-fun-typ x b c τ s*) **using** *wfFTQ-elims assms* **by** *auto*

thus *?thesis* **using** *wfFT-wf-aux* **by** *auto*
qed

lemma *wfFT-poly-wf*:
fixes τ::τ **and** Θ::Θ **and** Φ::Φ **and** *ftq* :: *fun-typ-q* **and** *s*::*s* **and** Δ::Δ
assumes Θ ; Φ $\vdash_{wf}$ (*AF-fun-typ-some bv* (*AF-fun-typ x b c* τ *s*))
shows Θ ; {|*bv*|} ; (*x*,*b*,*c*) $\#_\Gamma$ *GNil* $\vdash_{wf}$ τ $\wedge$ Θ ; Φ ; {|*bv*|} ; (*x*,*b*,*c*) $\#_\Gamma$ *GNil* ; $[]_\Delta$ $\vdash_{wf}$ *s* : *b-of* τ
proof −

obtain *bv1 ft1* **where** $*$:Θ ; Φ ; {|*bv1*|} $\vdash_{wf}$ *ft1* $\wedge$ [[*atom bv1*]]*lst*. *ft1* = [[*atom bv*]]*lst*. *AF-fun-typ x b c* τ *s*
using *wfFTQ-elims*(*3*)[*OF assms*] **by** *metis*

show *?thesis* **proof**(*cases bv1* = *bv*)
case *True*
then show *?thesis* **using** $*$ *fun-typ-q.eq-iff Abs1-eq-iff* **by** (*metis* (*no-types*, *hide-lams*) *wfFT-wf-aux*)
next
case *False*
obtain *x1 b1 c1 t1 s1* **where** $**$: *ft1* = *AF-fun-typ x1 b1 c1 t1 s1* **using** *fun-typ.eq-iff*
by (*meson fun-typ.exhaust*)

hence *eqv*: (*bv* $\leftrightarrow$ *bv1*) $\cdot$ *AF-fun-typ x1 b1 c1 t1 s1* = *AF-fun-typ x b c* τ *s* $\wedge$ *atom bv1* $\sharp$ *AF-fun-typ x b c* τ *s* **using**
Abs1-eq-iff(*3*) $*$ *False* **by** *metis*

have (*bv* $\leftrightarrow$ *bv1*) $\cdot$ Θ ; (*bv* $\leftrightarrow$ *bv1*) $\cdot$ Φ ; (*bv* $\leftrightarrow$ *bv1*) $\cdot$ {|*bv1*|} $\vdash_{wf}$ (*bv* $\leftrightarrow$ *bv1*) $\cdot$ *ft1* **using** *wfFT.eqvt* $*$ **by** *metis*
moreover have (*bv* $\leftrightarrow$ *bv1*) $\cdot$ Φ = Φ **using** *phi-flip-eq wfX-wfY* $*$ **by** *metis*
moreover have (*bv* $\leftrightarrow$ *bv1*) $\cdot$ Θ =Θ **using** *wfX-wfY* $*$ *theta-flip-eq2* **by** *metis*
moreover have (*bv* $\leftrightarrow$ *bv1*) $\cdot$ *ft1* = *AF-fun-typ x b c* τ *s* **using** *eqv* $**$ **by** *metis*
ultimately have Θ ; Φ ; {|*bv*|} $\vdash_{wf}$ *AF-fun-typ x b c* τ *s* **by** *auto*
thus *?thesis* **using** *wfFT-wf-aux* **by** *auto*
qed
qed

lemma *wfFT-poly-wfT*:
fixes τ::τ **and** Θ::Θ **and** Φ::Φ **and** *ft* :: *fun-typ-q*
assumes Θ ; Φ $\vdash_{wf}$ (*AF-fun-typ-some bv* (*AF-fun-typ x b c* τ *s*))
shows Θ ; {| *bv* |} ; (*x*,*b*,*c*) $\#_\Gamma$ *GNil* $\vdash_{wf}$ τ
using *wfFT-poly-wf assms* **by** *simp*

lemma *wfPhi-f-simple-wf*:
fixes τ::τ **and** Θ::Θ **and** Φ::Φ **and** *ft* :: *fun-typ-q* **and** *s*::*s* **and** Φ'::Φ
assumes *AF-fundef f* (*AF-fun-typ-none* (*AF-fun-typ x b c* τ *s*)) $\in$ *set* Φ **and** Θ $\vdash_{wf}$ Φ **and** *set* Φ $\subseteq$ *set* Φ' **and** Θ $\vdash_{wf}$ Φ'
shows Θ ; {||} ; (*x*,*b*,*c*) $\#_\Gamma$ *GNil* $\vdash_{wf}$ τ $\wedge$ Θ ; Φ' ; {||} ; (*x*,*b*,*c*) $\#_\Gamma$ *GNil* ; $[]_\Delta$ $\vdash_{wf}$ *s* : *b-of* τ
using *assms* **proof**(*induct* Φ *rule*: Φ-*induct*)
case *PNil*
then show *?case* **by** *auto*
next
case (*PConsSome f1 bv x1 b1 c1* τ*1 s'* Φ'')

hence *AF-fundef f (AF-fun-typ-none (AF-fun-typ x b c τ s)) ∈ set Φ″* **by** *auto*
moreover have *Θ ⊢wf Φ″ ∧ set Φ″ ⊆ set Φ′* **using** *wfPhi-elims(3) PConsSome* **by** *auto*
ultimately show *?case* **using** *PConsSome wfPhi-elims wfFT-simple-wf* **by** *auto*
next
case (*PConsNone f′ x′ b′ c′ τ′ s′ Φ″*)
show *?case* **proof**(*cases f=f′*)
case *True*
have *AF-fun-typ-none (AF-fun-typ x′ b′ c′ τ′ s′) = AF-fun-typ-none (AF-fun-typ x b c τ s)*
by (*metis PConsNone.prems(1) PConsNone.prems(2) True fun-def.eq-iff image-eqI name-of-fun.simps set-ConsD wfPhi-elims(2)*)
hence *∗:Θ ; Φ″ ⊢wf AF-fun-typ-none (AF-fun-typ x b c τ s)* **using** *wfPhi-elims(2)[OF PConsNone(3)]* **by** *metis*
hence *Θ ; Φ″ ; {||} ⊢wf (AF-fun-typ x b c τ s)* **using** *wfFTQ-elims(1)* **by** *metis*
thus *?thesis* **using** *wfFT-simple-wf[OF ∗] wb-phi-weakening PConsNone* **by** *force*
next
case *False*
hence *AF-fundef f (AF-fun-typ-none (AF-fun-typ x b c τ s)) ∈ set Φ″* **using** *PConsNone* **by** *simp*
moreover have *Θ ⊢wf Φ″ ∧ set Φ″ ⊆ set Φ′* **using** *wfPhi-elims(3) PConsNone* **by** *auto*
ultimately show *?thesis* **using** *PConsNone wfPhi-elims wfFT-simple-wf* **by** *auto*
qed
qed

lemma *wfPhi-f-simple-wfT*:
fixes *τ::τ* **and** *Θ::Θ* **and** *Φ::Φ* **and** *ft :: fun-typ-q*
assumes *Some (AF-fundef f (AF-fun-typ-none (AF-fun-typ x b c τ s))) = lookup-fun Φ f* **and** *Θ ⊢wf Φ*
shows *Θ ; {||} ; (x,b,c) #Γ GNil ⊢wf τ*
using *wfPhi-f-simple-wf assms* **using** *lookup-fun-member* **by** *blast*

lemma *wfPhi-f-simple-supp-t*:
fixes *τ::τ* **and** *Θ::Θ* **and** *Φ::Φ* **and** *ft :: fun-typ-q*
assumes *Some (AF-fundef f (AF-fun-typ-none (AF-fun-typ x b c τ s))) = lookup-fun Φ f* **and** *Θ ⊢wf Φ*
shows *supp τ ⊆ { atom x }*
using *wfPhi-f-simple-wfT wfT-supp assms* **by** *fastforce*

lemma *wfPhi-f-poly-wfT*:
fixes *τ::τ* **and** *Θ::Θ* **and** *Φ::Φ* **and** *ft :: fun-typ-q*
assumes *Some (AF-fundef f (AF-fun-typ-some bv (AF-fun-typ x b c τ s))) = lookup-fun Φ f* **and** *Θ ⊢wf Φ*
shows *Θ ; {| bv |} ; (x,b,c) #Γ GNil ⊢wf τ*
using *assms* **proof**(*induct Φ rule*: *Φ-induct*)
case *PNil*
then show *?case* **by** *auto*
next
case (*PConsSome f1 bv1 x1 b1 c1 τ1 s′ Φ′*)
then show *?case* **proof**(*cases f1=f*)
case *True*
hence *lookup-fun (AF-fundef f1 (AF-fun-typ-some bv1 (AF-fun-typ x1 b1 c1 τ1 s′)) # Φ′) f = Some (AF-fundef f1 (AF-fun-typ-some bv1 (AF-fun-typ x1 b1 c1 τ1 s′)))* **using**
lookup-fun.simps **using** *PConsSome.prems* **by** *simp*
then show *?thesis* **using** *PConsSome.prems wfPhi-elims wfFT-poly-wfT*

```
    by (metis option.inject)
  next
    case False
    then show ?thesis using PConsSome using lookup-fun.simps
      using wfPhi-elims(3) by auto
  qed
next
  case (PConsNone f' x' b' c' τ' s' Φ')
  then show ?case proof(cases f'=f)
    case True
     then have ∗:Θ ; Φ' ⊢wf AF-fun-typ-none (AF-fun-typ x' b' c' τ' s') using lookup-fun.simps
PConsNone wfPhi-elims by metis
    thus ?thesis using PConsNone wfFT-poly-wfT wfPhi-elims lookup-fun.simps
      by (metis fun-def.eq-iff fun-typ-q.distinct(1) option.inject)
  next
    case False
    thus ?thesis using PConsNone wfPhi-elims
      by (metis False lookup-fun.simps(2))
  qed
qed

lemma wfPhi-f-poly-supp-b:
  fixes τ::τ and Θ::Θ and Φ::Φ and ft :: fun-typ-q
  assumes Some (AF-fundef f (AF-fun-typ-some bv (AF-fun-typ x b c τ s))) = lookup-fun Φ f and Θ
⊢wf Φ
  shows supp b ⊆ supp bv
proof −
  have Θ ; {|bv|} ; (x,b,c) #Γ GNil ⊢wf τ using wfPhi-f-poly-wfT assms by auto
  thus ?thesis using wfT-wf wfG-cons wfB-supp by fastforce
qed

lemma wfPhi-f-poly-supp-t:
  fixes τ::τ and Θ::Θ and Φ::Φ and ft :: fun-typ-q
  assumes Some (AF-fundef f (AF-fun-typ-some bv (AF-fun-typ x b c τ s))) = lookup-fun Φ f and Θ
⊢wf Φ
  shows supp τ ⊆ { atom x , atom bv }
 using wfPhi-f-poly-wfT[OF assms, THEN wfT-supp] atom-dom.simps supp-at-base by auto

lemma b-of-supp:
  supp (b-of t) ⊆ supp t
proof(nominal-induct t rule:τ.strong-induct)
  case (T-refined-type x b c)
  then show ?case by auto
qed

lemma wfPhi-f-poly-supp-b-of-t:
  fixes τ::τ and Θ::Θ and Φ::Φ and ft :: fun-typ-q
  assumes Some (AF-fundef f (AF-fun-typ-some bv (AF-fun-typ x b c τ s))) = lookup-fun Φ f and Θ
⊢wf Φ
  shows supp (b-of τ) ⊆ { atom bv }
proof −
  have atom x ∉ supp (b-of τ) using x-fresh-b by auto
```

moreover have *supp* (*b-of* τ) $\subseteq$ { *atom x* , *atom bv* } **using** *wfPhi-f-poly-supp-t*
using *supp-at-base b-of.simps wfPhi-f-poly-supp-t* τ*.supp b-of-supp assms* **by** *fast*
ultimately show *?thesis* **by** *blast*
qed

lemma *wfPhi-f-supp-c*:
fixes τ::τ **and** Θ::Θ **and** Φ::Φ **and** *ft* :: *fun-typ-q*
assumes *Some* (*AF-fundef f* (*AF-fun-typ-none* (*AF-fun-typ x b c* τ *s*))) = *lookup-fun* Φ *f* **and** Θ $\vdash_{wf}$ Φ
shows *supp c* $\subseteq$ { *atom x* }
proof −
have Θ ; {||} ; (*x*,*b*,*c*) $\#_\Gamma$ *GNil* $\vdash_{wf}$ τ **using** *wfPhi-f-simple-wfT assms* **by** *auto*
thus *?thesis* **using** *wfG-wfC wfC-supp wfT-wf* **by** *fastforce*
qed

lemma *wfPhi-f-poly-supp-c*:
fixes τ::τ **and** Θ::Θ **and** Φ::Φ **and** *ft* :: *fun-typ-q*
assumes *Some* (*AF-fundef f* (*AF-fun-typ-some bv* (*AF-fun-typ x b c* τ *s*))) = *lookup-fun* Φ *f* **and** Θ $\vdash_{wf}$ Φ
shows *supp c* $\subseteq$ { *atom x*, *atom bv* }
proof −
have Θ ; {|*bv*|} ; (*x*,*b*,*c*) $\#_\Gamma$ *GNil* $\vdash_{wf}$ τ **using** *wfPhi-f-poly-wfT assms* **by** *auto*
thus *?thesis* **using** *wfG-wfC wfC-supp wfT-wf*
using *supp-at-base* **by** *fastforce*
qed

lemma *wfPhi-f-simple-supp-b*:
fixes τ::τ **and** Θ::Θ **and** Φ::Φ **and** *ft* :: *fun-typ-q*
assumes *Some* (*AF-fundef f* (*AF-fun-typ-none* (*AF-fun-typ x b c* τ *s*))) = *lookup-fun* Φ *f* **and** Θ $\vdash_{wf}$ Φ
shows *supp b* = {}
proof −
have Θ ; {||} ; (*x*,*b*,*c*) $\#_\Gamma$ *GNil* $\vdash_{wf}$ τ **using** *wfPhi-f-simple-wfT assms* **by** *auto*
thus *?thesis* **using** *wfT-wf wfG-cons wfB-supp* **by** *fastforce*
qed

lemma *wfPhi-f-simple-supp-s*:
fixes τ::τ **and** Θ::Θ **and** Φ::Φ **and** *ft* :: *fun-typ-q*
assumes *Some* (*AF-fundef f* (*AF-fun-typ-none* (*AF-fun-typ x b c* τ *s*))) = *lookup-fun* Φ *f* **and** Θ $\vdash_{wf}$ Φ
shows *supp s* $\subseteq$ {*atom x*}
proof −
have *AF-fundef f* (*AF-fun-typ-none* (*AF-fun-typ x b c* τ *s*)) $\in$ *set* Φ **using** *lookup-fun-member assms* **by** *auto*
hence Θ ; Φ ; {||} ; (*x*,*b*,*c*) $\#_\Gamma$ *GNil* ; []$_\Delta$ $\vdash_{wf}$ *s* : *b-of* τ **using** *wfPhi-f-simple-wf assms* **by** *auto*
thus *?thesis* **using** *wf-supp*(*3*) *atom-dom.simps setG.simps x-not-in-u-set x-not-in-b-set setD.simps*
using *wf-supp2*(*2*) **by** *fastforce*
qed

lemma *wfPhi-f-poly-wf*:
fixes τ::τ **and** Θ::Θ **and** Φ::Φ **and** *ft* :: *fun-typ-q* **and** *s*::*s* **and** Φ'::Φ
assumes *AF-fundef f* (*AF-fun-typ-some bv* (*AF-fun-typ x b c* τ *s*)) $\in$ *set* Φ **and** Θ $\vdash_{wf}$ Φ **and** *set*

$\Phi \subseteq$ *set* Φ' **and** $\Theta \vdash_{wf} \Phi'$
shows Θ ; $\{|bv|\}$; (x,b,c) $\#_\Gamma$ *GNil* $\vdash_{wf} \tau \wedge \Theta$; Φ' ; $\{|bv|\}$; (x,b,c) $\#_\Gamma$ *GNil* ; $[]_\Delta \vdash_{wf} s$: *b-of* τ
using *assms* **proof**(*induct* Φ *rule*: Φ-*induct*)
case *PNil*
then show *?case* **by** *auto*
next
case (*PConsNone f x b c* τ *s'* Φ'')
moreover have $\Theta \vdash_{wf} \Phi'' \wedge$ *set* $\Phi'' \subseteq$ *set* Φ' **using** *wfPhi-elims*(*3*) *PConsNone* **by** *auto*
ultimately show *?case* **using** *PConsNone wfPhi-elims wfFT-poly-wf* **by** *auto*
next
case (*PConsSome f1 bv1 x1 b1 c1* τ*1 s1* Φ'')
show *?case* **proof**(*cases f=f1*)
case *True*
have *AF-fun-typ-some bv1* (*AF-fun-typ x1 b1 c1* τ*1 s1*) = *AF-fun-typ-some bv* (*AF-fun-typ x b c* τ *s*)
by (*metis PConsSome.prems*(*1*) *PConsSome.prems*(*2*) *True fun-def.eq-iff list.set-intros*(*1*) *option.inject wfPhi-lookup-fun-unique*)
hence $*$:Θ ; Φ'' $\vdash_{wf}$ *AF-fun-typ-some bv* (*AF-fun-typ x b c* τ *s*) **using** *wfPhi-elims PConsSome* **by** *metis*
thus *?thesis* **using** *wfFT-poly-wf* $*$ *wb-phi-weakening PConsSome*
by (*meson set-subset-Cons*)
next
case *False*
hence *AF-fundef f* (*AF-fun-typ-some bv* (*AF-fun-typ x b c* τ *s*)) $\in$ *set* Φ'' **using** *PConsSome*
by (*meson fun-def.eq-iff set-ConsD*)
moreover have $\Theta \vdash_{wf} \Phi'' \wedge$ *set* $\Phi'' \subseteq$ *set* Φ' **using** *wfPhi-elims*(*3*) *PConsSome*
by (*meson dual-order.trans set-subset-Cons*)
ultimately show *?thesis* **using** *PConsSome wfPhi-elims wfFT-poly-wf*
by *blast*
qed
qed

lemma *wfPhi-f-poly-supp-s*:
fixes τ::τ **and** Θ::Θ **and** Φ::Φ **and** *ft* :: *fun-typ-q*
assumes *Some* (*AF-fundef f* (*AF-fun-typ-some bv* (*AF-fun-typ x b c* τ *s*))) = *lookup-fun* Φ *f* **and** $\Theta \vdash_{wf} \Phi$
shows *supp s* $\subseteq$ {*atom x*, *atom bv*}
proof −
have *AF-fundef f* (*AF-fun-typ-some bv* (*AF-fun-typ x b c* τ *s*)) $\in$ *set* Φ **using** *lookup-fun-member assms* **by** *auto*
hence Θ ; Φ ; $\{|bv|\}$; (x,b,c) $\#_\Gamma$ *GNil* ; $[]_\Delta \vdash_{wf} s$: *b-of* τ **using** *wfPhi-f-poly-wf assms* **by** *auto*
thus *?thesis* **using** *wf-supp2*(*2*) *atom-dom.simps setG.simps setD.simps*
using *Un-insert-right supp-at-base* **by** *fastforce*
qed

lemmas *wfPhi-f-supp* = *wfPhi-f-poly-supp-b wfPhi-f-simple-supp-b wfPhi-f-poly-supp-c*
wfPhi-f-simple-supp-t wfPhi-f-poly-supp-t wfPhi-f-simple-supp-t wfPhi-f-poly-wfT wfPhi-f-simple-wfT wfPhi-f-simple-supp-s

lemma *fun-typ-eq-ret-unique*:
assumes (*AF-fun-typ x1 b1 c1* τ*1' s1'*) = (*AF-fun-typ x2 b2 c2* τ*2' s2'*)
shows $\tau 1'[x1::=v]_{\tau v} = \tau 2'[x2::=v]_{\tau v}$

proof −
have $[[atom\ x1]]lst.\ \tau 1' = [[atom\ x2]]lst.\ \tau 2'$ **using** *assms lst-fst fun-typ.eq-iff lst-snd* **by** *metis*
thus *?thesis* **using** *subst-v-flip-eq-two*[*of x1* $\tau 1'$ *x2* $\tau 2'$ *v*] *subst-v-*τ*-def* **by** *metis*

qed

lemma *fun-typ-eq-body-unique*:
fixes $v::v$ **and** $x1::x$ **and** $x2::x$ **and** $s1'::s$ **and** $s2'::s$
assumes $(AF\text{-}fun\text{-}typ\ x1\ b1\ c1\ \tau 1'\ s1') = (AF\text{-}fun\text{-}typ\ x2\ b2\ c2\ \tau 2'\ s2')$
shows $s1'[x1::=v]_{sv} = s2'[x2::=v]_{sv}$
proof −
have $[[atom\ x1]]lst.\ s1' = [[atom\ x2]]lst.\ s2'$ **using** *assms lst-fst fun-typ.eq-iff lst-snd* **by** *metis*
thus *?thesis* **using** *subst-v-flip-eq-two*[*of x1 s1' x2 s2' v*] *subst-v-s-def* **by** *metis*
qed

lemma *fun-ret-unique*:
assumes $Some\ (AF\text{-}fundef\ f\ (AF\text{-}fun\text{-}typ\text{-}none\ (AF\text{-}fun\text{-}typ\ x1\ b1\ c1\ \tau 1'\ s1'))) = lookup\text{-}fun\ \Phi\ f$
and $Some\ (AF\text{-}fundef\ f\ (AF\text{-}fun\text{-}typ\text{-}none\ (AF\text{-}fun\text{-}typ\ x2\ b2\ c2\ \tau 2'\ s2'))) = lookup\text{-}fun\ \Phi\ f$
shows $\tau 1'[x1::=v]_{\tau v} = \tau 2'[x2::=v]_{\tau v}$
proof −
have ∗: $(AF\text{-}fundef\ f\ (AF\text{-}fun\text{-}typ\text{-}none\ (AF\text{-}fun\text{-}typ\ x1\ b1\ c1\ \tau 1'\ s1'))) = (AF\text{-}fundef\ f\ (AF\text{-}fun\text{-}typ\text{-}none\ (AF\text{-}fun\text{-}typ\ x2\ b2\ c2\ \tau 2'\ s2')))$ **using** *option.inject assms* **by** *metis*
thus *?thesis* **using** *fun-typ-eq-ret-unique fun-def.eq-iff fun-typ-q.eq-iff* **by** *metis*

qed

lemma *fun-poly-arg-unique*:
fixes $bv1::bv$ **and** $bv2::bv$ **and** $b::b$ **and** $\tau 1::\tau$ **and** $\tau 2::\tau$
assumes $[[atom\ bv1]]lst.\ (AF\text{-}fun\text{-}typ\ x1\ b1\ c1\ \tau 1\ s1) = [[atom\ bv2]]lst.\ (AF\text{-}fun\text{-}typ\ x2\ b2\ c2\ \tau 2\ s2)$ (**is** $[[atom\ ?x]]lst.\ ?a = [[atom\ ?y]]lst.\ ?b$)
shows $\{\!|\ x1 : b1[bv1::=b]_{bb} \mid c1[bv1::=b]_{cb}\ |\!\} = \{\!|\ x2 : b2[bv2::=b]_{bb} \mid c2[bv2::=b]_{cb}\ |\!\}$
proof −
obtain $c::bv$ **where** ∗:$atom\ c \sharp (b,b1,b2,c1,c2) \wedge atom\ c \sharp (bv1,\ bv2,\ AF\text{-}fun\text{-}typ\ x1\ b1\ c1\ \tau 1\ s1,\ AF\text{-}fun\text{-}typ\ x2\ b2\ c2\ \tau 2\ s2)$ **using** *obtain-fresh fresh-Pair* **by** *metis*
hence $(bv1 \leftrightarrow c) \cdot AF\text{-}fun\text{-}typ\ x1\ b1\ c1\ \tau 1\ s1 = (bv2 \leftrightarrow c) \cdot AF\text{-}fun\text{-}typ\ x2\ b2\ c2\ \tau 2\ s2$ **using** *Abs1-eq-iff-all*(3)[*of ?x ?a ?y ?b*] *assms* **by** *metis*
hence $AF\text{-}fun\text{-}typ\ x1\ ((bv1 \leftrightarrow c) \cdot b1)\ ((bv1 \leftrightarrow c) \cdot c1)\ ((bv1 \leftrightarrow c) \cdot \tau 1)\ ((bv1 \leftrightarrow c) \cdot s1) = AF\text{-}fun\text{-}typ\ x2\ ((bv2 \leftrightarrow c) \cdot b2)\ ((bv2 \leftrightarrow c) \cdot c2)\ ((bv2 \leftrightarrow c) \cdot \tau 2)\ ((bv2 \leftrightarrow c) \cdot s2)$
using *fun-typ-flip* **by** *metis*
hence ∗∗:$\{\!|\ x1 :((bv1 \leftrightarrow c) \cdot b1) \mid ((bv1 \leftrightarrow c) \cdot c1)\ |\!\} = \{\!|\ x2 : ((bv2 \leftrightarrow c) \cdot b2) \mid ((bv2 \leftrightarrow c) \cdot c2)\ |\!\}$ (**is** $\{\!|\ x1 : ?b1 \mid ?c1\ |\!\} = \{\!|\ x2 : ?b2 \mid ?c2\ |\!\}$) **using** *fun-arg-unique-aux* **by** *metis*
hence $\{\!|\ x1 :((bv1 \leftrightarrow c) \cdot b1) \mid ((bv1 \leftrightarrow c) \cdot c1)\ |\!\}[c::=b]_{\tau b} = \{\!|\ x2 : ((bv2 \leftrightarrow c) \cdot b2) \mid ((bv2 \leftrightarrow c) \cdot c2)\ |\!\}[c::=b]_{\tau b}$ **by** *metis*
hence $\{\!|\ x1 :((bv1 \leftrightarrow c) \cdot b1)[c::=b]_{bb} \mid ((bv1 \leftrightarrow c) \cdot c1)[c::=b]_{cb}\ |\!\} = \{\!|\ x2 : ((bv2 \leftrightarrow c) \cdot b2)[c::=b]_{bb} \mid ((bv2 \leftrightarrow c) \cdot c2)[c::=b]_{cb}\ |\!\}$ **using** *subst-tb.simps* **by** *metis*
thus *?thesis* **using** ∗ *flip-subst-subst subst-b-c-def subst-b-b-def fresh-prodN flip-commute* **by** *metis*
qed

lemma *fun-poly-ret-unique*:
assumes $Some\ (AF\text{-}fundef\ f\ (AF\text{-}fun\text{-}typ\text{-}some\ bv1\ (AF\text{-}fun\text{-}typ\ x1\ b1\ c1\ \tau 1'\ s1'))) = lookup\text{-}fun\ \Phi\ f$ **and** $Some\ (AF\text{-}fundef\ f\ (AF\text{-}fun\text{-}typ\text{-}some\ bv2\ (AF\text{-}fun\text{-}typ\ x2\ b2\ c2\ \tau 2'\ s2'))) = lookup\text{-}fun\ \Phi\ f$
shows $\tau 1'[bv1::=b]_{\tau b}[x1::=v]_{\tau v} = \tau 2'[bv2::=b]_{\tau b}[x2::=v]_{\tau v}$

proof −
  have ∗: (*AF-fundef f* (*AF-fun-typ-some bv1* (*AF-fun-typ x1 b1 c1* $\tau 1'$ *s1'*))) = (*AF-fundef f* (*AF-fun-typ-some bv2* (*AF-fun-typ x2 b2 c2* $\tau 2'$ *s2'*))) **using** *option.inject assms* **by** *metis*
  hence *AF-fun-typ-some bv1* (*AF-fun-typ x1 b1 c1* $\tau 1'$ *s1'*) = *AF-fun-typ-some bv2* (*AF-fun-typ x2 b2 c2* $\tau 2'$ *s2'*)
    (**is** *AF-fun-typ-some bv1 ?ft1* = *AF-fun-typ-some bv2 ?ft2*) **using** *fun-def.eq-iff* **by** *metis*
  hence ∗∗:[[*atom bv1*]]*lst. ?ft1* = [[*atom bv2*]]*lst. ?ft2* **using** *fun-typ-q.eq-iff*(*1*) **by** *metis*

  hence ∗:*subst-ft-b ?ft1 bv1 b* = *subst-ft-b ?ft2 bv2 b* **using** *subst-b-flip-eq-two subst-b-fun-typ-def* **by** *metis*
  have [[*atom x1*]]*lst.* $\tau 1'[bv1::=b]_{\tau b}$ = [[*atom x2*]]*lst.* $\tau 2'[bv2::=b]_{\tau b}$
    apply(*rule lst-snd*[*of - c1*$[bv1::=b]_{cb}$ *- - c2*$[bv2::=b]_{cb}$])
    apply(*rule lst-fst*[*of - - s1'*$[bv1::=b]_{sb}$ *- - s2'*$[bv2::=b]_{sb}$])
    using ∗ *subst-ft-b.simps fun-typ.eq-iff* **by** *metis*
  thus *?thesis* **using** *subst-v-flip-eq-two subst-v-*τ*-def* **by** *metis*
qed

lemma *fun-poly-body-unique*:
  assumes *Some* (*AF-fundef f* (*AF-fun-typ-some bv1* (*AF-fun-typ x1 b1 c1* $\tau 1'$ *s1'*))) = *lookup-fun* Φ *f* **and** *Some* (*AF-fundef f* (*AF-fun-typ-some bv2* (*AF-fun-typ x2 b2 c2* $\tau 2'$ *s2'*))) = *lookup-fun* Φ *f*
  shows *s1'*$[bv1::=b]_{sb}[x1::=v]_{sv}$ = *s2'*$[bv2::=b]_{sb}[x2::=v]_{sv}$
proof −
  have ∗: (*AF-fundef f* (*AF-fun-typ-some bv1* (*AF-fun-typ x1 b1 c1* $\tau 1'$ *s1'*))) = (*AF-fundef f* (*AF-fun-typ-some bv2* (*AF-fun-typ x2 b2 c2* $\tau 2'$ *s2'*)))
    using *option.inject assms* **by** *metis*
  hence *AF-fun-typ-some bv1* (*AF-fun-typ x1 b1 c1* $\tau 1'$ *s1'*) = *AF-fun-typ-some bv2* (*AF-fun-typ x2 b2 c2* $\tau 2'$ *s2'*)
    (**is** *AF-fun-typ-some bv1 ?ft1* = *AF-fun-typ-some bv2 ?ft2*) **using** *fun-def.eq-iff* **by** *metis*
  hence ∗∗:[[*atom bv1*]]*lst. ?ft1* = [[*atom bv2*]]*lst. ?ft2* **using** *fun-typ-q.eq-iff*(*1*) **by** *metis*

  hence ∗:*subst-ft-b ?ft1 bv1 b* = *subst-ft-b ?ft2 bv2 b* **using** *subst-b-flip-eq-two subst-b-fun-typ-def* **by** *metis*
  have [[*atom x1*]]*lst. s1'*$[bv1::=b]_{sb}$ = [[*atom x2*]]*lst. s2'*$[bv2::=b]_{sb}$
    using *lst-snd lst-fst subst-ft-b.simps fun-typ.eq-iff*
    by (*metis local.*∗)

  thus *?thesis* **using** *subst-v-flip-eq-two subst-v-s-def* **by** *metis*
qed

lemma *funtyp-eq-iff-equalities*:
  fixes *s'*::*s* **and** *s*::*s*
  assumes [[*atom x'*]]*lst.* ((*c'*, τ'), *s'*) = [[*atom x*]]*lst.* ((*c*, τ), *s*)
  shows $\{\!|\ x' : b\ |\ c'\ |\!\}$ = $\{\!|\ x : b\ |\ c\ |\!\}$ ∧ *s'*$[x'::=v]_{sv}$ = *s*$[x::=v]_{sv}$ ∧ $\tau'[x'::=v]_{\tau v}$ = $\tau[x::=v]_{\tau v}$
proof −
  have [[*atom x'*]]*lst. s'* = [[*atom x*]]*lst. s* **and** [[*atom x'*]]*lst.* τ' = [[*atom x*]]*lst.* τ **and**
      [[*atom x'*]]*lst. c'* = [[*atom x*]]*lst. c* **using** *lst-snd lst-fst assms* **by** *metis*+
  thus *?thesis* **using** *subst-v-flip-eq-two* τ*.eq-iff*
    by (*metis assms fun-typ.eq-iff fun-typ-eq-body-unique fun-typ-eq-ret-unique*)
qed

8.14 Weakening

lemma *wfX-wfB1*:
fixes Γ::Γ **and** Γ'::Γ **and** v::v **and** e::e **and** c::c **and** τ::τ **and** ts::(*string*$*\tau$) *list* **and** Δ::Δ **and** s::s **and** b::b **and** $\mathcal{B}$::$\mathcal{B}$ **and** Φ::Φ **and** *ftq*::*fun-typ-q* **and** *ft*::*fun-typ* **and** *ce*::*ce* **and** *td*::*type-def* **and** *cs*::*branch-s* **and** *css*::*branch-list*
shows *wfV-wfB*: Θ ; $\mathcal{B}$; Γ $\vdash_{wf}$ v : b $\Longrightarrow$ Θ ; $\mathcal{B}$ $\vdash_{wf}$ b **and**
Θ ; $\mathcal{B}$; Γ $\vdash_{wf}$ c $\Longrightarrow$ *True* **and**
Θ ; $\mathcal{B}$ $\vdash_{wf}$ Γ $\Longrightarrow$ *True* **and**
wfT-wfB: Θ ; $\mathcal{B}$; Γ $\vdash_{wf}$ τ $\Longrightarrow$ Θ ; $\mathcal{B}$ $\vdash_{wf}$ *b-of* τ **and**
Θ ; $\mathcal{B}$; Γ $\vdash_{wf}$ *ts* $\Longrightarrow$ *True* **and**
$\vdash_{wf}$ Θ $\Longrightarrow$ *True* **and**
Θ ; $\mathcal{B}$ $\vdash_{wf}$ b $\Longrightarrow$ *True* **and**
wfCE-wfB: Θ ; $\mathcal{B}$; Γ $\vdash_{wf}$ *ce* : b $\Longrightarrow$ Θ ; $\mathcal{B}$ $\vdash_{wf}$ b **and**
Θ $\vdash_{wf}$ *td* $\Longrightarrow$ *True*
proof(*induct rule*:*wfV-wfC-wfG-wfT-wfTs-wfTh-wfB-wfCE-wfTD.inducts*)
case (*wfV-varI* Θ $\mathcal{B}$ Γ b c x)
hence (x,b,c) $\in$ *setG* Γ **using** *lookup-iff wfV-wf* **using** *lookup-in-g* **by** *presburger*
hence b $\in$ *fst*‘*snd*‘*setG* Γ **by** *force*
hence *wfB* Θ $\mathcal{B}$ b **using** *wfG-wfB wfV-varI* **by** *metis*
then show *?case* **using** *wfV-elims wfG-wf wf-intros* **by** *metis*
next
case (*wfV-litI* Θ Γ l)
moreover have *wfTh* Θ **using** *wfV-wf wfG-wf wfV-litI* **by** *metis*
ultimately show *?case* **using** *wfV-wf wfG-wf wf-intros base-for-lit.simps l.exhaust* **by** *metis*
next
case (*wfV-pairI* Θ Γ *v1 b1 v2 b2*)
then show *?case* **using** *wfG-wf wf-intros* **by** *metis*
next
case (*wfV-consI s dclist* Θ *dc x b c B* Γ v)
then show *?case*
using *wfV-wf wfG-wf wfB-consI* **by** *metis*
next
case (*wfV-conspI s bv dclist* Θ *dc x b' c* $\mathcal{B}$ b Γ v)
then show *?case*
using *wfV-wf wfG-wf* **using** *wfB-appI* **by** *metis*
next
case (*wfCE-valI* Θ $\mathcal{B}$ Γ v b)
then show *?case* **using** *wfB-elims* **by** *auto*
next
case (*wfCE-plusI* Θ $\mathcal{B}$ Γ *v1 v2*)
then show *?case* **using** *wfB-elims* **by** *auto*
next
case (*wfCE-leqI* Θ $\mathcal{B}$ Γ *v1 v2*)
then show *?case* **using** *wfV-wf wfG-wf wf-intros wfX-wfY* **by** *metis*
next
case (*wfCE-fstI* Θ $\mathcal{B}$ Γ *v1 b1 b2*)
then show *?case* **using** *wfB-elims* **by** *metis*
next
case (*wfCE-sndI* Θ $\mathcal{B}$ Γ *v1 b1 b2*)
then show *?case* **using** *wfB-elims* **by** *metis*
next
case (*wfCE-concatI* Θ $\mathcal{B}$ Γ *v1 v2*)

then show *?case* **using** *wfB-elims* **by** *auto*
next
case (*wfCE-lenI* Θ $\mathcal{B}$ Γ *v1*)
then show *?case* **using** *wfV-wf wfG-wf wf-intros wfX-wfY* **by** *metis*
qed(*auto* | *metis wfV-wf wfG-wf wf-intros*)+

lemma *wfX-wfB2*:
fixes Γ::Γ **and** Γ'::Γ **and** *v*::*v* **and** *e*::*e* **and** *c*::*c* **and** τ::τ **and** *ts*::(*string*$*\tau$) *list* **and** Δ::Δ **and** *s*::*s* **and** *b*::*b* **and** $\mathcal{B}$::$\mathcal{B}$ **and** Φ::Φ **and** *ftq*::*fun-typ-q* **and** *ft*::*fun-typ* **and** *ce*::*ce* **and** *td*::*type-def* **and** *cs*::*branch-s* **and** *css*::*branch-list*
shows
wfE-wfB: Θ ; Φ ; $\mathcal{B}$; Γ ; Δ $\vdash_{wf}$ *e* : *b* $\Longrightarrow$ Θ ; $\mathcal{B}$ $\vdash_{wf}$ *b* **and**
wfS-wfB: Θ ; Φ ; $\mathcal{B}$; Γ ; Δ $\vdash_{wf}$ *s* : *b* $\Longrightarrow$ Θ ; $\mathcal{B}$ $\vdash_{wf}$ *b* **and**
wfCS-wfB: Θ ; Φ ; $\mathcal{B}$; Γ ; Δ ; *tid* ; *dc* ; *t* $\vdash_{wf}$ *cs* : *b* $\Longrightarrow$ Θ ; $\mathcal{B}$ $\vdash_{wf}$ *b* **and**
wfCSS-wfB: Θ ; Φ ; $\mathcal{B}$; Γ ; Δ ; *tid* ; *dclist* $\vdash_{wf}$ *css* : *b* $\Longrightarrow$ Θ ; $\mathcal{B}$ $\vdash_{wf}$ *b* **and**
Θ $\vdash_{wf}$ Φ $\Longrightarrow$ *True* **and**
Θ ; $\mathcal{B}$; Γ $\vdash_{wf}$ Δ $\Longrightarrow$ *True* **and**
Θ ; Φ $\vdash_{wf}$ *ftq* $\Longrightarrow$ *True* **and**
Θ ; Φ ; $\mathcal{B}$ $\vdash_{wf}$ *ft* $\Longrightarrow$ $\mathcal{B}$ $|\subseteq|$ $\mathcal{B}'$ $\Longrightarrow$ Θ ; Φ ; $\mathcal{B}'$ $\vdash_{wf}$ *ft*
proof(*induct rule*:*wfE-wfS-wfCS-wfCSS-wfPhi-wfD-wfFTQ-wfFT.inducts*)
case (*wfE-valI* Θ Φ $\mathcal{B}$ Γ Δ *v b*)
then show *?case* **using** *wfB-elims wfX-wfB1* **by** *metis*
next
case (*wfE-plusI* Θ Φ $\mathcal{B}$ Γ Δ *v1 v2*)
then show *?case* **using** *wfB-elims wfX-wfB1* **by** *metis*
next
case (*wfE-fstI* Θ Φ Γ Δ *v1 b1 b2*)
then show *?case* **using** *wfB-elims wfX-wfB1* **by** *metis*
next
case (*wfE-sndI* Θ Φ Γ Δ *v1 b1 b2*)
then show *?case* **using** *wfB-elims wfX-wfB1* **by** *metis*
next
case (*wfE-concatI* Θ Φ $\mathcal{B}$ Γ Δ *v1 v2*)
then show *?case* **using** *wfB-elims wfX-wfB1* **by** *metis*
next
case (*wfE-splitI* Θ Φ $\mathcal{B}$ Γ Δ *v1 v2*)
then show *?case* **using** *wfB-elims wfX-wfB1*
using *wfB-pairI* **by** *auto*
next
case (*wfE-lenI* Θ Φ $\mathcal{B}$ Γ Δ *v1*)
then show *?case* **using** *wfB-elims wfX-wfB1*
using *wfB-intI wfX-wfY1*(*1*) **by** *auto*
next
case (*wfE-appI* Θ Φ $\mathcal{B}$ Γ Δ *f x b c* τ *s v*)
hence Θ ; $\mathcal{B}$;(*x*,*b*,*c*) $\#_\Gamma$ *GNil* $\vdash_{wf}$ τ **using** *wfPhi-f-simple-wfT wfT-b-weakening* **by** *fast*
then show *?case* **using** *b-of.simps* **using** *wfT-b-weakening*
by (*metis b-of.cases bot.extremum wfT-elims*(*2*))
next
case (*wfE-appPI* Θ Φ $\mathcal{B}$ Γ Δ *b'* *bv v* τ *f x b c s*)
hence Θ ; {| *bv* |} ;(*x*,*b*,*c*) $\#_\Gamma$ *GNil* $\vdash_{wf}$ τ **using** *wfPhi-f-poly-wfT wfX-wfY* **by** *blast*
then show *?case* **using** *wfE-appPI b-of.simps* **using** *wfT-b-weakening wfT-elims wfT-subst-wfB subst-b-b-def* **by** *metis*

next
case (*wfE-mvarI* Θ Φ $\mathcal{B}$ Γ Δ u τ)
hence Θ ; $\mathcal{B}$; $\Gamma \vdash_{wf} \tau$ **using** *wfD-wfT* **by** *fast*
then show *?case* **using** *wfT-elims b-of.simps* **by** *metis*
next
case (*wfFTNone* Θ *ft*)
then show *?case* **by** *auto*
next
case (*wfFTSome* Θ *bv ft*)
then show *?case* **by** *auto*
next
case (*wfS-valI* Θ Φ $\mathcal{B}$ Γ *v b* Δ)
then show *?case* **using** *wfX-wfB1* **by** *auto*
next
case (*wfS-letI* Θ Φ $\mathcal{B}$ Γ Δ *e b' x s b*)
then show *?case* **using** *wfX-wfB1* **by** *auto*
next
case (*wfS-let2I* Θ Φ $\mathcal{B}$ Γ Δ *s1* τ *x s2 b*)
then show *?case* **using** *wfX-wfB1* **by** *auto*
next
case (*wfS-ifI* Θ $\mathcal{B}$ Γ *v* Φ Δ *s1 b s2*)
then show *?case* **using** *wfX-wfB1* **by** *auto*
next
case (*wfS-varI* Θ $\mathcal{B}$ Γ τ *v u* Φ Δ *b s*)
then show *?case* **using** *wfX-wfB1* **by** *auto*
next
case (*wfS-assignI u* τ Δ Θ $\mathcal{B}$ Γ Φ *v*)
then show *?case* **using** *wfX-wfB1*
using *wfB-unitI wfX-wfY2*(*5*) **by** *auto*
next
case (*wfS-whileI* Θ Φ $\mathcal{B}$ Γ Δ *s1 s2 b*)
then show *?case* **using** *wfX-wfB1* **by** *auto*
next
case (*wfS-seqI* Θ Φ $\mathcal{B}$ Γ Δ *s1 s2 b*)
then show *?case* **using** *wfX-wfB1* **by** *auto*
next
case (*wfS-matchI* Θ $\mathcal{B}$ Γ *v tid dclist* Δ Φ *cs b*)
then show *?case* **using** *wfX-wfB1* **by** *auto*
next
case (*wfS-branchI* Θ Φ $\mathcal{B}$ *x* τ Γ Δ *s b tid dc*)
then show *?case* **using** *wfX-wfB1* **by** *auto*
next
case (*wfS-finalI* Θ Φ $\mathcal{B}$ Γ Δ *tid dc t cs b*)
then show *?case* **using** *wfX-wfB1* **by** *auto*
next
case (*wfS-cons* Θ Φ $\mathcal{B}$ Γ Δ *tid dc t cs b dclist css*)
then show *?case* **using** *wfX-wfB1* **by** *auto*
next
case (*wfD-emptyI* Θ $\mathcal{B}$ Γ)
then show *?case* **using** *wfX-wfB1* **by** *auto*
next
case (*wfD-cons* Θ $\mathcal{B}$ Γ Δ τ *u*)

then show *?case* **using** *wfX-wfB1* **by** *auto*
next
case (*wfPhi-emptyI* Θ)
then show *?case* **using** *wfX-wfB1* **by** *auto*
next
case (*wfPhi-consI f* Θ Φ *ft*)
then show *?case* **using** *wfX-wfB1* **by** *auto*
next
case (*wfFTI* Θ *B b* Φ *x c s* τ)
then show *?case* **using** *wfX-wfB1*
by (*meson Wellformed.wfFTI wb-b-weakening2*(*8*))
qed(*metis wfV-wf wfG-wf wf-intros wfX-wfB1*)

lemmas *wfX-wfB* = *wfX-wfB1 wfX-wfB2*

lemma *wf-weakening1*:
fixes Γ::Γ **and** Γ'::Γ **and** *v*::*v* **and** *e*::*e* **and** *c*::*c* **and** τ::τ **and** *ts*::(*string*$*\tau$) *list* **and** Δ::Δ **and** *s*::*s* **and** $\mathcal{B}$::$\mathcal{B}$ **and** *ftq*::*fun-typ-q* **and** *ft*::*fun-typ* **and** *ce*::*ce* **and** *td*::*type-def*
and *cs*::*branch-s* **and** *css*::*branch-list*

shows *wfV-weakening*: Θ ; $\mathcal{B}$; Γ $\vdash_{wf}$ *v* : *b* $\Longrightarrow$ Θ ; $\mathcal{B}$ $\vdash_{wf}$ Γ' $\Longrightarrow$ *setG* Γ $\subseteq$ *setG* Γ' $\Longrightarrow$ Θ ; $\mathcal{B}$; Γ' $\vdash_{wf}$ *v* : *b* **and**
wfC-weakening: Θ ; $\mathcal{B}$; Γ $\vdash_{wf}$ *c* $\Longrightarrow$ Θ ; $\mathcal{B}$ $\vdash_{wf}$ Γ' $\Longrightarrow$ *setG* Γ $\subseteq$ *setG* Γ' $\Longrightarrow$ Θ ; $\mathcal{B}$; Γ' $\vdash_{wf}$ *c* **and**
Θ ; $\mathcal{B}$ $\vdash_{wf}$ Γ $\Longrightarrow$ *True* **and**
wfT-weakening: Θ ; $\mathcal{B}$; Γ $\vdash_{wf}$ τ $\Longrightarrow$ Θ ; $\mathcal{B}$ $\vdash_{wf}$ Γ' $\Longrightarrow$ *setG* Γ $\subseteq$ *setG* Γ' $\Longrightarrow$ Θ ; $\mathcal{B}$; Γ' $\vdash_{wf}$ τ **and**
Θ ; $\mathcal{B}$; Γ $\vdash_{wf}$ *ts* $\Longrightarrow$ *True* **and**
$\vdash_{wf}$ *P* $\Longrightarrow$ *True* **and**
wfB-weakening: *wfB* Θ $\mathcal{B}$ *b* $\Longrightarrow$ $\mathcal{B}$ |$\subseteq$| $\mathcal{B}'$ $\Longrightarrow$ *wfB* Θ $\mathcal{B}$ *b* **and**
wfCE-weakening: Θ ; $\mathcal{B}$; Γ $\vdash_{wf}$ *ce* : *b* $\Longrightarrow$ Θ ; $\mathcal{B}$ $\vdash_{wf}$ Γ' $\Longrightarrow$ *setG* Γ $\subseteq$ *setG* Γ' $\Longrightarrow$ Θ ; $\mathcal{B}$; Γ' $\vdash_{wf}$ *ce* : *b* **and**
Θ $\vdash_{wf}$ *td* $\Longrightarrow$ *True*
proof(*nominal-induct*
b **and** *c* **and** Γ **and** τ **and** *ts* **and** *P* **and** *b* **and** *b* **and** *td*
avoiding: Γ'
rule:*wfV-wfC-wfG-wfT-wfTs-wfTh-wfB-wfCE-wfTD.strong-induct*)
case (*wfV-varI* Θ $\mathcal{B}$ Γ *b c x*)
hence *Some* (*b*, *c*) = *lookup* Γ' *x* **using** *lookup-weakening* **by** *metis*
then show *?case* **using** *Wellformed.wfV-varI wfV-varI* **by** *metis*
next
case (*wfTI z* Θ $\mathcal{B}$ Γ *b c*)
show *?case* **proof**
show ‹*atom z* $\sharp$ (Θ, $\mathcal{B}$, Γ')› **using** *wfTI* **by** *auto*
show ‹ Θ ; $\mathcal{B}$ $\vdash_{wf}$ *b* › **using** *wfTI* **by** *auto*
have *:*setG* ((*z*, *b*, *TRUE*) #$_\Gamma$ Γ) $\subseteq$ *setG* ((*z*, *b*, *TRUE*) #$_\Gamma$ Γ') **using** *setG.simps wfTI* **by** *auto*
thus ‹ Θ ; $\mathcal{B}$; (*z*, *b*, *TRUE*) #$_\Gamma$ Γ' $\vdash_{wf}$ *c* › **using** *wfTI*(*8*)[*OF* - *] *wfTI wfX-wfY*
by (*simp add*: *wfG-cons-TRUE*)
qed
next
case (*wfV-conspI s bv dclist* Θ *dc x b'* *c* $\mathcal{B}$ *b* Γ *v*)

show *?case* **proof**
show ‹*AF-typedef-poly s bv dclist* ∈ *set* Θ› **using** *wfV-conspI* **by** *auto*
show ‹(*dc*, ⟦ *x* : *b′* | *c* ⟧) ∈ *set dclist*› **using** *wfV-conspI* **by** *auto*
show ‹ Θ ; $\mathcal{B}$ ⊢$_{wf}$ *b* › **using** *wfV-conspI* **by** *auto*
show ‹*atom bv* ♯ (Θ, $\mathcal{B}$, Γ′, *b*, *v*)› **using** *wfV-conspI* **by** *simp*
show ‹ Θ ; $\mathcal{B}$; Γ′ ⊢$_{wf}$ *v* : *b′*[*bv*::=*b*]$_{bb}$ › **using** *wfV-conspI* **by** *auto*
qed

qed(*metis wf-intros*)+

lemma *wf-weakening2*:
fixes Γ::Γ **and** Γ′::Γ **and** *v*::*v* **and** *e*::*e* **and** *c*::*c* **and** τ::τ **and** *ts*::(*string*∗τ) *list* **and** Δ::Δ **and** *s*::*s* **and** $\mathcal{B}$::$\mathcal{B}$ **and** *ftq*::*fun-typ-q* **and** *ft*::*fun-typ* **and** *ce*::*ce* **and** *td*::*type-def*
and *cs*::*branch-s* **and** *css*::*branch-list*
shows
wfE-weakening: Θ ; Φ ; $\mathcal{B}$; Γ ; Δ ⊢$_{wf}$ *e* : *b* ⟹ Θ ; $\mathcal{B}$ ⊢$_{wf}$ Γ′ ⟹ *setG* Γ ⊆ *setG* Γ′ ⟹ Θ ; Φ ; $\mathcal{B}$; Γ′ ; Δ ⊢$_{wf}$ *e* : *b* **and**
wfS-weakening: Θ ; Φ ; $\mathcal{B}$; Γ ; Δ ⊢$_{wf}$ *s* : *b* ⟹ Θ ; $\mathcal{B}$ ⊢$_{wf}$ Γ′ ⟹ *setG* Γ ⊆ *setG* Γ′ ⟹ Θ ; Φ ; $\mathcal{B}$; Γ′ ; Δ ⊢$_{wf}$ *s* : *b* **and**
Θ ; Φ ; $\mathcal{B}$; Γ ; Δ ; *tid* ; *dc* ; *t* ⊢$_{wf}$ *cs* : *b* ⟹ Θ ; $\mathcal{B}$ ⊢$_{wf}$ Γ′ ⟹ *setG* Γ ⊆ *setG* Γ′ ⟹ Θ ; Φ ; $\mathcal{B}$; Γ′ ; Δ ; *tid* ; *dc* ; *t* ⊢$_{wf}$ *cs* : *b* **and**
Θ ; Φ ; $\mathcal{B}$; Γ ; Δ ; *tid* ; *dclist* ⊢$_{wf}$ *css* : *b* ⟹ Θ ; $\mathcal{B}$ ⊢$_{wf}$ Γ′ ⟹ *setG* Γ ⊆ *setG* Γ′ ⟹ Θ ; Φ ; $\mathcal{B}$; Γ′ ; Δ ; *tid* ; *dclist* ⊢$_{wf}$ *css* : *b* **and**
Θ ⊢$_{wf}$ (Φ::Φ) ⟹ *True* **and**
wfD-weakning: Θ ; $\mathcal{B}$; Γ ⊢$_{wf}$ Δ ⟹ Θ ; $\mathcal{B}$ ⊢$_{wf}$ Γ′ ⟹ *setG* Γ ⊆ *setG* Γ′ ⟹ Θ ; $\mathcal{B}$; Γ′ ⊢$_{wf}$ Δ **and**
Θ ; Φ ⊢$_{wf}$ *ftq* ⟹ *True* **and**
Θ ; Φ ; $\mathcal{B}$ ⊢$_{wf}$ *ft* ⟹ *True*
proof(*nominal-induct*
b **and** *b* **and** *b* **and** *b* **and** Φ **and** Δ **and** *ftq* **and** *ft*
avoiding: Γ′
rule:*wfE-wfS-wfCS-wfCSS-wfPhi-wfD-wfFTQ-wfFT.strong-induct*)

case (*wfE-appPI* Θ Φ $\mathcal{B}$ Γ Δ *b′ bv v* τ *f x b c s*)
show *?case* **proof**
show ‹ Θ ⊢$_{wf}$ Φ › **using** *wfE-appPI* **by** *auto*
show ‹ Θ ; $\mathcal{B}$; Γ′ ⊢$_{wf}$ Δ › **using** *wfE-appPI* **by** *auto*
show ‹ Θ ; $\mathcal{B}$ ⊢$_{wf}$ *b′* › **using** *wfE-appPI* **by** *auto*
show ‹*atom bv* ♯ (Φ, Θ, $\mathcal{B}$, Γ′, Δ, *b′*, *v*, (*b-of* τ)[*bv*::=*b′*]$_b$)› **using** *wfE-appPI* **by** *auto*
show ‹*Some* (*AF-fundef f* (*AF-fun-typ-some bv* (*AF-fun-typ x b c* τ *s*))) = *lookup-fun* Φ *f*› **using** *wfE-appPI* **by** *auto*
show ‹ Θ ; $\mathcal{B}$; Γ′ ⊢$_{wf}$ *v* : *b*[*bv*::=*b′*]$_b$ › **using** *wfE-appPI wf-weakening1* **by** *auto*
qed
next
case (*wfS-letI* Θ Φ $\mathcal{B}$ Γ Δ *e b′ x s b*)
show *?case* **proof**(*rule*)
show ‹ Θ ; Φ ; $\mathcal{B}$; Γ′ ; Δ ⊢$_{wf}$ *e* : *b′* › **using** *wfS-letI* **by** *auto*
have *setG* ((*x*, *b′*, *TRUE*) #$_\Gamma$ Γ) ⊆ *setG* ((*x*, *b′*, *TRUE*) #$_\Gamma$ Γ′) **using** *wfS-letI* **by** *auto*
thus ‹ Θ ; Φ ; $\mathcal{B}$; (*x*, *b′*, *TRUE*) #$_\Gamma$ Γ′ ; Δ ⊢$_{wf}$ *s* : *b* › **using** *wfS-letI* **by** (*meson wfG-cons wfG-cons-TRUE wfS-wf*)
show ‹ Θ ; $\mathcal{B}$; Γ′ ⊢$_{wf}$ Δ › **using** *wfS-letI* **by** *auto*
show ‹*atom x* ♯ (Φ, Θ, $\mathcal{B}$, Γ′, Δ, *e*, *b*)› **using** *wfS-letI* **by** *auto*

qed
next
case (*wfS-let2I* Θ Φ $\mathcal{B}$ Γ Δ *s1* τ *x s2 b*)
show *?case* **proof**
show ‹ Θ ; Φ ; $\mathcal{B}$; Γ' ; $\Delta \vdash_{wf}$ *s1* : *b-of* τ › **using** *wfS-let2I* **by** *auto*
show ‹ Θ ; $\mathcal{B}$; Γ' $\vdash_{wf}$ τ › **using** *wfS-let2I wf-weakening1* **by** *auto*
have *setG* ((x, *b-of* τ, *TRUE*) $\#_\Gamma$ Γ) $\subseteq$ *setG* ((x, *b-of* τ, *TRUE*) $\#_\Gamma$ Γ') **using** *wfS-let2I* **by** *auto*
thus ‹ Θ ; Φ ; $\mathcal{B}$; (x, *b-of* τ, *TRUE*) $\#_\Gamma$ Γ' ; $\Delta \vdash_{wf}$ *s2* : *b* › **using** *wfS-let2I* **by** (*meson wfG-cons wfG-cons-TRUE wfS-wf*)
show ‹*atom x* ♯ (Φ, Θ, $\mathcal{B}$, Γ', Δ, *s1*, *b*, τ)› **using** *wfS-let2I* **by** *auto*
qed
next
case (*wfS-varI* Θ $\mathcal{B}$ Γ τ *v u* Φ Δ *b s*)
show *?case* **proof**
show Θ ; $\mathcal{B}$; $\Gamma' \vdash_{wf} \tau$ **using** *wfS-varI wf-weakening1* **by** *auto*
show Θ ; $\mathcal{B}$; $\Gamma' \vdash_{wf}$ *v* : *b-of* τ **using** *wfS-varI wf-weakening1* **by** *auto*
show *atom u* ♯ (Φ, Θ, $\mathcal{B}$, Γ', Δ, τ, *v*, *b*) **using** *wfS-varI* **by** *auto*
show Θ ; Φ ; $\mathcal{B}$; Γ' ; (u, τ) $\#_\Delta$ Δ $\vdash_{wf}$ s : b **using** *wfS-varI* **by** *auto*
qed
next
case (*wfS-branchI* Θ Φ $\mathcal{B}$ x τ Γ Δ s *b tid dc*)
show *?case* **proof**
have *setG* ((x, *b-of* τ, *TRUE*) $\#_\Gamma$ Γ) $\subseteq$ *setG* ((x, *b-of* τ, *TRUE*) $\#_\Gamma$ Γ') **using** *wfS-branchI* **by** *auto*
thus ‹ Θ ; Φ ; $\mathcal{B}$; (x, *b-of* τ, *TRUE*) $\#_\Gamma$ Γ' ; $\Delta \vdash_{wf}$ s : b › **using** *wfS-branchI* **by** (*meson wfG-cons wfG-cons-TRUE wfS-wf*)
show ‹*atom x* ♯ (Φ, Θ, $\mathcal{B}$, Γ', Δ, Γ', τ)› **using** *wfS-branchI* **by** *auto*
show ‹ Θ ; $\mathcal{B}$; $\Gamma' \vdash_{wf}$ Δ › **using** *wfS-branchI* **by** *auto*
qed
next
case (*wfS-finalI* Θ Φ $\mathcal{B}$ Γ Δ *tid dclist' cs b dclist*)
then show *?case* **using** *wf-intros* **by** *metis*
next
case (*wfS-cons* Θ Φ $\mathcal{B}$ Γ Δ *tid dclist' cs b css dclist*)
then show *?case* **using** *wf-intros* **by** *metis*
next
case (*wfS-assertI* Θ Φ $\mathcal{B}$ x c Γ Δ s *b*)
show *?case* **proof**(*rule*)
show ‹ Θ ; $\mathcal{B}$; Γ' $\vdash_{wf}$ c › **using** *wfS-assertI wf-weakening1* **by** *auto*
have Θ ; $\mathcal{B}$ $\vdash_{wf}$ (x, *B-bool*, c) $\#_\Gamma$ Γ' **proof**(*rule wfG-consI*)
show ‹ Θ ; $\mathcal{B}$ $\vdash_{wf}$ Γ' › **using** *wfS-assertI* **by** *auto*
show ‹*atom x* ♯ Γ'› **using** *wfS-assertI* **by** *auto*
show ‹ Θ ; $\mathcal{B}$ $\vdash_{wf}$ *B-bool* › **using** *wfS-assertI wfB-boolI wfX-wfY* **by** *metis*
have Θ ; $\mathcal{B}$ $\vdash_{wf}$ (x, *B-bool*, *TRUE*) $\#_\Gamma$ Γ' **proof**
show (*TRUE*) $\in$ {*TRUE*, *FALSE*} **by** *auto*
show ‹ Θ ; $\mathcal{B}$ $\vdash_{wf}$ Γ' › **using** *wfS-assertI* **by** *auto*
show ‹*atom x* ♯ Γ'› **using** *wfS-assertI* **by** *auto*
show ‹ Θ ; $\mathcal{B}$ $\vdash_{wf}$ *B-bool* › **using** *wfS-assertI wfB-boolI wfX-wfY* **by** *metis*
qed
thus ‹ Θ ; $\mathcal{B}$; (x, *B-bool*, *TRUE*) $\#_\Gamma$ Γ' $\vdash_{wf}$ c ›
using *wf-weakening1*(*2*)[*OF* ‹ Θ ; $\mathcal{B}$; Γ' $\vdash_{wf}$ c › ‹ Θ ; $\mathcal{B}$ $\vdash_{wf}$ (x, *B-bool*, *TRUE*) $\#_\Gamma$ Γ' ›] **by**

force
qed

thus ‹ Θ ; Φ ; $\mathcal{B}$; (x, *B-bool*, c) $\#_\Gamma$ Γ' ; $\Delta \vdash_{wf} s : b$ › **using** *wfS-assertI* **by** *fastforce*

show ‹ Θ ; $\mathcal{B}$; $\Gamma' \vdash_{wf} \Delta$ › **using** *wfS-assertI* **by** *auto*
show ‹*atom* $x \sharp$ (Φ, Θ, $\mathcal{B}$, Γ', Δ, c, b, s)› **using** *wfS-assertI* **by** *auto*
qed

qed(*metis wf-intros wf-weakening1*)+

lemmas *wf-weakening* = *wf-weakening1 wf-weakening2*

lemma *wfV-weakening-cons*:
fixes Γ::Γ **and** Γ'::Γ **and** v::v **and** c::c
assumes Θ ; $\mathcal{B}$; $\Gamma \vdash_{wf} v : b$ **and** *atom* $y \sharp \Gamma$ **and** Θ ; $\mathcal{B}$; (y,b',*TRUE*) $\#_\Gamma$ Γ) $\vdash_{wf} c$
shows Θ ; $\mathcal{B}$; (y,b',c) $\#_\Gamma\Gamma \vdash_{wf} v : b$
proof −
have *wfG* Θ $\mathcal{B}$ ((y,b',c) $\#_\Gamma\Gamma$) **using** *wfG-intros2 assms* **by** *auto*
moreover have *setG* $\Gamma \subseteq$ *setG* ((y,b',c) $\#_\Gamma\Gamma$) **using** *setG.simps* **by** *auto*
ultimately show *?thesis* **using** *wf-weakening* **using** *assms*(*1*) **by** *blast*
qed

lemma *wfG-cons-weakening*:
fixes Γ'::Γ
assumes Θ ; $\mathcal{B} \vdash_{wf}$ ((x, b, c) $\#_\Gamma$ Γ) **and** Θ ; $\mathcal{B} \vdash_{wf} \Gamma'$ **and** *setG* $\Gamma \subseteq$ *setG* Γ' **and** *atom* $x \sharp \Gamma'$
shows Θ ; $\mathcal{B} \vdash_{wf}$ ((x, b, c) $\#_\Gamma$ Γ')
proof(*cases* $c \in \{$*TRUE*,*FALSE*$\}$)
case *True*
then show *?thesis* **using** *wfG-wfB wfG-cons2I assms* **by** *auto*
next
case *False*
hence $*$:Θ ; $\mathcal{B} \vdash_{wf} \Gamma \wedge$ *atom* $x \sharp \Gamma \wedge$ Θ ; $\mathcal{B}$; (x, b, *TRUE*) $\#_\Gamma$ $\Gamma \vdash_{wf} c$
using *wfG-elims*(*2*)[*OF assms*(*1*)] **by** *auto*
have *a1*:Θ ; $\mathcal{B} \vdash_{wf}$ (x, b, *TRUE*) $\#_\Gamma$ Γ' **using** *wfG-wfB wfG-cons2I assms* **by** *simp*
moreover have *a2*:*setG* ((x, b, *TRUE*) $\#_\Gamma$ Γ) $\subseteq$ *setG* ((x, b, *TRUE*) $\#_\Gamma$ Γ') **using** *setG.simps assms* **by** *blast*
moreover have Θ ; $\mathcal{B} \vdash_{wf}$ (x, b, *TRUE*) $\#_\Gamma$ Γ' **proof**
show (*TRUE*) $\in$ {*TRUE*, *FALSE*} **by** *auto*
show Θ ; $\mathcal{B} \vdash_{wf} \Gamma'$ **using** *assms* **by** *auto*
show *atom* $x \sharp \Gamma'$ **using** *assms* **by** *auto*
show Θ ; $\mathcal{B} \vdash_{wf} b$ **using** *assms wfG-elims* **by** *metis*
qed
hence Θ ; $\mathcal{B}$; (x, b, *TRUE*) $\#_\Gamma$ $\Gamma' \vdash_{wf} c$ **using** *wf-weakening a1 a2* $*$ **by** *auto*
then show *?thesis* **using** *wfG-cons1I*[*of c* Θ $\mathcal{B}$ Γ' x b, *OF False*] *wfG-wfB assms* **by** *simp*
qed

lemma *wfT-weakening-aux*:
fixes Γ::Γ **and** Γ'::Γ **and** c::c
assumes Θ ; $\mathcal{B}$; $\Gamma \vdash_{wf} \{\!|\, z : b \mid c \,|\!\}$ **and** Θ ; $\mathcal{B} \vdash_{wf} \Gamma'$ **and** *setG* $\Gamma \subseteq$ *setG* Γ' **and** *atom* $z \sharp \Gamma'$
shows Θ ; $\mathcal{B}$; $\Gamma' \vdash_{wf} \{\!|\, z : b \mid c \,|\!\}$
proof

show ‹*atom z* ♯ (Θ, $\mathcal{B}$, Γ′)›
using *wf-supp wfX-wfY assms fresh-prodN fresh-def x-not-in-b-set wfG-fresh-x* **by** *metis*
show ‹ Θ ; $\mathcal{B}$ $\vdash_{wf}$ *b* › **using** *assms wfT-elims* **by** *metis*
show ‹ Θ ; $\mathcal{B}$; (*z*, *b*, *TRUE*) $\#_\Gamma$ Γ′ $\vdash_{wf}$ *c* › **proof** −
have ∗:Θ ; $\mathcal{B}$; (*z*,*b*,*TRUE*) $\#_\Gamma$Γ $\vdash_{wf}$ *c* **using** *wfT-wfC fresh-weakening assms* **by** *auto*
moreover have *a1*:Θ ; $\mathcal{B}$ $\vdash_{wf}$ (*z*, *b*, *TRUE*) $\#_\Gamma$ Γ′ **using** *wfG-cons2I assms* ‹Θ ; $\mathcal{B}$ $\vdash_{wf}$ *b*› **by** *simp*
moreover have *a2*:*setG* ((*z*, *b*, *TRUE*) $\#_\Gamma$ Γ) ⊆ *setG* ((*z*, *b*, *TRUE*) $\#_\Gamma$ Γ′) **using** *setG.simps assms* **by** *blast*
moreover have Θ ; $\mathcal{B}$ $\vdash_{wf}$ (*z*, *b*, *TRUE*) $\#_\Gamma$ Γ′ **proof**
show (*TRUE*) ∈ {*TRUE*, *FALSE*} **by** *auto*
show Θ ; $\mathcal{B}$ $\vdash_{wf}$ Γ′ **using** *assms* **by** *auto*
show *atom z* ♯ Γ′ **using** *assms* **by** *auto*
show Θ ; $\mathcal{B}$ $\vdash_{wf}$ *b* **using** *assms wfT-elims* **by** *metis*
qed
thus *?thesis* **using** *wf-weakening a1 a2* ∗ **by** *auto*
qed
qed

lemma *wfT-weakening-all*:
fixes Γ::Γ **and** Γ′::Γ **and** τ::τ
assumes Θ ; $\mathcal{B}$; Γ $\vdash_{wf}$ τ **and** Θ ; $\mathcal{B}'$ $\vdash_{wf}$ Γ′ **and** *setG* Γ ⊆ *setG* Γ′ **and** $\mathcal{B}$ |⊆| $\mathcal{B}'$
shows Θ ; $\mathcal{B}'$; Γ′ $\vdash_{wf}$ τ
using *wb-b-weakening assms wfT-weakening* **by** *metis*

lemma *wfT-weakening-nil*:
fixes Γ::Γ **and** Γ′::Γ **and** τ::τ
assumes Θ ; {||} ; *GNil* $\vdash_{wf}$ τ **and** Θ ; $\mathcal{B}'$ $\vdash_{wf}$ Γ′
shows Θ ; $\mathcal{B}'$; Γ′ $\vdash_{wf}$ τ
using *wfT-weakening-all*
using *assms*(*1*) *assms*(*2*) *setG.simps*(*1*) **by** *blast*

lemma *dc-t-closed*:
fixes *x*::*x* **and** *v*::*v* **and** τ::τ **and** *G*::Γ
assumes *wfTh* Θ **and** *AF-typedef s dclist* ∈ *set* Θ **and**
(*dc*, τ) ∈ *set dclist* **and** Θ ; *B* $\vdash_{wf}$ *G*
shows *supp* τ = {} **and** τ[*x*::=*v*]$_{\tau v}$ = τ **and** *wfT* Θ *B G* τ
proof −
show *supp* τ = {} **proof**(*rule ccontr*)
assume *a1*: *supp* τ ≠ {}
have *supp* Θ ≠ {} **proof** −
obtain *dclist* **where** *dc*: *AF-typedef s dclist* ∈ *set* Θ ∧ (*dc*, τ) ∈ *set dclist*
using *assms* **by** *auto*
hence *supp* (*dc*,τ) ≠ {}
using *a1* **by** (*simp add*: *supp-Pair*)
hence *supp dclist* ≠ {} **using** *dc supp-list-member* **by** *auto*
hence *supp* (*AF-typedef s dclist*) ≠ {} **using** *type-def.supp* **by** *auto*
thus *?thesis* **using** *supp-list-member dc* **by** *auto*
qed

thus *False* **using** *assms wfTh-supp* **by** *simp*
qed
thus $\tau[x::=v]_{\tau v} = \tau$ **by** (*simp add*: *fresh-def*)
have *wfT* Θ {||} *GNil* τ **using** *assms wfTh-wfT* **by** *auto*
thus *wfT* Θ *B G* τ **using** *assms wfT-weakening-nil* **by** *simp*

qed

lemma *u-fresh-d*:
assumes *atom u* $\sharp$ *D*
shows $u \notin$ *fst* ‘ *setD D*
using *assms* **proof**(*induct D rule*: Δ-*induct*)
case *DNil*
then show *?case* **by** *auto*
next
case (*DCons u′ t′* Δ')
then show *?case* **unfolding** *setD.simps*
using *fresh-DCons fresh-Pair* **by** (*simp add*: *fresh-Pair fresh-at-base*(*2*))
qed

lemma *wf-d-weakening*:
fixes Γ::Γ **and** Γ'::Γ **and** *v*::*v* **and** *e*::*e* **and** *c*::*c* **and** τ::τ **and** *ts*::(*string*$*\tau$) *list* **and** Δ::Δ **and** *s*::*s* **and** $\mathcal{B}$::$\mathcal{B}$ **and** *ftq*::*fun-typ-q* **and** *ft*::*fun-typ* **and** *ce*::*ce* **and** *td*::*type-def*
and *cs*::*branch-s* **and** *css*::*branch-list*
shows
Θ ; Φ ; $\mathcal{B}$; Γ ; $\Delta \vdash_{wf} e : b \Longrightarrow \Theta$; $\mathcal{B}$; $\Gamma \vdash_{wf} \Delta' \Longrightarrow$ *setD* $\Delta \subseteq$ *setD* $\Delta' \Longrightarrow \Theta$; Φ ; $\mathcal{B}$; Γ ; $\Delta' \vdash_{wf} e : b$ **and**
Θ ; Φ ; $\mathcal{B}$; Γ ; $\Delta \vdash_{wf} s : b \Longrightarrow \Theta$; $\mathcal{B}$; $\Gamma \vdash_{wf} \Delta' \Longrightarrow$ *setD* $\Delta \subseteq$ *setD* $\Delta' \Longrightarrow \Theta$; Φ ; $\mathcal{B}$; Γ ; $\Delta' \vdash_{wf} s : b$ **and**
Θ ; Φ ; $\mathcal{B}$; Γ ; Δ ; *tid* ; *dc* ; *t* $\vdash_{wf}$ *cs* : *b* $\Longrightarrow \Theta$; $\mathcal{B}$; $\Gamma \vdash_{wf} \Delta' \Longrightarrow$ *setD* $\Delta \subseteq$ *setD* $\Delta' \Longrightarrow$ Θ ; Φ ; $\mathcal{B}$; Γ ; Δ' ; *tid* ; *dc* ; *t* $\vdash_{wf}$ *cs* : *b* **and**
Θ ; Φ ; $\mathcal{B}$; Γ ; Δ ; *tid* ; *dclist* $\vdash_{wf}$ *css* : *b* $\Longrightarrow \Theta$; $\mathcal{B}$; $\Gamma \vdash_{wf} \Delta' \Longrightarrow$ *setD* $\Delta \subseteq$ *setD* $\Delta' \Longrightarrow$ Θ ; Φ ; $\mathcal{B}$; Γ ; Δ' ; *tid* ; *dclist* $\vdash_{wf}$ *css* : *b* **and**
$\Theta \vdash_{wf} (\Phi::\Phi) \Longrightarrow$ *True* **and**
Θ ; $\mathcal{B}$; $\Gamma \vdash_{wf} \Delta \Longrightarrow$ *True* **and**
Θ ; $\Phi \vdash_{wf}$ *ftq* $\Longrightarrow$ *True* **and**
Θ ; Φ ; $\mathcal{B} \vdash_{wf}$ *ft* $\Longrightarrow$ *True*
proof(*nominal-induct*
b **and** *b* **and** *b* **and** *b* **and** Φ **and** Δ **and** *ftq* **and** *ft*
avoiding: Δ'
rule:*wfE-wfS-wfCS-wfCSS-wfPhi-wfD-wfFTQ-wfFT.strong-induct*)
case (*wfE-valI* Θ Φ $\mathcal{B}$ Γ Δ *v b*)
then show *?case* **using** *wf-intros* **by** *metis*
next
case (*wfE-plusI* Θ Φ $\mathcal{B}$ Γ Δ *v1 v2*)
then show *?case* **using** *wf-intros* **by** *metis*
next
case (*wfE-leqI* Θ Φ $\mathcal{B}$ Γ Δ *v1 v2*)
then show *?case* **using** *wf-intros* **by** *metis*
next
case (*wfE-fstI* Θ Φ $\mathcal{B}$ Γ Δ *v1 b1 b2*)

then show *?case* **using** *wf-intros* **by** *metis*
next
case (*wfE-sndI* Θ Φ $\mathcal{B}$ Γ Δ *v1 b1 b2*)
then show *?case* **using** *wf-intros* **by** *metis*
next
case (*wfE-concatI* Θ Φ $\mathcal{B}$ Γ Δ *v1 v2*)
then show *?case* **using** *wf-intros* **by** *metis*
next
case (*wfE-splitI* Θ Φ $\mathcal{B}$ Γ Δ *v1 v2*)
then show *?case* **using** *wf-intros* **by** *metis*
next
case (*wfE-lenI* Θ Φ $\mathcal{B}$ Γ Δ *v1*)
then show *?case* **using** *wf-intros* **by** *metis*
next
case (*wfE-appI* Θ Φ $\mathcal{B}$ Γ Δ *f x b c* τ *s v*)
then show *?case* **using** *wf-intros* **by** *metis*
next
case (*wfE-appPI* Θ Φ $\mathcal{B}$ Γ Δ *b' bv v* τ *f x b c s*)
show *?case* **proof**(*rule*, (*rule wfE-appPI*)+)
show ‹*atom bv* $\sharp$ (Φ, Θ, $\mathcal{B}$, Γ, Δ', *b'*, *v*, (*b-of* τ)[*bv*::=*b'*]$_b$)› **using** *wfE-appPI* **by** *auto*
show ‹*Some* (*AF-fundef f* (*AF-fun-typ-some bv* (*AF-fun-typ x b c* τ *s*))) = *lookup-fun* Φ *f*› **using** *wfE-appPI* **by** *auto*
show ‹ Θ ; $\mathcal{B}$; Γ $\vdash_{wf}$ *v* : *b*[*bv*::=*b'*]$_b$ › **using** *wfE-appPI* **by** *auto*
qed
next
case (*wfE-mvarI* Θ Φ $\mathcal{B}$ Γ Δ *u* τ)
show *?case* **proof**
show ‹ Θ $\vdash_{wf}$ Φ › **using** *wfE-mvarI* **by** *auto*
show ‹ Θ ; $\mathcal{B}$; Γ $\vdash_{wf}$ Δ' › **using** *wfE-mvarI* **by** *auto*
show ‹(*u*, τ) $\in$ *setD* Δ'› **using** *wfE-mvarI* **by** *auto*
qed
next
case (*wfS-valI* Θ Φ $\mathcal{B}$ Γ *v b* Δ)
then show *?case* **using** *wf-intros* **by** *metis*
next
case (*wfS-letI* Θ Φ $\mathcal{B}$ Γ Δ *e b' x s b*)
show *?case* **proof**(*rule*)
show ‹ Θ ; Φ ; $\mathcal{B}$; Γ ; Δ' $\vdash_{wf}$ *e* : *b'* › **using** *wfS-letI* **by** *auto*
have Θ ; $\mathcal{B}$ $\vdash_{wf}$ (*x*, *b'*, *TRUE*) $\#_\Gamma$ Γ **using** *wfG-cons2I wfX-wfY wfS-letI* **by** *metis*
hence Θ ; $\mathcal{B}$; (*x*, *b'*, *TRUE*) $\#_\Gamma$ Γ $\vdash_{wf}$ Δ' **using** *wf-weakening2*(*6*) *wfS-letI* **by** *force*
thus ‹ Θ ; Φ ; $\mathcal{B}$; (*x*, *b'*, *TRUE*) $\#_\Gamma$ Γ ; Δ' $\vdash_{wf}$ *s* : *b* › **using** *wfS-letI* **by** *metis*
show ‹ Θ ; $\mathcal{B}$; Γ $\vdash_{wf}$ Δ' › **using** *wfS-letI* **by** *auto*
show ‹*atom x* $\sharp$ (Φ, Θ, $\mathcal{B}$, Γ, Δ', *e*, *b*)› **using** *wfS-letI* **by** *auto*
qed
next
case (*wfS-assertI* Θ Φ $\mathcal{B}$ *x c* Γ Δ *s b*)
show *?case* **proof**
have Θ ; $\mathcal{B}$; (*x*, *B-bool*, *c*) $\#_\Gamma$ Γ $\vdash_{wf}$ Δ' **proof**(*rule wf-weakening2*(*6*))
show ‹ Θ ; $\mathcal{B}$; Γ $\vdash_{wf}$ Δ' › **using** *wfS-assertI* **by** *auto*
next
show ‹ Θ ; $\mathcal{B}$ $\vdash_{wf}$ (*x*, *B-bool*, *c*) $\#_\Gamma$ Γ › **using** *wfS-assertI wfX-wfY* **by** *metis*
next

show ‹*setG* Γ ⊆ *setG* ((*x*, *B-bool*, *c*) #Γ Γ)› **using** *wfS-assertI* **by** *auto*
qed
thus ‹ Θ ; Φ ; $\mathcal{B}$; (*x*, *B-bool*, *c*) #Γ Γ ; Δ′ ⊢*wf* *s* : *b* › **using** *wfS-assertI wfX-wfY* **by** *metis*
next
show ‹ Θ ; $\mathcal{B}$; Γ ⊢*wf* *c* › **using** *wfS-assertI* **by** *auto*
next
show ‹ Θ ; $\mathcal{B}$; Γ ⊢*wf* Δ′ › **using** *wfS-assertI* **by** *auto*
next
show ‹*atom x* ♯ (Φ, Θ, $\mathcal{B}$, Γ, Δ′, *c*, *b*, *s*)› **using** *wfS-assertI* **by** *auto*
qed
next
case (*wfS-let2I* Θ Φ $\mathcal{B}$ Γ Δ *s1* τ *x s2 b*)
show *?case* **proof**
show ‹ Θ ; Φ ; $\mathcal{B}$; Γ ; Δ′ ⊢*wf* *s1* : *b-of* τ › **using** *wfS-let2I* **by** *auto*
show ‹ Θ ; $\mathcal{B}$; Γ ⊢*wf* τ › **using** *wfS-let2I* **by** *auto*
have Θ ; $\mathcal{B}$ ⊢*wf* (*x*, *b-of* τ, *TRUE*) #Γ Γ **using** *wfG-cons2I wfX-wfY wfS-let2I* **by** *metis*
hence Θ ; $\mathcal{B}$; (*x*, *b-of* τ, *TRUE*) #Γ Γ ⊢*wf* Δ′ **using** *wf-weakening2*(*6*) *wfS-let2I* **by** *force*
thus ‹ Θ ; Φ ; $\mathcal{B}$; (*x*, *b-of* τ, *TRUE*) #Γ Γ ; Δ′ ⊢*wf* *s2* : *b* › **using** *wfS-let2I* **by** *metis*
show ‹*atom x* ♯ (Φ, Θ, $\mathcal{B}$, Γ, Δ′, *s1*, *b*,τ)› **using** *wfS-let2I* **by** *auto*
qed
next
case (*wfS-ifI* Θ $\mathcal{B}$ Γ *v* Φ Δ *s1 b s2*)
then show *?case* **using** *wf-intros* **by** *metis*
next
case (*wfS-varI* Θ $\mathcal{B}$ Γ τ *v u* Φ Δ *b s*)
show *?case* **proof**
show ‹ Θ ; $\mathcal{B}$; Γ ⊢*wf* τ › **using** *wfS-varI* **by** *auto*
show ‹ Θ ; $\mathcal{B}$; Γ ⊢*wf* *v* : *b-of* τ › **using** *wfS-varI* **by** *auto*
show ‹*atom u* ♯ (Φ, Θ, $\mathcal{B}$, Γ, Δ′, τ, *v*, *b*)› **using** *wfS-varI setD.simps* **by** *auto*
have Θ ; $\mathcal{B}$; Γ ⊢*wf* (*u*, τ) #Δ Δ′ **using** *wfS-varI wfD-cons setD.simps u-fresh-d* **by** *metis*
thus ‹ Θ ; Φ ; $\mathcal{B}$; Γ ; (*u*, τ) #Δ Δ′ ⊢*wf* *s* : *b* › **using** *wfS-varI setD.simps* **by** *blast*
qed
next
case (*wfS-assignI* *u* τ Δ Θ $\mathcal{B}$ Γ Φ *v*)
show *?case* **proof**
show ‹(*u*, τ) ∈ *setD* Δ′› **using** *wfS-assignI setD.simps* **by** *auto*
show ‹ Θ ; $\mathcal{B}$; Γ ⊢*wf* Δ′ › **using** *wfS-assignI* **by** *auto*
show ‹ Θ ⊢*wf* Φ › **using** *wfS-assignI* **by** *auto*
show ‹ Θ ; $\mathcal{B}$; Γ ⊢*wf* *v* : *b-of* τ › **using** *wfS-assignI* **by** *auto*
qed
next
case (*wfS-whileI* Θ Φ $\mathcal{B}$ Γ Δ *s1 s2 b*)
then show *?case* **using** *wf-intros* **by** *metis*
next
case (*wfS-seqI* Θ Φ $\mathcal{B}$ Γ Δ *s1 s2 b*)
then show *?case* **using** *wf-intros* **by** *metis*
next
case (*wfS-matchI* Θ $\mathcal{B}$ Γ *v tid dclist* Δ Φ *cs b*)
then show *?case* **using** *wf-intros* **by** *metis*
next
case (*wfS-branchI* Θ Φ $\mathcal{B}$ *x* τ Γ Δ *s b tid dc*)
show *?case* **proof**

have Θ ; $\mathcal{B}$ $\vdash_{wf}$ $(x,\ b\text{-}of\ \tau,\ TRUE)$ $\#_\Gamma$ Γ **using** *wfG-cons2I wfX-wfY wfS-branchI* **by** *metis*
hence Θ ; $\mathcal{B}$; $(x,\ b\text{-}of\ \tau,\ TRUE)$ $\#_\Gamma$ $\Gamma \vdash_{wf} \Delta'$ **using** *wf-weakening2(6) wfS-branchI* **by** *force*
thus ‹ Θ ; Φ ; $\mathcal{B}$; $(x,\ b\text{-}of\ \tau,\ TRUE)$ $\#_\Gamma$ Γ ; $\Delta' \vdash_{wf} s : b$ › **using** *wfS-branchI* **by** *simp*
show ‹ *atom* $x \sharp (\Phi, \Theta, \mathcal{B}, \Gamma, \Delta', \Gamma, \tau)$› **using** *wfS-branchI* **by** *auto*
show ‹ Θ ; $\mathcal{B}$; $\Gamma \vdash_{wf} \Delta'$ › **using** *wfS-branchI* **by** *auto*
qed
next
case (*wfS-finalI* Θ Φ $\mathcal{B}$ Γ Δ *tid dclist' cs b dclist*)
then show *?case* **using** *wf-intros* **by** *metis*
next
case (*wfS-cons* Θ Φ $\mathcal{B}$ Γ Δ *tid dclist' cs b css dclist*)
then show *?case* **using** *wf-intros* **by** *metis*
qed(*auto*+)

8.15 Forms

Well-formedness for particular constructs that we will need later

lemma *wfC-e-eq*:
fixes *ce*::*ce* **and** Γ::Γ
assumes Θ ; $\mathcal{B}$; $\Gamma \vdash_{wf} ce : b$ **and** *atom* $x \sharp \Gamma$
shows Θ ; $\mathcal{B}$; $((x,\ b,\ TRUE)\ \#_\Gamma\ \Gamma) \vdash_{wf} (CE\text{-}val\ (V\text{-}var\ x)\ ==\ ce)$
proof −
have Θ ; $\mathcal{B} \vdash_{wf} b$ **using** *assms wfX-wfB* **by** *auto*
hence *wbg*: Θ ; $\mathcal{B} \vdash_{wf} \Gamma$ **using** *wfX-wfY assms* **by** *auto*
show *?thesis* **proof**
show ∗:Θ ; $\mathcal{B}$; $(x,\ b,\ TRUE)$ $\#_\Gamma$ $\Gamma \vdash_{wf} CE\text{-}val\ (V\text{-}var\ x) : b$
proof(*rule*)
show Θ ; $\mathcal{B}$; $(x,\ b,\ TRUE)$ $\#_\Gamma$ $\Gamma \vdash_{wf} V\text{-}var\ x : b$ **proof**
show Θ ; $\mathcal{B} \vdash_{wf} (x,\ b,\ TRUE)$ $\#_\Gamma$ Γ **using** *wfG-cons2I wfX-wfY assms* ‹Θ ; $\mathcal{B} \vdash_{wf} b$› **by** *auto*
show *Some* $(b,\ TRUE) = lookup\ ((x,\ b,\ TRUE)\ \#_\Gamma\ \Gamma)\ x$ **using** *lookup.simps* **by** *auto*
qed
qed
show Θ ; $\mathcal{B}$; $(x,\ b,\ TRUE)$ $\#_\Gamma$ $\Gamma \vdash_{wf} ce : b$
using *assms wf-weakening1(8)*[*OF assms(1)*, *of* $(x,\ b,\ TRUE)$ $\#_\Gamma$ Γ] ∗ *setG.simps wfX-wfY*
by (*metis Un-subset-iff equalityE*)
qed
qed

lemma *wfC-e-eq2*:
fixes *e1*::*ce* **and** *e2*::*ce*
assumes Θ ; $\mathcal{B}$; $\Gamma \vdash_{wf} e1 : b$ **and** Θ ; $\mathcal{B}$; $\Gamma \vdash_{wf} e2 : b$ **and** $\vdash_{wf} \Theta$ **and** *atom* $x \sharp \Gamma$
shows Θ ; $\mathcal{B}$; $(x,\ b,\ (CE\text{-}val\ (V\text{-}var\ x)) == e1)$ $\#_\Gamma$ $\Gamma \vdash_{wf} (CE\text{-}val\ (V\text{-}var\ x)) == e2$
proof(*rule wfC-eqI*)
have ∗: Θ ; $\mathcal{B} \vdash_{wf} (x,\ b,\ CE\text{-}val\ (V\text{-}var\ x)\ ==\ e1)$ $\#_\Gamma$ Γ **proof**(*rule wfG-cons1I*)
show $(CE\text{-}val\ (V\text{-}var\ x)\ ==\ e1) \notin \{TRUE, FALSE\}$ **by** *auto*
show Θ ; $\mathcal{B} \vdash_{wf} \Gamma$ **using** *assms wfX-wfY* **by** *metis*
show ∗:*atom* $x \sharp \Gamma$ **using** *assms* **by** *auto*
show Θ ; $\mathcal{B}$; $(x,\ b,\ TRUE)$ $\#_\Gamma$ $\Gamma \vdash_{wf} CE\text{-}val\ (V\text{-}var\ x)\ ==\ e1$ **using** *wfC-e-eq assms* ∗ **by** *auto*
show Θ ; $\mathcal{B} \vdash_{wf} b$ **using** *assms wfX-wfB* **by** *auto*

qed
show Θ ; $\mathcal{B}$; (x, b, *CE-val* (*V-var* x) == *e1*) $\#_\Gamma$ Γ $\vdash_{wf}$ *CE-val* (*V-var* x) : b **using** *assms* $*$ *wfCE-valI wfV-varI* **by** *auto*
show Θ ; $\mathcal{B}$; (x, b, *CE-val* (*V-var* x) == *e1*) $\#_\Gamma$ Γ $\vdash_{wf}$ *e2* : b **proof**(*rule wf-weakening1*(*8*))
show Θ ; $\mathcal{B}$; Γ $\vdash_{wf}$ *e2* : b **using** *assms* **by** *auto*
show Θ ; $\mathcal{B}$ $\vdash_{wf}$ (x, b, *CE-val* (*V-var* x) == *e1*) $\#_\Gamma$ Γ **using** $*$ **by** *auto*
show *setG* Γ $\subseteq$ *setG* ((x, b, *CE-val* (*V-var* x) == *e1*) $\#_\Gamma$ Γ) **by** *auto*
qed
qed

lemma *wfT-wfT-if-rev*:
assumes *wfV P* $\mathcal{B}$ Γ v (*base-for-lit l*) **and** *wfT P* $\mathcal{B}$ Γ t **and** ‹*atom z1* $\sharp$ Γ›
shows *wfT P* $\mathcal{B}$ Γ ({| *z1* : *b-of t* | *CE-val v* == *CE-val* (*V-lit l*) *IMP* (*c-of t z1*) |})
proof
show ‹ *P* ; $\mathcal{B}$ $\vdash_{wf}$ *b-of t* › **using** *wfX-wfY assms* **by** *meson*
have *wfg*: *P* ; $\mathcal{B}$ $\vdash_{wf}$ (*z1*, *b-of t*, *TRUE*) $\#_\Gamma$ Γ **using** *assms wfV-wf wfG-cons2I wfX-wfY*
by (*meson wfG-cons-TRUE*)
show ‹ *P* ; $\mathcal{B}$; (*z1*, *b-of t*, *TRUE*) $\#_\Gamma$ Γ $\vdash_{wf}$ [v]^{ce} == [[l]^{v}]^{ce} *IMP* *c-of t z1* › **proof**
show $*$: ‹ *P* ; $\mathcal{B}$; (*z1*, *b-of t*, *TRUE*) $\#_\Gamma$ Γ $\vdash_{wf}$ [v]^{ce} == [[l]^{v}]^{ce} ›
proof(*rule wfC-eqI*[**where** *b*=*base-for-lit l*])
show *P* ; $\mathcal{B}$; (*z1*, *b-of t*, *TRUE*) $\#_\Gamma$ Γ $\vdash_{wf}$ [v]^{ce} : *base-for-lit l*
using *assms wf-intros wf-weakening wfg* **by** (*meson wfV-weakening-cons*)
show *P* ; $\mathcal{B}$; (*z1*, *b-of t*, *TRUE*) $\#_\Gamma$ Γ $\vdash_{wf}$ [[l]^{v}]^{ce} : *base-for-lit l* **using** *wfg assms wf-intros wf-weakening wfV-weakening-cons* **by** *meson*
qed
have t = {| *z1* : *b-of t* | *c-of t z1* |} **using** *c-of-eq*
using *assms*(*2*) *assms*(*3*) *b-of-c-of-eq wfT-x-fresh* **by** *auto*
thus ‹ *P* ; $\mathcal{B}$; (*z1*, *b-of t*, *TRUE*) $\#_\Gamma$ Γ $\vdash_{wf}$ *c-of t z1* › **using** *wfT-wfC assms wfG-elims* $*$ **by** *simp*
qed
show ‹*atom z1* $\sharp$ (*P*, $\mathcal{B}$, Γ)› **using** *assms wfG-fresh-x wfX-wfY* **by** *metis*
qed

lemma *wfT-eq-imp*:
fixes *zz*::x **and** *ll*::l **and** τ'::τ
assumes *base-for-lit ll* = *B-bool* **and** Θ ; {||} ; *GNil* $\vdash_{wf}$ τ' **and**
Θ ; {||} $\vdash_{wf}$ (x, *b-of* {| z' : *B-bool* | *TRUE* |}, *c-of* {| z' : *B-bool* | *TRUE* |} x) $\#_\Gamma$ *GNil* **and** *atom zz* $\sharp$ x
shows Θ ; {||} ; (x, *b-of* {| z' : *B-bool* | *TRUE* |}, *c-of* {| z' : *B-bool* | *TRUE* |} x) $\#_\Gamma$
GNil $\vdash_{wf}$ {| *zz* : *b-of* τ' | [[x]^{v}]^{ce} == [[*ll*]^{v}]^{ce} *IMP* *c-of* τ' *zz* |}
proof(*rule wfT-wfT-if-rev*)
show ‹ Θ ; {||} ; (x, *b-of* {| z' : *B-bool* | *TRUE* |}, *c-of* {| z' : *B-bool* | *TRUE* |} x) $\#_\Gamma$ *GNil* $\vdash_{wf}$ [x]^{v} : *base-for-lit ll* ›
using *wfV-varI lookup.simps base-for-lit.simps assms* **by** *simp*
show ‹ Θ ; {||} ; (x, *b-of* {| z' : *B-bool* | *TRUE* |}, *c-of* {| z' : *B-bool* | *TRUE* |} x) $\#_\Gamma$ *GNil* $\vdash_{wf}$ τ' ›
using *wf-weakening assms setG.simps* **by** *auto*
show ‹*atom zz* $\sharp$ (x, *b-of* {| z' : *B-bool* | *TRUE* |}, *c-of* {| z' : *B-bool* | *TRUE* |} x) $\#_\Gamma$ *GNil*›
unfolding *fresh-GCons fresh-prod3 b-of.simps c-of-true*
using *x-fresh-b fresh-GNil c-of-true c.fresh assms* **by** *metis*
qed

lemma *wfC-v-eq*:
fixes *ce*::*ce* **and** Γ::Γ **and** *v*::*v*
assumes Θ ; $\mathcal{B}$; Γ $\vdash_{wf}$ *v* : *b* **and** *atom x* ♯ Γ
shows Θ ; $\mathcal{B}$; ((*x*, *b*, *TRUE*) $\#_\Gamma$ Γ) $\vdash_{wf}$ (*CE-val* (*V-var x*) == *CE-val v*)
using *wfC-e-eq wfCE-valI assms wfX-wfY* **by** *auto*

lemma *wfT-e-eq*:
fixes *ce*::*ce*
assumes Θ ; $\mathcal{B}$; Γ $\vdash_{wf}$ *ce* : *b* **and** *atom z* ♯ Γ
shows Θ ; $\mathcal{B}$; Γ $\vdash_{wf}$ {| *z* : *b* | *CE-val* (*V-var z*) == *ce* |}
proof
show Θ ; $\mathcal{B}$ $\vdash_{wf}$ *b* **using** *wfX-wfB assms* **by** *auto*
show *atom z* ♯ (Θ, $\mathcal{B}$, Γ) **using** *assms wfG-fresh-x wfX-wfY* **by** *metis*
show Θ ; $\mathcal{B}$; (*z*, *b*, *TRUE*) $\#_\Gamma$ Γ $\vdash_{wf}$ *CE-val* (*V-var z*) == *ce*
using *wfTI wfC-e-eq assms wfTI* **by** *auto*
qed

lemma *wfT-v-eq*:
assumes *wfB* Θ $\mathcal{B}$ *b* **and** *wfV* Θ $\mathcal{B}$ Γ *v b* **and** *atom z* ♯ Γ
shows *wfT* Θ $\mathcal{B}$ Γ {| *z* : *b* | *C-eq* (*CE-val* (*V-var z*)) (*CE-val v*)|}
using *wfT-e-eq wfE-valI assms wfX-wfY*
by (*simp add*: *wfCE-valI*)

lemma *wfC-wfG*:
fixes Γ::Γ **and** *c*::*c* **and** *b*::*b*
assumes Θ ; *B* ; Γ $\vdash_{wf}$ *c* **and** Θ ; *B* $\vdash_{wf}$ *b* **and** *atom x* ♯ Γ
shows Θ ; *B* $\vdash_{wf}$ (*x*,*b*,*c*)$\#_\Gamma$ Γ
proof −
have Θ ; *B* $\vdash_{wf}$ (*x*, *b*, *TRUE*) $\#_\Gamma$ Γ **using** *wfG-cons2I assms wfX-wfY* **by** *fast*
hence Θ ; *B* ; (*x*, *b*, *TRUE*) $\#_\Gamma$ Γ $\vdash_{wf}$ *c* **using** *wfC-weakening assms* **by** *force*
thus *?thesis* **using** *wfG-consI assms wfX-wfY* **by** *metis*
qed

8.16 Replacing

lemma *wfG-cons-fresh2*:
fixes Γ′::Γ
assumes *wfG P* $\mathcal{B}$ (((*x*′,*b*′,*c*′) $\#_\Gamma$ Γ′ @ (*x*, *b*, *c*) $\#_\Gamma$ Γ))
shows *x*′≠*x*
proof −
have *atom x*′ ♯ (Γ′ @ (*x*, *b*, *c*) $\#_\Gamma$ Γ)
using *assms wfG-elims*(*2*) **by** *blast*
thus *?thesis*
using *fresh-gamma-append*[*of atom x*′ Γ′ (*x*, *b*, *c*) $\#_\Gamma$ Γ] *fresh-GCons fresh-prod3*[*of atom x*′ *x b c*] **by** *auto*
qed

lemma *replace-in-g-inside*:
fixes Γ::Γ
assumes *wfG P* $\mathcal{B}$ (Γ′@((*x*,*b0*,*c0*′) $\#_\Gamma$Γ))
shows *replace-in-g* (Γ′@((*x*,*b0*,*c0*′) $\#_\Gamma$Γ)) *x c0* = (Γ′@((*x*,*b0*,*c0*) $\#_\Gamma$Γ))

using *assms* **proof**(*induct* Γ' *rule*: Γ-*induct*)
case *GNil*
then show *?case* **using** *replace-in-g.simps* **by** *auto*
next
case (*GCons* x' b' c' Γ'')
hence P ; $\mathcal{B} \vdash_{wf} ((x', b', c') \ \#_\Gamma \ (\Gamma''@ \ (x, \ b0, \ c0') \ \#_\Gamma \ \Gamma))$ **by** *simp*
hence $x \neq x'$ **using** *wfG-cons-fresh2* **by** *metis*
then show *?case* **using** *replace-in-g.simps GCons* **by** (*simp add*: *wfG-cons*)
qed

lemma *wfG-supp-rig-eq*:
fixes Γ::Γ
assumes *wfG* P $\mathcal{B}$ $(\Gamma'' \ @ \ (x, \ b0, \ c0) \ \#_\Gamma \ \Gamma)$ **and** *wfG* P $\mathcal{B}$ $(\Gamma'' \ @ \ (x, \ b0, \ c0') \ \#_\Gamma \ \Gamma)$
shows *supp* $(\Gamma'' \ @ \ (x, \ b0, \ c0') \ \#_\Gamma \ \Gamma) \cup$ *supp* $\mathcal{B}$ = *supp* $(\Gamma'' \ @ \ (x, \ b0, \ c0) \ \#_\Gamma \ \Gamma) \cup$ *supp* $\mathcal{B}$
using *assms* **proof**(*induct* Γ'')
case *GNil*
have *supp* $(GNil \ @ \ (x, \ b0, \ c0') \ \#_\Gamma \ \Gamma) \cup$ *supp* $\mathcal{B}$ = *supp* $((x, \ b0, \ c0') \ \#_\Gamma \ \Gamma) \cup$ *supp* $\mathcal{B}$ **using** *supp-Cons supp-GNil* **by** *auto*
also have ... = *supp* $x \cup$ *supp* $b0 \cup$ *supp* $c0' \cup$ *supp* $\Gamma \cup$ *supp* $\mathcal{B}$ **using** *supp-GCons* **by** *auto*
also have ... = *supp* $x \cup$ *supp* $b0 \cup$ *supp* $c0 \cup$ *supp* $\Gamma \cup$ *supp* $\mathcal{B}$ **using** *GNil wfG-wfC*[*THEN wfC-supp-cons*(*2*)] **by** *fastforce*
also have ... = (*supp* $((x, \ b0, \ c0) \ \#_\Gamma \ \Gamma)) \cup$ *supp* $\mathcal{B}$ **using** *supp-GCons* **by** *auto*
finally have *supp* $(GNil \ @ \ (x, \ b0, \ c0') \ \#_\Gamma \ \Gamma) \cup$ *supp* $\mathcal{B}$ = *supp* $(GNil \ @ \ (x, \ b0, \ c0) \ \#_\Gamma \ \Gamma) \cup$ *supp* $\mathcal{B}$ **using** *supp-Cons supp-GNil* **by** *auto*
then show *?case* **using** *supp-GCons wfG-cons2* **by** *auto*
next
case (*GCons xbc* $\Gamma 1$)
moreover have $(xbc \ \#_\Gamma \ \Gamma 1) \ @ \ (x, \ b0, \ c0) \ \#_\Gamma \ \Gamma = (xbc \ \#_\Gamma \ (\Gamma 1 \ @ \ (x, \ b0, \ c0) \ \#_\Gamma \ \Gamma))$ **by** *simp*
moreover have $(xbc \ \#_\Gamma \ \Gamma 1) \ @ \ (x, \ b0, \ c0') \ \#_\Gamma \ \Gamma = (xbc \ \#_\Gamma \ (\Gamma 1 \ @ \ (x, \ b0, \ c0') \ \#_\Gamma \ \Gamma))$ **by** *simp*
ultimately have $(P \ ; \mathcal{B} \ \vdash_{wf} \Gamma 1 \ @ \ ((x, \ b0, \ c0) \ \#_\Gamma \ \Gamma)) \ \wedge \ P \ ; \mathcal{B} \ \vdash_{wf} \Gamma 1 \ @ \ ((x, \ b0, \ c0') \ \#_\Gamma \ \Gamma)$ **using** *wfG-cons2* **by** *metis*
thus *?case* **using** *GCons supp-GCons* **by** *auto*
qed

lemma *fresh-replace-inside*[*ms-fresh*]:
fixes y::x **and** Γ::Γ
assumes *wfG* P $\mathcal{B}$ $(\Gamma'' \ @ \ (x, \ b, \ c) \ \#_\Gamma \ \Gamma)$ **and** *wfG* P $\mathcal{B}$ $(\Gamma'' \ @ \ (x, \ b, \ c') \ \#_\Gamma \ \Gamma)$
shows *atom* $y \sharp (\Gamma'' \ @ \ (x, \ b, \ c) \ \#_\Gamma \ \Gamma) =$ *atom* $y \sharp (\Gamma'' \ @ \ (x, \ b, \ c') \ \#_\Gamma \ \Gamma)$
unfolding *fresh-def* **using** *wfG-supp-rig-eq assms x-not-in-b-set* **by** *fast*

lemma *wf-replace-inside1*:
fixes Γ::Γ **and** Φ::Φ **and** Θ::Θ **and** Γ'::Γ **and** v::v **and** e::e **and** c::c **and** c''::c **and** c'::c **and** τ::τ **and** ts::(*string*$*\tau$) *list* **and** Δ::Δ **and** b'::b **and** b::b **and** s::s
and *ftq*::*fun-typ-q* **and** *ft*::*fun-typ* **and** *ce*::*ce* **and** *td*::*type-def* **and** *cs*::*branch-s* **and** *css*::*branch-list*

shows *wfV-replace-inside*: Θ ; $\mathcal{B}$; $G \vdash_{wf} v : b' \Longrightarrow G = (\Gamma' \ @ \ (x, \ b, \ c') \ \#_\Gamma \ \Gamma) \Longrightarrow \Theta$; $\mathcal{B}$; $((x,b,TRUE) \ \#_\Gamma \Gamma) \vdash_{wf} c \Longrightarrow \Theta$; $\mathcal{B}$; $(\Gamma' \ @ \ (x, \ b, \ c) \ \#_\Gamma \ \Gamma) \vdash_{wf} v : b'$ **and**
wfC-replace-inside: Θ ; $\mathcal{B}$; $G \vdash_{wf} c'' \Longrightarrow G = (\Gamma' \ @ \ (x, \ b, \ c') \ \#_\Gamma \ \Gamma) \Longrightarrow \Theta$; $\mathcal{B}$; $((x,b,TRUE) \ \#_\Gamma \Gamma) \vdash_{wf} c \Longrightarrow \Theta$; $\mathcal{B}$; $(\Gamma' \ @ \ (x, \ b, \ c) \ \#_\Gamma \ \Gamma) \vdash_{wf} c''$ **and**

wfG-replace-inside: $\Theta\ ;\ \mathcal{B} \vdash_{wf} G \Longrightarrow G = (\Gamma' @ (x,\ b,\ c')\ \#_\Gamma\ \Gamma) \Longrightarrow \Theta\ ;\ \mathcal{B}\ ;\ ((x,b,TRUE)\ \#_\Gamma\Gamma) \vdash_{wf} c \Longrightarrow \Theta\ ;\ \mathcal{B} \vdash_{wf} (\Gamma' @ (x,\ b,\ c)\ \#_\Gamma\ \Gamma)$ **and**
wfT-replace-inside: $\Theta\ ;\ \mathcal{B}\ ;\ G \vdash_{wf} \tau \Longrightarrow G = (\Gamma' @ (x,\ b,\ c')\ \#_\Gamma\ \Gamma) \Longrightarrow \Theta\ ;\ \mathcal{B}\ ;\ ((x,b,TRUE)\ \#_\Gamma\Gamma) \vdash_{wf} c \Longrightarrow \Theta\ ;\ \mathcal{B}\ ;\ (\Gamma' @ (x,\ b,\ c)\ \#_\Gamma\ \Gamma) \vdash_{wf} \tau$ **and**
$\Theta\ ;\ \mathcal{B}\ ;\ \Gamma \vdash_{wf} ts \Longrightarrow True$ **and**
$\vdash_{wf} P \Longrightarrow True$ **and**
$\Theta\ ;\ \mathcal{B} \vdash_{wf} b \Longrightarrow True$ **and**
wfCE-replace-inside: $\Theta\ ;\ \mathcal{B}\ ;\ G \vdash_{wf} ce : b' \Longrightarrow G = (\Gamma' @ (x,\ b,\ c')\ \#_\Gamma\ \Gamma) \Longrightarrow \Theta\ ;\ \mathcal{B}\ ;\ ((x,b,TRUE)\ \#_\Gamma\Gamma) \vdash_{wf} c \Longrightarrow \Theta\ ;\ \mathcal{B}\ ;\ (\Gamma' @ (x,\ b,\ c)\ \#_\Gamma\ \Gamma) \vdash_{wf} ce : b'$ **and**
$\Theta \vdash_{wf} td \Longrightarrow True$

proof(*nominal-induct*
b' **and** c'' **and** G **and** τ **and** ts **and** P **and** b **and** b' **and** td
avoiding: $\Gamma'\ c'$
rule:*wfV-wfC-wfG-wfT-wfTs-wfTh-wfB-wfCE-wfTD.strong-induct*)
case (*wfV-varI* $\Theta\ \mathcal{B}\ \Gamma 2\ b2\ c2\ x2$)
then show *?case* **using** *wf-intros* **by** (*metis lookup-in-rig-eq lookup-in-rig-neq replace-in-g-inside*)
next
case (*wfV-conspI s bv dclist* Θ *dc x1 b' c1* $\mathcal{B}$ *b1* $\Gamma 1$ *v*)
show *?case* **proof**
show ‹*AF-typedef-poly s bv dclist* ∈ *set* Θ› **using** *wfV-conspI* **by** *auto*
show ‹(*dc*, {| *x1* : b' | *c1* |}) ∈ *set dclist*› **using** *wfV-conspI* **by** *auto*
show ‹ Θ ; $\mathcal{B}$ $\vdash_{wf}$ *b1* › **using** *wfV-conspI* **by** *auto*
show ∗: ‹ Θ ; $\mathcal{B}$; Γ' @ (x, b, c) $\#_\Gamma$ $\Gamma \vdash_{wf} v : b'[bv::=b1]_{bb}$ › **using** *wfV-conspI* **by** *auto*
moreover have $\Theta\ ;\ \mathcal{B} \vdash_{wf} \Gamma' @ (x,\ b,\ c')\ \#_\Gamma\ \Gamma$ **using** *wfV-wf wfV-conspI* **by** *simp*
ultimately have *atom bv* ♯ $\Gamma' @ (x,\ b,\ c)\ \#_\Gamma\ \Gamma$ **unfolding** *fresh-def* **using** *wfV-wf wfG-supp-rig-eq wfV-conspI*
by (*metis Un-iff fresh-def*)
thus ‹*atom bv* ♯ (Θ, $\mathcal{B}$, $\Gamma' @ (x,\ b,\ c)\ \#_\Gamma\ \Gamma$, *b1*, *v*)›
unfolding *fresh-prodN* **using** *fresh-prodN wfV-conspI* **by** *metis*
qed
next
case (*wfTI z* Θ $\mathcal{B}$ *G b1 c1*)
show *?case* **proof**
show ‹ Θ ; $\mathcal{B}$ $\vdash_{wf}$ *b1* › **using** *wfTI* **by** *auto*

have $\Theta\ ;\ \mathcal{B} \vdash_{wf} (x,\ b,\ c)\ \#_\Gamma\ \Gamma$ **using** *wfG-consI wfTI wfG-cons wfX-wfY* **by** *metis*
moreover hence ∗:*wfG* Θ $\mathcal{B}$ ($\Gamma' @ (x,\ b,\ c)\ \#_\Gamma\ \Gamma$) **using** *wfX-wfY*
by (*metis append-g.simps*(2) *wfG-cons2 wfTI.hyps wfTI.prems*(1) *wfTI.prems*(2))
hence ‹*atom z* ♯ $\Gamma' @ (x,\ b,\ c)\ \#_\Gamma\ \Gamma$›
using *fresh-replace-inside*[*of* Θ $\mathcal{B}$ Γ' *x b c* Γ c' *z*,*OF* ∗] *wfTI wfX-wfY wfG-elims* **by** *metis*
thus ‹*atom z* ♯ (Θ, $\mathcal{B}$, $\Gamma' @ (x,\ b,\ c)\ \#_\Gamma\ \Gamma$)› **using** *wfG-fresh-x*[*OF* ∗] **by** *auto*

have $(z,\ b1,\ TRUE)\ \#_\Gamma\ G = ((z,\ b1,\ TRUE)\ \#_\Gamma\ \Gamma') @ (x,\ b,\ c')\ \#_\Gamma\ \Gamma$
using *wfTI append-g.simps* **by** *metis*
thus ‹ Θ ; $\mathcal{B}$; $(z,\ b1,\ TRUE)\ \#_\Gamma\ \Gamma' @ (x,\ b,\ c)\ \#_\Gamma\ \Gamma \vdash_{wf} c1$ ›
using *wfTI*(9)[*OF - wfTI*(11)] **by** *fastforce*
qed
next
case (*wfG-nilI* Θ)
hence *GNil* = $(x,\ b,\ c')\ \#_\Gamma\ \Gamma$ **using** *append-g.simps* Γ.*distinct GNil-append* **by** *auto*
hence *False* **using** Γ.*distinct* **by** *auto*
then show *?case* **by** *auto*

next
case (*wfG-cons1I c1* Θ $\mathcal{B}$ *G x1 b1*)
show *?case* **proof**(*cases* Γ'=*GNil*)
case *True*
then show *?thesis* **using** *wfG-cons1I wfG-consI* **by** *auto*
next
case *False*
then obtain G'::Γ **where** $*$:(*x1*, *b1*, *c1*) $\#_\Gamma$ $G' = \Gamma'$ **using** *wfG-cons1I wfG-cons1I*(*7*) *GCons-eq-append-conv* **by** *auto*
hence $**$: $G = G'$ @ (*x*, *b*, c') $\#_\Gamma$ Γ **using** *wfG-cons1I* **by** *auto*
hence Θ ; $\mathcal{B}$ $\vdash_{wf}$ G' @ (*x*, *b*, *c*) $\#_\Gamma$ Γ **using** *wfG-cons1I* **by** *auto*
have Θ ; $\mathcal{B}$ $\vdash_{wf}$ (*x1*, *b1*, *c1*) $\#_\Gamma$ G' @ (*x*, *b*, *c*) $\#_\Gamma$ Γ **proof**(*rule Wellformed.wfG-cons1I*)
show *c1* $\notin$ {*TRUE*, *FALSE*} **using** *wfG-cons1I* **by** *auto*
show Θ ; $\mathcal{B}$ $\vdash_{wf}$ G' @ (*x*, *b*, *c*) $\#_\Gamma$ Γ **using** *wfG-cons1I*(*3*)[*of* G',*OF* $**$] *wfG-cons1I* **by** *auto*
show *atom x1* $\sharp$ G' @ (*x*, *b*, *c*) $\#_\Gamma$ Γ **using** *wfG-cons1I* $*$ $**$ *fresh-replace-inside* **by** *metis*
show Θ ; $\mathcal{B}$; (*x1*, *b1*, *TRUE*) $\#_\Gamma$ G' @ (*x*, *b*, *c*) $\#_\Gamma$ Γ $\vdash_{wf}$ *c1* **using** *wfG-cons1I*(*6*)[*of* (*x1*, *b1*, *TRUE*) $\#_\Gamma$ G'] *wfG-cons1I* $**$ **by** *auto*
show Θ ; $\mathcal{B}$ $\vdash_{wf}$ *b1* **using** *wfG-cons1I* **by** *auto*
qed
thus *?thesis* **using** $*$ **by** *auto*
qed
next
case (*wfG-cons2I c1* Θ $\mathcal{B}$ *G x1 b1*)
show *?case* **proof**(*cases* Γ'=*GNil*)
case *True*
then show *?thesis* **using** *wfG-cons2I wfG-consI* **by** *auto*
next
case *False*
then obtain G'::Γ **where** $*$:(*x1*, *b1*, *c1*) $\#_\Gamma$ $G' = \Gamma'$ **using** *wfG-cons2I GCons-eq-append-conv* **by** *auto*
hence $**$: $G = G'$ @ (*x*, *b*, c') $\#_\Gamma$ Γ **using** *wfG-cons2I* **by** *auto*
moreover have Θ ; $\mathcal{B}$ $\vdash_{wf}$ G' @ (*x*, *b*, *c*) $\#_\Gamma$ Γ **using** *wfG-cons2I* $*$ $**$ **by** *auto*
moreover hence *atom x1* $\sharp$ G' @ (*x*, *b*, *c*) $\#_\Gamma$ Γ **using** *wfG-cons2I* $*$ $**$ *fresh-replace-inside* **by** *metis*
ultimately show *?thesis* **using** *Wellformed.wfG-cons2I*[*OF wfG-cons2I*(*1*), *of* Θ $\mathcal{B}$ G'@ (*x*, *b*, *c*) $\#_\Gamma$ Γ *x1 b1*] *wfG-cons2I* $*$ $**$ **by** *auto*
qed
qed(*metis wf-intros*)+

lemma *wf-replace-inside2*:
fixes Γ::Γ **and** Φ::Φ **and** Θ::Θ **and** Γ'::Γ **and** *v*::*v* **and** *e*::*e* **and** *c*::*c* **and** c''::*c* **and** c'::*c* **and** τ::τ **and** *ts*::(*string*$*\tau$) *list* **and** Δ::Δ **and** b'::*b* **and** *b*::*b* **and** *s*::*s*
and *ftq*::*fun-typ-q* **and** *ft*::*fun-typ* **and** *ce*::*ce* **and** *td*::*type-def* **and** *cs*::*branch-s* **and** *css*::*branch-list*
shows
Θ ; Φ ; $\mathcal{B}$; G ; D $\vdash_{wf}$ e : b' $\Longrightarrow$ G = (Γ' @ (*x*, *b*, c') $\#_\Gamma$ Γ) $\Longrightarrow$ Θ ; $\mathcal{B}$; ((*x*,*b*,*TRUE*) $\#_\Gamma\Gamma$) $\vdash_{wf}$ *c* $\Longrightarrow$ Θ ; Φ ; $\mathcal{B}$; (Γ' @ (*x*, *b*, *c*) $\#_\Gamma$ Γ); D $\vdash_{wf}$ e : b' **and**
Θ ; Φ ; $\mathcal{B}$; Γ ; Δ $\vdash_{wf}$ s : b $\Longrightarrow$ *True* **and**
Θ ; Φ ; $\mathcal{B}$; Γ ; Δ ; *tid* ; *dc* ; t $\vdash_{wf}$ *cs* : b $\Longrightarrow$ *True* **and**
Θ ; Φ ; $\mathcal{B}$; Γ ; Δ ; *tid* ; *dclist* $\vdash_{wf}$ *css* : b $\Longrightarrow$ *True* **and**
Θ $\vdash_{wf}$ Φ $\Longrightarrow$ *True* **and**
Θ ; $\mathcal{B}$; G $\vdash_{wf}$ Δ $\Longrightarrow$ G = (Γ' @ (*x*, *b*, c') $\#_\Gamma$ Γ) $\Longrightarrow$ Θ ; $\mathcal{B}$; ((*x*,*b*,*TRUE*) $\#_\Gamma\Gamma$) $\vdash_{wf}$ *c* $\Longrightarrow$

Θ ; $\mathcal{B}$; $(\Gamma' @ (x, b, c) \ \#_\Gamma \ \Gamma) \vdash_{wf} \Delta$ **and**
Θ ; $\Phi \ \vdash_{wf} ftq \Longrightarrow True$ **and**
Θ ; Φ ; $\mathcal{B} \vdash_{wf} ft \Longrightarrow \ True$
proof(*nominal-induct*
b' **and** b **and** b **and** b **and** Φ **and** Δ **and** ftq **and** ft
avoiding: Γ' c'
rule:*wfE-wfS-wfCS-wfCSS-wfPhi-wfD-wfFTQ-wfFT.strong-induct*)
case (*wfE-valI* Θ Φ $\mathcal{B}$ Γ Δ v b)
then show *?case* **using** *wf-replace-inside1 Wellformed.wfE-valI* **by** *auto*
next
case (*wfE-plusI* Θ Φ $\mathcal{B}$ Γ Δ *v1 v2*)
then show *?case* **using** *wf-replace-inside1 Wellformed.wfE-plusI* **by** *auto*
next
case (*wfE-leqI* Θ Φ $\mathcal{B}$ Γ Δ *v1 v2*)
then show *?case* **using** *wf-replace-inside1 Wellformed.wfE-leqI* **by** *auto*
next
case (*wfE-fstI* Θ Φ $\mathcal{B}$ Γ Δ *v1 b1 b2*)
then show *?case* **using** *wf-replace-inside1 Wellformed.wfE-fstI* **by** *metis*
next
case (*wfE-sndI* Θ Φ $\mathcal{B}$ Γ Δ *v1 b1 b2*)
then show *?case* **using** *wf-replace-inside1 Wellformed.wfE-sndI* **by** *metis*
next
case (*wfE-concatI* Θ Φ $\mathcal{B}$ Γ Δ *v1 v2*)
then show *?case* **using** *wf-replace-inside1 Wellformed.wfE-concatI* **by** *auto*
next
case (*wfE-splitI* Θ Φ $\mathcal{B}$ Γ Δ *v1 v2*)
then show *?case* **using** *wf-replace-inside1 Wellformed.wfE-splitI* **by** *auto*
next
case (*wfE-lenI* Θ Φ $\mathcal{B}$ Γ Δ *v1*)
then show *?case* **using** *wf-replace-inside1 Wellformed.wfE-lenI* **by** *metis*
next
case (*wfE-appI* Θ Φ $\mathcal{B}$ Γ Δ f x b c τ s v)
then show *?case* **using** *wf-replace-inside1 Wellformed.wfE-appI* **by** *metis*
next
case (*wfE-appPI* Θ Φ $\mathcal{B}$ Γ'' Δ b' *bv v* τ *f x1 b1 c1 s*)
show *?case* **proof**
show ‹ $\Theta \ \vdash_{wf} \Phi$ › **using** *wfE-appPI* **by** *auto*
show ‹ Θ ; $\mathcal{B}$; $\Gamma' @ (x, b, c) \ \#_\Gamma \ \Gamma \vdash_{wf} \Delta$ › **using** *wfE-appPI* **by** *auto*
show ‹ Θ ; $\mathcal{B} \ \vdash_{wf} b'$ › **using** *wfE-appPI* **by** *auto*
show ∗:‹ Θ ; $\mathcal{B}$; $\Gamma' @ (x, b, c) \ \#_\Gamma \ \Gamma \vdash_{wf} v : b1[bv::=b']_b$ › **using** *wfE-appPI wf-replace-inside1* **by** *auto*

moreover have Θ ; $\mathcal{B} \ \vdash_{wf} \Gamma' @ (x, b, c') \ \#_\Gamma \ \Gamma$ **using** *wfV-wf wfE-appPI* **by** *metis*
ultimately have *atom bv* $\sharp$ $\Gamma' @ (x, b, c) \ \#_\Gamma \ \Gamma$
unfolding *fresh-def* **using** *wfV-wf wfG-supp-rig-eq wfE-appPI Un-iff fresh-def* **by** *metis*

thus ‹*atom bv* $\sharp$ (Φ, Θ, $\mathcal{B}$, $\Gamma' @ (x, b, c) \ \#_\Gamma \ \Gamma$, Δ, b', v, $(b\text{-}of\ \tau)[bv::=b']_b$)›
using *wfE-appPI fresh-prodN* **by** *metis*
show ‹*Some* (*AF-fundef f* (*AF-fun-typ-some bv* (*AF-fun-typ x1 b1 c1* τ *s*))) = *lookup-fun* Φ *f*›
using *wfE-appPI* **by** *auto*
qed
next

case (*wfE-mvarI* Θ Φ $\mathcal{B}$ Γ Δ u τ)
then show *?case* **using** *wf-replace-inside1 Wellformed.wfE-mvarI* **by** *metis*
next
case (*wfD-emptyI* Θ $\mathcal{B}$ Γ)
then show *?case* **using** *wf-replace-inside1 Wellformed.wfD-emptyI* **by** *metis*
next
case (*wfD-cons* Θ $\mathcal{B}$ Γ Δ τ u)
then show *?case* **using** *wf-replace-inside1 Wellformed.wfD-emptyI*
by (*simp add*: *wfE-wfS-wfCS-wfCSS-wfPhi-wfD-wfFTQ-wfFT.wfD-cons*)
next
case (*wfFTNone* Θ Φ *ft*)
then show *?case* **using** *wf-replace-inside1 Wellformed.wfD-emptyI* **by** *metis*
next
case (*wfFTSome* Θ Φ *bv ft*)
then show *?case* **using** *wf-replace-inside1 Wellformed.wfD-emptyI* **by** *metis*
qed(*auto*)

lemmas *wf-replace-inside* = *wf-replace-inside1 wf-replace-inside2*

lemma *wfC-replace-cons*:
assumes *wfG P* $\mathcal{B}$ ((*x*,*b*,*c1*) $\#_\Gamma\Gamma$) **and** *wfC P* $\mathcal{B}$ ((*x*,*b*,*TRUE*) $\#_\Gamma\Gamma$) *c2*
shows *wfC P* $\mathcal{B}$ ((*x*,*b*,*c1*) $\#_\Gamma\Gamma$) *c2*
proof −
have *wfC P* $\mathcal{B}$ (*GNil*@((*x*,*b*,*c1*) $\#_\Gamma\Gamma$)) *c2* **proof**(*rule wf-replace-inside1*(*2*))
show *P* ; $\mathcal{B}$; (*x*, *b*, *TRUE*) $\#_\Gamma$ Γ $\vdash_{wf}$ *c2* **using** *wfG-elim2 assms* **by** *auto*
show ‹(*x*, *b*, *TRUE*) $\#_\Gamma$ Γ = *GNil* @ (*x*, *b*, *TRUE*) $\#_\Gamma$ Γ› **using** *append-g.simps* **by** *auto*
show ‹*P* ; $\mathcal{B}$; (*x*, *b*, *TRUE*) $\#_\Gamma$ Γ $\vdash_{wf}$ *c1* › **using** *wfG-elim2 assms* **by** *auto*
qed
thus *?thesis* **using** *append-g.simps* **by** *auto*
qed

lemma *wfC-refl*:
assumes *wfG* Θ $\mathcal{B}$ ((*x*, *b′*, *c′*) $\#_\Gamma\Gamma$)
shows *wfC* Θ $\mathcal{B}$ ((*x*, *b′*, *c′*) $\#_\Gamma\Gamma$) *c′*
using *wfG-wfC assms wfC-replace-cons* **by** *auto*

lemma *wfG-wfC-inside*:
assumes (*x*, *b*, *c*) ∈ *setG G* **and** *wfG* Θ *B G*
shows *wfC* Θ *B G c*
using *assms* **proof**(*induct G rule*: Γ-*induct*)
case *GNil*
then show *?case* **by** *auto*
next
case (*GCons x′ b′ c′* Γ')
then consider (*hd*) (*x*, *b*, *c*) = (*x′*,*b′*,*c′*) | (*tail*) (*x*, *b*, *c*) ∈ *setG* Γ' **using** *setG.simps* **by** *auto*
then show *?case* **proof**(*cases*)
case *hd*
then show *?thesis* **using** *GCons wf-weakening*
by (*metis wfC-replace-cons wfG-cons-wfC*)
next
case *tail*
then show *?thesis* **using** *GCons wf-weakening*

by (*metis insert-iff insert-is-Un subsetI setG.simps*(2) *wfG-cons2*)
qed
qed

lemma *wfT-wf-cons3*:
assumes Θ ; $\mathcal{B}$; Γ $\vdash_{wf}$ $\{\!|$ *z* : *b* | *c* $|\!\}$ **and** *atom y* ♯ (*c*,Γ)
shows Θ ; $\mathcal{B}$ $\vdash_{wf}$ (*y*, *b*, *c*[*z*::=*V-var y*]$_{cv}$) #$_\Gamma$ Γ
proof −
have $\{\!|$ *z* : *b* | *c* $|\!\}$ = $\{\!|$ *y* : *b* | (*y* ↔ *z*) ∙ *c* $|\!\}$ **using** *type-eq-flip assms* **by** *auto*
moreover hence (*y* ↔ *z*) ∙ *c* = *c*[*z*::=*V-var y*]$_{cv}$ **using** *assms subst-v-c-def* **by** *auto*
ultimately have $\{\!|$ *z* : *b* | *c* $|\!\}$ = $\{\!|$ *y* : *b* | *c*[*z*::=*V-var y*]$_{cv}$ $|\!\}$ **by** *metis*
thus *?thesis* **using** *assms wfT-wf-cons*[*of* Θ $\mathcal{B}$ Γ *y b*] *fresh-Pair* **by** *metis*
qed

lemma *wfT-wfC-cons*:
assumes *wfT P* $\mathcal{B}$ Γ $\{\!|$ *z1* : *b* | *c1* $|\!\}$ **and** *wfT P* $\mathcal{B}$ Γ $\{\!|$ *z2* : *b* | *c2* $|\!\}$ **and** *atom x* ♯ (*c1*,*c2*,Γ)
shows *wfC P* $\mathcal{B}$ ((*x*,*b*,*c1*[*z1*::=*V-var x*]$_v$) #$_\Gamma$Γ) (*c2*[*z2*::=*V-var x*]$_v$) (**is** *wfC P* $\mathcal{B}$ *?G ?c*)
proof −
have *eq*: $\{\!|$ *z2* : *b* | *c2* $|\!\}$ = $\{\!|$ *x* : *b* | *c2*[*z2*::=*V-var x*]$_{cv}$ $|\!\}$ **using** *type-eq-subst assms fresh-prod3* **by** *simp*
have *eq2*: $\{\!|$ *z1* : *b* | *c1* $|\!\}$ = $\{\!|$ *x* : *b* | *c1*[*z1*::=*V-var x*]$_{cv}$ $|\!\}$ **using** *type-eq-subst assms fresh-prod3* **by** *simp*
moreover have *wfT P* $\mathcal{B}$ Γ $\{\!|$ *x* : *b* | *c1*[*z1*::=*V-var x*]$_{cv}$ $|\!\}$ **using** *assms eq2* **by** *auto*
moreover hence *wfG P* $\mathcal{B}$ ((*x*,*b*,*c1*[*z1*::=*V-var x*]$_{cv}$) #$_\Gamma$Γ) **using** *wfT-wf-cons fresh-prod3 assms* **by** *auto*
moreover have *wfT P* $\mathcal{B}$ Γ $\{\!|$ *x* : *b* | *c2*[*z2*::=*V-var x*]$_{cv}$ $|\!\}$ **using** *assms eq* **by** *auto*
moreover hence *wfC P* $\mathcal{B}$ ((*x*,*b*,*TRUE*) #$_\Gamma$Γ) (*c2*[*z2*::=*V-var x*]$_{cv}$) **using** *wfT-wfC assms fresh-prod3* **by** *simp*
ultimately show *?thesis* **using** *wfC-replace-cons subst-v-c-def* **by** *simp*
qed

lemma *wfT-wfC2*:
fixes *c*::*c* **and** *x*::*x*
assumes Θ ; $\mathcal{B}$; Γ $\vdash_{wf}$ $\{\!|$ *z* : *b* | *c* $|\!\}$ **and** *atom x* ♯ Γ
shows Θ ; $\mathcal{B}$; (*x*,*b*,*TRUE*)#$_\Gamma$Γ $\vdash_{wf}$ *c*[*z*::=[*x*]^v]$_v$
proof(*cases x*=*z*)
case *True*
then show *?thesis* **using** *wfT-wfC assms* **by** *auto*
next
case *False*
hence *atom x* ♯ *c* **using** *wfT-fresh-c assms* **by** *metis*
hence $\{\!|$ *x* : *b* | *c*[*z*::=[*x*]^v]$_v$ $|\!\}$ = $\{\!|$ *z* : *b* | *c* $|\!\}$
using *τ.eq-iff Abs1-eq-iff*(3)[*of x c*[*z*::=[*x*]^v]$_v$ *z c*]
by (*metis flip-subst-v type-eq-flip*)
hence Θ ; $\mathcal{B}$; Γ $\vdash_{wf}$ $\{\!|$ *x* : *b* | *c*[*z*::=[*x*]^v]$_v$ $|\!\}$ **using** *assms* **by** *metis*
thus *?thesis* **using** *wfT-wfC assms* **by** *auto*
qed

lemma *wfT-wfG*:
fixes *x*::*x* **and** Γ::Γ **and** *z*::*x* **and** *c*::*c* **and** *b*::*b*

assumes Θ ; $\mathcal{B}$; $\Gamma \vdash_{wf} \{\!|\ z : b \mid c\ |\!\}$ **and** *atom* $x \sharp \Gamma$
shows Θ ; $\mathcal{B}$ $\vdash_{wf}$ $(x,b,\ c[z::=[\ x\ ]^v]_v)$ $\#_\Gamma$ Γ
proof −
have Θ ; $\mathcal{B}$; $(x,\ b,\ TRUE)$ $\#_\Gamma$ Γ $\vdash_{wf}$ $c[z::=[\ x\ ]^v]_v$ **using** *wfT-wfC2 assms* **by** *metis*
thus *?thesis* **using** *wfG-consI assms wfT-wfB b-of.simps wfX-wfY* **by** *metis*
qed

lemma *wfG-replace-inside2*:
fixes Γ::Γ
assumes *wfG P* $\mathcal{B}$ $(\Gamma'$ @ $(x,\ b,\ c')$ $\#_\Gamma$ $\Gamma)$ **and** *wfG P* $\mathcal{B}$ $((x,b,c)$ $\#_\Gamma\Gamma)$
shows *wfG P* $\mathcal{B}$ $(\Gamma'$ @ $(x,\ b,\ c)$ $\#_\Gamma$ $\Gamma)$
proof −
have *wfC P* $\mathcal{B}$ $((x,b,TRUE)$ $\#_\Gamma\Gamma)$ c **using** *wfG-wfC assms* **by** *auto*
thus *?thesis* **using** *wf-replace-inside1(3)[OF assms(1)]* **by** *auto*
qed

lemma *wfG-replace-inside-full*:
fixes Γ::Γ
assumes *wfG P* $\mathcal{B}$ $(\Gamma'$ @ $(x,\ b,\ c')$ $\#_\Gamma$ $\Gamma)$ **and** *wfG P* $\mathcal{B}$ $(\Gamma'@((x,b,c)$ $\#_\Gamma\Gamma))$
shows *wfG P* $\mathcal{B}$ $(\Gamma'$ @ $(x,\ b,\ c)$ $\#_\Gamma$ $\Gamma)$
proof −
have *wfG P* $\mathcal{B}$ $((x,b,c)$ $\#_\Gamma\Gamma)$ **using** *wfG-suffix assms* **by** *auto*
thus *?thesis* **using** *wfG-replace-inside assms* **by** *auto*
qed

lemma *wfT-replace-inside2*:
assumes *wfT* Θ $\mathcal{B}$ $(\Gamma'$ @ $(x,\ b,\ c')$ $\#_\Gamma$ $\Gamma)$ t **and** *wfG* Θ $\mathcal{B}$ $(\Gamma'@((x,b,c)$ $\#_\Gamma\Gamma))$
shows *wfT* Θ $\mathcal{B}$ $(\Gamma'$ @ $(x,\ b,\ c)$ $\#_\Gamma$ $\Gamma)$ t
proof −
have *wfG* Θ $\mathcal{B}$ $(((x,b,c)$ $\#_\Gamma\Gamma))$ **using** *wfG-suffix assms* **by** *auto*
hence *wfC* Θ $\mathcal{B}$ $((x,b,TRUE)$ $\#_\Gamma\Gamma)$ c **using** *wfG-wfC* **by** *auto*
thus *?thesis* **using** *wf-replace-inside assms* **by** *metis*
qed

lemma *wfD-unique*:
assumes *wfD P* $\mathcal{B}$ Γ Δ **and** (u,τ') $\in$ *setD* Δ **and** (u,τ) $\in$ *setD* Δ
shows $\tau'=\tau$
using *assms* **proof**(*induct* Δ *rule*: Δ-*induct*)
case *DNil*
then show *?case* **by** *auto*
next
case (*DCons* u' t' D)
hence $*$: *wfD P* $\mathcal{B}$ Γ $((u',t')$ $\#_\Delta$ $D)$ **using** *Cons* **by** *auto*
show *?case* **proof**(*cases* $u=u'$)
case *True*
then have $u \notin$ *fst* ' *setD D* **using** *wfD-elims* $*$ **by** *blast*
then show *?thesis* **using** *DCons* **by** *force*
next
case *False*
then show *?thesis* **using** *DCons wfD-elims* $*$ **by** (*metis fst-conv setD-ConsD*)

qed
qed

lemma *replace-in-g-forget*:
fixes $x::x$
assumes *wfG P B G*
shows *atom* $x \notin$ *atom-dom* $G \Longrightarrow (G[x \longmapsto c]) = G$ **and**
atom $x \sharp G \Longrightarrow (G[x \longmapsto c]) = G$
proof −
show *atom* $x \notin$ *atom-dom* $G \Longrightarrow G[x \longmapsto c] = G$ **by** (*induct G rule*: Γ-*induct*,*auto*)
thus *atom* $x \sharp G \Longrightarrow (G[x \longmapsto c]) = G$ **using** *wfG-x-fresh assms* **by** *simp*
qed

lemma *replace-in-g-fresh-single*:
fixes $G::\Gamma$ **and** $x::x$
assumes $\langle \Theta\ ;\ \mathcal{B} \vdash_{wf} G[x' \longmapsto c'] \rangle$ **and** *atom* $x \sharp G$ **and** $\langle \Theta\ ;\ \mathcal{B} \vdash_{wf} G \rangle$
shows *atom* $x \sharp G[x' \longmapsto c']$
using *rig-dom-eq wfG-dom-supp assms fresh-def atom-dom.simps dom.simps* **by** *metis*

8.17 Substitution

lemma *wfC-cons-switch*:
fixes $c::c$ **and** $c'::c$
assumes $\Theta\ ;\ \mathcal{B}\ ;\ (x,\ b,\ c)\ \#_\Gamma\ \Gamma\ \vdash_{wf} c'$
shows $\Theta\ ;\ \mathcal{B}\ ;\ (x,\ b,\ c')\ \#_\Gamma\ \Gamma\ \vdash_{wf} c$
proof −
have $*{:}\Theta\ ;\ \mathcal{B} \vdash_{wf} (x,\ b,\ c)\ \#_\Gamma\ \Gamma$ **using** *wfC-wf assms* **by** *auto*
hence *atom* $x \sharp \Gamma \wedge$ *wfG* $\Theta\ \mathcal{B}\ \Gamma \wedge \Theta\ ;\ \mathcal{B} \vdash_{wf} b$ **using** *wfG-cons* **by** *auto*
hence $\Theta\ ;\ \mathcal{B}\ ;\ (x,\ b,\ TRUE)\ \#_\Gamma\ \Gamma\ \vdash_{wf} TRUE$ **using** *wfC-trueI wfG-cons2I* **by** *simp*
hence $\Theta\ ;\ \mathcal{B}\ ;(x,\ b,\ TRUE)\ \#_\Gamma\ \Gamma\ \vdash_{wf} c'$
using *wf-replace-inside1*(*2*)[*of* $\Theta\ \mathcal{B}\ (x,\ b,\ c)\ \#_\Gamma\ \Gamma\ c'$ *GNil x b c* Γ *TRUE*] *assms* **by** *auto*
hence *wfG* $\Theta\ \mathcal{B}\ ((x,b,c')\ \#_\Gamma\Gamma)$ **using** *wf-replace-inside1*(*3*)[*OF* $*$, *of GNil x b c* Γ c'] **by** *auto*
moreover have *wfC* $\Theta\ \mathcal{B}\ ((x,b,TRUE)\ \#_\Gamma\Gamma)$ *c* **proof**(*cases c* $\in \{$ *TRUE*, *FALSE* $\}$)
case *True*
have $\Theta\ ;\ \mathcal{B} \vdash_{wf} \Gamma \wedge$ *atom* $x \sharp \Gamma \wedge \Theta\ ;\ \mathcal{B} \vdash_{wf} b$ **using** *wfG-elims*(*2*)[*OF* $*$] **by** *auto*
hence $\Theta\ ;\ \mathcal{B} \vdash_{wf} (x,b,TRUE)\ \#_\Gamma\ \Gamma$ **using** *wfG-cons-TRUE* **by** *auto*
then show *?thesis* **using** *wfC-trueI wfC-falseI True* **by** *auto*
next
case *False*
then show *?thesis* **using** *wfG-elims*(*2*)[*OF* $*$] **by** *auto*
qed
ultimately show *?thesis* **using** *wfC-replace-cons* **by** *auto*
qed

lemma *subst-g-inside-simple*:
fixes $\Gamma_1::\Gamma$ **and** $\Gamma_2::\Gamma$
assumes *wfG P* $\mathcal{B}$ $(\Gamma_1@((x,b,c)\ \#_\Gamma\Gamma_2))$
shows $(\Gamma_1@((x,b,c)\ \#_\Gamma\Gamma_2))[x::=v]_{\Gamma v} = \Gamma_1[x::=v]_{\Gamma v}@\Gamma_2$
using *assms* **proof**(*induct* Γ_1 *rule*: Γ-*induct*)
case *GNil*
then show *?case* **using** *subst-gv.simps* **by** *simp*
next

case (*GCons x′ b′ c′ G*)
hence $*$:*P* ; $\mathcal{B}$ $\vdash_{wf}$ (*x′*, *b′*, *c′*) $\#_\Gamma$ (*G* @ (*x*, *b*, *c*) $\#_\Gamma$ Γ_2) **by** *auto*
hence *x*≠*x′*
using *GCons append-Cons wfG-cons-fresh2*[*OF* $*$] **by** *auto*
hence ((*GCons* (*x′*, *b′*, *c′*) *G*) @ (*GCons* (*x*, *b*, *c*) Γ_2))[*x*::=*v*]$_{\Gamma v}$ =
(*GCons* (*x′*, *b′*, *c′*) (*G* @ (*GCons* (*x*, *b*, *c*) Γ_2)))[*x*::=*v*]$_{\Gamma v}$ **by** *auto*
also have ... = *GCons* (*x′*, *b′*, *c′*[*x*::=*v*]$_{cv}$) ((*G* @ (*GCons* (*x*, *b*, *c*) Γ_2))[*x*::=*v*]$_{\Gamma v}$)
using *subst-gv.simps* ‹*x*≠*x′*› **by** *simp*
also have ... = (*x′*, *b′*, *c′*[*x*::=*v*]$_{cv}$) $\#_\Gamma$ (*G*[*x*::=*v*]$_{\Gamma v}$ @ Γ_2) **using** *GCons* $*$ *wfG-elims* **by** *metis*
also have ... = ((*x′*, *b′*, *c′*) $\#_\Gamma$ *G*)[*x*::=*v*]$_{\Gamma v}$ @ Γ_2 **using** *subst-gv.simps* ‹*x*≠*x′*› **by** *simp*
finally show *?case* **by** *blast*
qed

lemma *subst-c-TRUE-FALSE*:
fixes *c*::*c*
assumes *c* ∉ {*TRUE*,*FALSE*}
shows *c*[*x*::=*v′*]$_{cv}$ ∉ {*TRUE*, *FALSE*}
using *assms* **by**(*nominal-induct c rule*:*c.strong-induct*,*auto simp add*: *subst-cv.simps*)

lemma *lookup-subst*:
assumes *Some* (*b*, *c*) = *lookup* Γ *x* **and** *x* ≠ *x′*
shows ∃ *c′*. *Some* (*b*,*c′*) = *lookup* Γ[*x′*::=*v′*]$_{\Gamma v}$ *x*
using *assms* **proof**(*induct* Γ *rule*: Γ-*induct*)
case *GNil*
then show *?case* **by** *auto*
next
case (*GCons x1 b1 c1* Γ*1*)
then show *?case* **proof**(*cases x1*=*x′*)
case *True*
then show *?thesis* **using** *subst-gv.simps GCons* **by** *auto*
next
case *False*
thm *subst-gv.simps*
hence $*$:((*x1*, *b1*, *c1*) $\#_\Gamma$ Γ*1*)[*x′*::=*v′*]$_{\Gamma v}$ = ((*x1*, *b1*, *c1*[*x′*::=*v′*]$_{cv}$) $\#_\Gamma$ Γ*1*[*x′*::=*v′*]$_{\Gamma v}$) **using** *subst-gv.simps* **by** *auto*
then show *?thesis* **proof**(*cases x1*=*x*)
case *True*
then show *?thesis* **using** *lookup.simps* $*$
using *GCons.prems*(*1*) **by** *auto*
next
case *False*
then show *?thesis* **using** *lookup.simps* $*$
using *GCons.prems*(*1*) **by** (*simp add*: *GCons.hyps assms*(*2*))
qed
qed
qed

lemma *wf-subst1*:
fixes Γ::Γ **and** Γ′::Γ **and** *v*::*v* **and** *e*::*e* **and** *c*::*c* **and** τ::τ **and** *ts*::(*string*$*$τ) *list* **and** Δ::Δ **and** *b*::*b* **and** *ftq*::*fun-typ-q* **and** *ft*::*fun-typ* **and** *ce*::*ce* **and** *td*::*type-def*
shows *wfV-subst*: Θ ; $\mathcal{B}$; Γ $\vdash_{wf}$ *v* : *b* $\Longrightarrow$ Γ=Γ$_1$@((*x*,*b′*,*c′*) $\#_\Gamma$Γ$_2$) $\Longrightarrow$ Θ ; $\mathcal{B}$;Γ$_2$ $\vdash_{wf}$ *v′* : *b′* $\Longrightarrow$ Θ ; $\mathcal{B}$; Γ[*x*::=*v′*]$_{\Gamma v}$ $\vdash_{wf}$ *v*[*x*::=*v′*]$_{vv}$: *b* **and**

wfC-subst: Θ ; $\mathcal{B}$; Γ $\vdash_{wf}$ c $\Longrightarrow$ Γ=Γ_1@((x,b',c') #$_\Gamma$$\Gamma_2$) $\Longrightarrow$ Θ ; $\mathcal{B}$; Γ_2 $\vdash_{wf}$ v' : b' $\Longrightarrow$ Θ ; $\mathcal{B}$; $\Gamma[x::=v']_{\Gamma v}$ $\vdash_{wf}$ $c[x::=v']_{cv}$ **and**
wfG-subst: Θ ; $\mathcal{B}$ $\vdash_{wf}$ Γ $\Longrightarrow$ Γ=Γ_1@((x,b',c') #$_\Gamma$$\Gamma_2$) $\Longrightarrow$ Θ ; $\mathcal{B}$; Γ_2 $\vdash_{wf}$ v' : b' $\Longrightarrow$ Θ ; $\mathcal{B}$ $\vdash_{wf}$ $\Gamma[x::=v']_{\Gamma v}$ **and**
wfT-subst: Θ ; $\mathcal{B}$; Γ $\vdash_{wf}$ τ $\Longrightarrow$ Γ=Γ_1@((x,b',c') #$_\Gamma$$\Gamma_2$) $\Longrightarrow$ Θ ; $\mathcal{B}$; Γ_2 $\vdash_{wf}$ v' : b' $\Longrightarrow$ Θ ; $\mathcal{B}$; $\Gamma[x::=v']_{\Gamma v}$ $\vdash_{wf}$ $\tau[x::=v']_{\tau v}$ **and**
Θ ; $\mathcal{B}$; Γ $\vdash_{wf}$ ts $\Longrightarrow$ *True* **and**
$\vdash_{wf}$ Θ $\Longrightarrow$ *True* **and**
Θ ; $\mathcal{B}$ $\vdash_{wf}$ b $\Longrightarrow$ *True* **and**
wfCE-subst: Θ ; $\mathcal{B}$; Γ $\vdash_{wf}$ ce : b $\Longrightarrow$ Γ=Γ_1@((x,b',c') #$_\Gamma$$\Gamma_2$) $\Longrightarrow$ Θ ; $\mathcal{B}$; Γ_2 $\vdash_{wf}$ v' : b' $\Longrightarrow$ Θ ; $\mathcal{B}$; $\Gamma[x::=v']_{\Gamma v}$ $\vdash_{wf}$ $ce[x::=v']_{cev}$: b **and**
Θ $\vdash_{wf}$ td $\Longrightarrow$ *True*
proof(*nominal-induct*
b **and** c **and** Γ **and** τ **and** ts **and** Θ **and** b **and** b **and** td
avoiding: x v'
arbitrary: Γ_1 **and** Γ_1 **and** Γ_1 **and** Γ_1 **and** Γ_1 **and** Γ_1 **and** Γ_1 **and** Γ_1 **and** Γ_1 **and** Γ_1 **and** Γ_1 **and** Γ_1 **and** Γ_1 **and** Γ_1 **and** Γ_1
rule:*wfV-wfC-wfG-wfT-wfTs-wfTh-wfB-wfCE-wfTD.strong-induct*)
case (*wfV-varI* Θ $\mathcal{B}$ Γ *b1 c1 x1*)

show *?case* **proof**(*cases x1=x*)
case *True*
hence (*V-var x1*)$[x::=v']_{vv}$ = v' **using** *subst-vv.simps* **by** *auto*
moreover have b' = *b1* **using** *wfV-varI True lookup-inside-wf*
by (*metis option.inject prod.inject*)
moreover have Θ ; $\mathcal{B}$; $\Gamma[x::=v']_{\Gamma v}$ $\vdash_{wf}$ v' : b' **using** *wfV-varI subst-g-inside-simple wf-weakening*
append-g-setGU sup-ge2 wfV-wf **by** *metis*
ultimately show *?thesis* **by** *auto*
next
case *False*
hence (*V-var x1*)$[x::=v']_{vv}$ = (*V-var x1*) **using** *subst-vv.simps* **by** *auto*
moreover have Θ ; $\mathcal{B}$ $\vdash_{wf}$ $\Gamma[x::=v']_{\Gamma v}$ **using** *wfV-varI* **by** *simp*
moreover obtain *c1'* **where** *Some* (*b1*, *c1'*) = *lookup* $\Gamma[x::=v']_{\Gamma v}$ *x1* **using** *wfV-varI False lookup-subst* **by** *metis*
ultimately show *?thesis* **using** *Wellformed.wfV-varI*[*of* Θ $\mathcal{B}$ $\Gamma[x::=v']_{\Gamma v}$ *b1 c1' x1*] **by** *metis*
qed
next
case (*wfV-litI* Θ Γ *l*)

then show *?case* **using** *subst-vv.simps wf-intros* **by** *auto*
next
case (*wfV-pairI* Θ Γ *v1 b1 v2 b2*)
then show *?case* **using** *subst-vv.simps wf-intros* **by** *auto*
next
case (*wfV-consI s dclist* Θ *dc x b c* Γ *v*)
then show *?case* **using** *subst-vv.simps wf-intros* **by** *auto*
next
case (*wfV-conspI s bv dclist* Θ *dc x' b' c* $\mathcal{B}$ *b* Γ *va*)
show *?case* **unfolding** *subst-vv.simps* **proof**
show ‹*AF-typedef-poly s bv dclist* ∈ *set* Θ› **and** ‹(*dc*, {‖ x' : b' | c ‖}) ∈ *set dclist*› **using** *wfV-conspI* **by** *auto*

show ‹ Θ ;$\mathcal{B}$ $\vdash_{wf}$ b › **using** *wfV-conspI* **by** *auto*
have *atom bv* $\sharp$ $\Gamma[x::=v']_{\Gamma v}$ **using** *fresh-subst-gv-if wfV-conspI* **by** *metis*
moreover have *atom bv* $\sharp$ $va[x::=v']_{vv}$ **using** *wfV-conspI fresh-subst-if* **by** *simp*
ultimately show ‹*atom bv* $\sharp$ (Θ, $\mathcal{B}$, $\Gamma[x::=v']_{\Gamma v}$, b, $va[x::=v']_{vv}$)› **unfolding** *fresh-prodN* **using** *wfV-conspI* **by** *auto*
show ‹ Θ ; $\mathcal{B}$; $\Gamma[x::=v']_{\Gamma v}$ $\vdash_{wf}$ $va[x::=v']_{vv}$: $b'[bv::=b]_{bb}$ › **using** *wfV-conspI* **by** *auto*
qed

next
case (*wfTI z* Θ $\mathcal{B}$ Γ b c)
have Θ ; $\mathcal{B}$; $\Gamma[x::=v']_{\Gamma v}$ $\vdash_{wf}$ {‖ z : b | $c[x::=v']_{cv}$ ‖} **proof**
have ‹Θ ; $\mathcal{B}$; ($(z, b, TRUE)$ $\#_\Gamma$ Γ)$[x::=v']_{\Gamma v}$ $\vdash_{wf}$ $c[x::=v']_{cv}$ ›
proof(*rule wfTI*(*9*))
show ‹$(z, b, TRUE)$ $\#_\Gamma$ Γ = $((z, b, TRUE)$ $\#_\Gamma$ $\Gamma_1)$ @ (x, b', c') $\#_\Gamma$ Γ_2› **using** *wfTI append-g.simps* **by** *simp*
show ‹ Θ ; $\mathcal{B}$; Γ_2 $\vdash_{wf}$ v' : b' › **using** *wfTI* **by** *auto*
qed
thus ∗:‹Θ ; $\mathcal{B}$; $(z, b, TRUE)$ $\#_\Gamma$ $\Gamma[x::=v']_{\Gamma v}$ $\vdash_{wf}$ $c[x::=v']_{cv}$ ›
using *subst-gv.simps subst-cv.simps wfTI fresh-x-neq* **by** *auto*

have *atom z* $\sharp$ $\Gamma[x::=v']_{\Gamma v}$ **using** *fresh-subst-gv-if wfTI* **by** *metis*
moreover have Θ ; $\mathcal{B}$ $\vdash_{wf}$ $\Gamma[x::=v']_{\Gamma v}$ **using** *wfTI wfX-wfY wfG-elims subst-gv.simps* ∗ **by** *metis*
ultimately show ‹*atom z* $\sharp$ (Θ, $\mathcal{B}$, $\Gamma[x::=v']_{\Gamma v}$)› **using** *wfG-fresh-x* **by** *metis*
show ‹ Θ ; $\mathcal{B}$ $\vdash_{wf}$ b › **using** *wfTI* **by** *auto*

qed
thus *?case* **using** *subst-tv.simps wfTI* **by** *auto*
next
case (*wfC-trueI* Θ Γ)
then show *?case* **using** *subst-cv.simps wf-intros* **by** *auto*
next
case (*wfC-falseI* Θ Γ)
then show *?case* **using** *subst-cv.simps wf-intros* **by** *auto*
next
case (*wfC-eqI* Θ $\mathcal{B}$ Γ *e1 b e2*)
show *?case* **proof**(*subst subst-cv.simps*,*rule*)
show Θ ; $\mathcal{B}$; $\Gamma[x::=v']_{\Gamma v}$ $\vdash_{wf}$ $e1[x::=v']_{cev}$: b **using** *wfC-eqI subst-dv.simps* **by** *auto*
show Θ ; $\mathcal{B}$; $\Gamma[x::=v']_{\Gamma v}$ $\vdash_{wf}$ $e2[x::=v']_{cev}$: b **using** *wfC-eqI* **by** *auto*
qed
next
case (*wfC-conjI* Θ Γ *c1 c2*)
then show *?case* **using** *subst-cv.simps wf-intros* **by** *auto*
next
case (*wfC-disjI* Θ Γ *c1 c2*)
then show *?case* **using** *subst-cv.simps wf-intros* **by** *auto*
next
case (*wfC-notI* Θ Γ *c1*)
then show *?case* **using** *subst-cv.simps wf-intros* **by** *auto*
next
case (*wfC-impI* Θ Γ *c1 c2*)
then show *?case* **using** *subst-cv.simps wf-intros* **by** *auto*
next

```
  case (wfG-nilI Θ)
  then show ?case using subst-cv.simps  wf-intros by auto
next
  case (wfG-cons1I c Θ B Γ y b)

  show ?case proof(cases x=y)
    case True
    hence ((y, b, c) #Γ Γ)[x::=v']Γv = Γ using subst-gv.simps by auto
    moreover have Θ ; B ⊢wf Γ using wfG-cons1I by auto
    ultimately show ?thesis by auto
  next
    case False
    have Γ1 ≠ GNil using wfG-cons1I False by auto
    then obtain G where Γ1 = (y, b, c) #Γ G using GCons-eq-append-conv wfG-cons1I by auto
    hence ∗:Γ = G @ (x, b', c') #Γ Γ2 using wfG-cons1I by auto
    hence ((y, b, c) #Γ Γ)[x::=v']Γv =(y, b, c[x::=v']cv) #ΓΓ[x::=v']Γv using subst-gv.simps False
by auto
    moreover have Θ ; B ⊢wf (y, b, c[x::=v']cv) #ΓΓ[x::=v']Γv proof(rule Wellformed.wfG-cons1I)
      show ‹c[x::=v']cv ∉ {TRUE, FALSE}› using wfG-cons1I subst-c-TRUE-FALSE by auto
      show ‹ Θ ; B ⊢wf Γ[x::=v']Γv › using wfG-cons1I ∗ by auto
      have Γ = (G @ ((x, b', c') #Γ GNil)) @ Γ2 using ∗ append-g-assoc by auto
      hence atom y ♯ Γ2 using fresh-suffix ‹atom y ♯ Γ› by auto
      hence atom y ♯ v' using wfG-cons1I wfV-x-fresh by metis
      thus ‹atom y ♯ Γ[x::=v']Γv› using fresh-subst-gv wfG-cons1I by auto
       have ((y, b, TRUE) #Γ Γ)[x::=v']Γv = (y, b, TRUE) #Γ Γ[x::=v']Γv using subst-gv.simps
subst-cv.simps False by auto
    thus ‹ Θ ; B ; (y, b, TRUE) #Γ Γ[x::=v']Γv ⊢wf c[x::=v']cv › using wfG-cons1I(6)[of (y,b,TRUE)
#Γ G] ∗ subst-gv.simps
        wfG-cons1I by fastforce
      show Θ ; B ⊢wf b using wfG-cons1I by auto
    qed
    ultimately show ?thesis by auto
  qed
next
  case (wfG-cons2I c Θ B Γ y b)

  show ?case proof(cases x=y)
    case True
    hence ((y, b, c) #Γ Γ)[x::=v']Γv = Γ using subst-gv.simps by auto
    moreover have Θ ; B ⊢wf Γ using wfG-cons2I by auto
    ultimately show ?thesis by auto
  next
    case False
    have Γ1 ≠ GNil using wfG-cons2I False by auto
    then obtain G where Γ1 = (y, b, c) #Γ G using GCons-eq-append-conv wfG-cons2I by auto
    hence ∗:Γ = G @ (x, b', c') #Γ Γ2 using wfG-cons2I by auto
    hence ((y, b, c) #Γ Γ)[x::=v']Γv =(y, b, c[x::=v']cv) #ΓΓ[x::=v']Γv using subst-gv.simps False
by auto
    moreover have Θ ; B ⊢wf (y, b, c[x::=v']cv) #ΓΓ[x::=v']Γv proof(rule Wellformed.wfG-cons2I)
      show ‹c[x::=v']cv ∈ {TRUE, FALSE}› using subst-cv.simps wfG-cons2I by auto
      show ‹ Θ ; B ⊢wf Γ[x::=v']Γv › using wfG-cons2I ∗ by auto
      have Γ = (G @ ((x, b', c') #Γ GNil)) @ Γ2 using ∗ append-g-assoc by auto
```

hence *atom* $y \sharp \Gamma_2$ **using** *fresh-suffix wfG-cons2I* **by** *metis*
hence *atom* $y \sharp v'$ **using** *wfG-cons2I wfV-x-fresh* **by** *metis*
thus ‹*atom* $y \sharp \Gamma[x::=v']_{\Gamma v}$› **using** *fresh-subst-gv wfG-cons2I* **by** *auto*
show $\Theta\ ;\ \mathcal{B} \vdash_{wf} b$ **using** *wfG-cons2I* **by** *auto*
qed
ultimately show *?thesis* **by** *auto*
qed
next
case (*wfCE-valI* $\Theta\ \mathcal{B}\ \Gamma\ v\ b$)
then show *?case* **using** *subst-vv.simps wf-intros* **by** *auto*
next
case (*wfCE-plusI* $\Theta\ \mathcal{B}\ \Gamma$ *v1 v2*)
then show *?case* **using** *subst-vv.simps wf-intros* **by** *auto*
next
case (*wfCE-leqI* $\Theta\ \mathcal{B}\ \Gamma$ *v1 v2*)
then show *?case* **using** *subst-vv.simps wf-intros* **by** *auto*
next
case (*wfCE-fstI* $\Theta\ \mathcal{B}\ \Gamma$ *v1 b1 b2*)
then show *?case* **using** *Wellformed.wfCE-fstI subst-cev.simps* **by** *metis*
next
case (*wfCE-sndI* $\Theta\ \mathcal{B}\ \Gamma$ *v1 b1 b2*)
then show *?case* **using** *subst-cev.simps wf-intros* **by** *metis*
next
case (*wfCE-concatI* $\Theta\ \mathcal{B}\ \Gamma$ *v1 v2*)
then show *?case* **using** *subst-vv.simps wf-intros* **by** *auto*
next
case (*wfCE-lenI* $\Theta\ \mathcal{B}\ \Gamma$ *v1*)
then show *?case* **using** *subst-vv.simps wf-intros* **by** *auto*
qed(*metis subst-sv.simps wf-intros*)+

lemma *wf-subst2*:
fixes Γ::Γ **and** Γ'::Γ **and** *v*::*v* **and** *e*::*e* **and** *c*::*c* **and** τ::τ **and** *ts*::(*string*∗τ) *list* **and** Δ::Δ **and** *b*::*b* **and** *ftq*::*fun-typ-q* **and** *ft*::*fun-typ* **and** *ce*::*ce* **and** *td*::*type-def*
shows $\Theta\ ;\ \Phi\ ;\ \mathcal{B}\ ;\ \Gamma\ ;\ \Delta \vdash_{wf} e : b \implies \Gamma{=}\Gamma_1@((x,b',c')\ \#_\Gamma\Gamma_2) \implies \Theta\ ;\ \mathcal{B}\ ;\ \Gamma_2 \vdash_{wf} v' : b' \implies \Theta\ ;\ \Phi\ ;\ \mathcal{B}\ ;\ \Gamma[x::=v']_{\Gamma v}\ ;\ \Delta[x::=v']_{\Delta v} \vdash_{wf} e[x::=v']_{ev} : b$ **and**
$\Theta\ ;\ \Phi\ ;\ \mathcal{B}\ ;\ \Gamma\ ;\ \Delta \vdash_{wf} s : b \implies \Gamma{=}\Gamma_1@((x,b',c')\ \#_\Gamma\Gamma_2) \implies \Theta\ ;\mathcal{B}\ ;\ \Gamma_2 \vdash_{wf} v' : b' \implies \Theta\ ;\ \Phi\ ;\ \mathcal{B}\ ;\ \Gamma[x::=v']_{\Gamma v}\ ;\ \Delta[x::=v']_{\Delta v} \vdash_{wf} s[x::=v']_{sv} : b$ **and**
$\Theta\ ;\ \Phi\ ;\ \mathcal{B}\ ;\ \Gamma\ ;\ \Delta\ ;\ tid\ ;\ dc\ ;\ t \vdash_{wf} cs : b \implies \Gamma{=}\Gamma_1@((x,b',c')\ \#_\Gamma\Gamma_2) \implies \Theta\ ;\ \mathcal{B}\ ;\ \Gamma_2 \vdash_{wf} v' : b' \implies \Theta\ ;\ \Phi\ ;\ \mathcal{B}\ ;\ \Gamma[x::=v']_{\Gamma v}\ ;\ \Delta[x::=v']_{\Delta v}\ ;\ tid\ ;\ dc\ ;\ t \vdash_{wf} \mathit{subst\text{-}branchv\ cs\ x\ v'} : b$ **and**
$\Theta\ ;\ \Phi\ ;\ \mathcal{B}\ ;\ \Gamma\ ;\ \Delta\ ;\ tid\ ;\ dclist \vdash_{wf} css : b \implies \Gamma{=}\Gamma_1@((x,b',c')\ \#_\Gamma\Gamma_2) \implies \Theta\ ;\ \mathcal{B}\ ;\ \Gamma_2 \vdash_{wf} v' : b' \implies \Theta\ ;\ \Phi\ ;\ \mathcal{B}\ ;\ \Gamma[x::=v']_{\Gamma v}\ ;\ \Delta[x::=v']_{\Delta v}\ ;\ tid\ ;\ dclist \vdash_{wf} \mathit{subst\text{-}branchlv\ css\ x\ v'} : b$ **and**
$\Theta \vdash_{wf} (\Phi::\Phi) \implies \mathit{True}$ **and**
$\Theta\ ;\ \mathcal{B}\ ;\ \Gamma \vdash_{wf} \Delta \implies \Gamma{=}\Gamma_1@((x,b',c')\ \#_\Gamma\Gamma_2) \implies \Theta\ ;\ \mathcal{B}\ ;\ \Gamma_2 \vdash_{wf} v' : b' \implies \Theta\ ;\ \mathcal{B}\ ;\ \Gamma[x::=v']_{\Gamma v} \vdash_{wf} \Delta[x::=v']_{\Delta v}$ **and**
$\Theta\ ;\ \Phi \vdash_{wf} ftq \implies \mathit{True}$ **and**
$\Theta\ ;\ \Phi\ ;\ \mathcal{B} \vdash_{wf} ft \implies \mathit{True}$
proof(*nominal-induct*
b **and** *b* **and** *b* **and** *b* **and** Φ **and** Δ **and** *ftq* **and** *ft*
avoiding: *x v'*
arbitrary: Γ_1 **and** Γ_1 **and** Γ_1 **and** Γ_1 **and** Γ_1 **and** Γ_1 **and** Γ_1 **and** Γ_1 **and** Γ_1 **and** Γ_1 **and** Γ_1 **and** Γ_1 **and** Γ_1 **and** Γ_1 **and** Γ_1
rule:*wfE-wfS-wfCS-wfCSS-wfPhi-wfD-wfFTQ-wfFT.strong-induct*)

case (*wfE-valI* Θ Γ *v b*)
then show *?case* **using** *subst-vv.simps wf-intros wf-subst1*
by (*metis subst-ev.simps*(*1*))
next
case (*wfE-plusI* Θ Γ *v1 v2*)
then show *?case* **using** *subst-vv.simps wf-intros wf-subst1* **by** *auto*
next
case (*wfE-leqI* Θ Φ Γ Δ *v1 v2*)
then show *?case*
using *subst-vv.simps subst-ev.simps subst-ev.simps wf-subst1 Wellformed.wfE-leqI*
by *auto*
next
case (*wfE-fstI* Θ Γ *v1 b1 b2*)
then show *?case* **using** *subst-vv.simps subst-ev.simps wf-subst1 Wellformed.wfE-fstI*
proof −
show *?thesis*
by (*metis* (*full-types*) *subst-ev.simps*(*5*) *wfE-fstI.hyps*(*1*) *wfE-fstI.hyps*(*4*) *wfE-fstI.hyps*(*5*) *wfE-fstI.prems*(*1*) *wfE-fstI.prems*(*2*) *wfE-wfS-wfCS-wfCSS-wfPhi-wfD-wfFTQ-wfFT.wfE-fstI wf-subst1*(*1*))
qed
next
case (*wfE-sndI* Θ Γ *v1 b1 b2*)
then show *?case*
by (*metis* (*full-types*) *subst-ev.simps wfE-sndI Wellformed.wfE-sndI wf-subst1*(*1*))
next
case (*wfE-concatI* Θ Φ Γ Δ *v1 v2*)
then show *?case*
by (*metis* (*full-types*) *subst-ev.simps wfE-sndI Wellformed.wfE-concatI wf-subst1*(*1*))
next
case (*wfE-splitI* Θ Φ Γ Δ *v1 v2*)
then show *?case*
by (*metis* (*full-types*) *subst-ev.simps wfE-sndI Wellformed.wfE-splitI wf-subst1*(*1*))
next
case (*wfE-lenI* Θ Φ Γ Δ *v1*)
then show *?case*
by (*metis* (*full-types*) *subst-ev.simps wfE-sndI Wellformed.wfE-lenI wf-subst1*(*1*))
next
case (*wfE-appI* Θ Φ Γ Δ *f x b c* τ *s'* *v*)
then show *?case*
by (*metis* (*full-types*) *subst-ev.simps wfE-sndI Wellformed.wfE-appI wf-subst1*(*1*))
next
case (*wfE-appPI* Θ Φ $\mathcal{B}$ Γ Δ *b' bv1 v1* τ*1 f1 x1 b1 c1 s1*)
show *?case* **proof**(*subst subst-ev.simps*, *rule*)
show $\Theta \vdash_{wf} \Phi$ **using** *wfE-appPI wfX-wfY* **by** *metis*
show Θ ; $\mathcal{B}$; $\Gamma[x::=v']_{\Gamma v} \vdash_{wf} \Delta[x::=v']_{\Delta v}$ **using** *wfE-appPI* **by** *auto*
show *Some* (*AF-fundef f1* (*AF-fun-typ-some bv1* (*AF-fun-typ x1 b1 c1* τ*1 s1*))) = *lookup-fun* Φ *f1*
using *wfE-appPI* **by** *auto*
show Θ ; $\mathcal{B}$; $\Gamma[x::=v']_{\Gamma v} \vdash_{wf} v1[x::=v']_{vv} : b1[bv1::=b']_b$ **using** *wfE-appPI wf-subst1* **by** *auto*
show Θ ; $\mathcal{B} \vdash_{wf} b'$ **using** *wfE-appPI* **by** *auto*
have *atom bv1* $\sharp$ $\Gamma[x::=v']_{\Gamma v}$ **using** *fresh-subst-gv-if wfE-appPI* **by** *metis*
moreover have *atom bv1* $\sharp$ $v1[x::=v']_{vv}$ **using** *wfE-appPI fresh-subst-if* **by** *simp*
moreover have *atom bv1* $\sharp$ $\Delta[x::=v']_{\Delta v}$ **using** *wfE-appPI fresh-subst-dv-if* **by** *simp*
ultimately show *atom bv1* $\sharp$ (Φ, Θ, $\mathcal{B}$, $\Gamma[x::=v']_{\Gamma v}$, $\Delta[x::=v']_{\Delta v}$, *b'*, $v1[x::=v']_{vv}$, (*b-of* τ*1*)$[bv1::=b']_b$)

```
    using wfE-appPI fresh-prodN by metis
  qed
next
  case (wfE-mvarI Θ Φ B Γ Δ u τ)
  have  Θ ; Φ  ; B ; Γ[x::=v']Γv ; Δ[x::=v']Δv ⊢wf (AE-mvar u) : b-of τ[x::=v']τv proof
    show Θ ⊢wf Φ  using wfE-mvarI by auto
    show Θ ; B  ; Γ[x::=v']Γv  ⊢wf Δ[x::=v']Δv  using wfE-mvarI by auto
    show (u, τ[x::=v']τv) ∈ setD Δ[x::=v']Δv using wfE-mvarI subst-dv-member by auto
  qed
  thus ?case using subst-ev.simps b-of-subst by auto
next
  case (wfD-emptyI Θ Γ)
  then show ?case using subst-dv.simps  wf-intros wf-subst1 by auto
next
   case (wfD-cons Θ B Γ Δ τ u)
  moreover hence u ∉ fst ' setD Δ[x::=v']Δv using subst-dv.simps subst-dv-iff  using subst-dv-fst-eq
by presburger
  ultimately show ?case using subst-dv.simps Wellformed.wfD-cons wf-subst1 by auto
next
  case (wfPhi-emptyI Θ)
  then show ?case by auto
next
  case (wfPhi-consI f Θ Φ ft)
  then show ?case by auto
next
   case (wfS-assertI Θ Φ B x2 c Γ Δ s b)
   show ?case unfolding subst-sv.simps proof
     show ‹ Θ ; Φ ; B ; (x2, B-bool, c[x::=v']cv) #Γ Γ[x::=v']Γv ; Δ[x::=v']Δv ⊢wf s[x::=v']sv : b ›
       using wfS-assertI(4)[of (x2, B-bool, c) #Γ Γ1 x ]  wfS-assertI by auto

     show ‹ Θ ; B ; Γ[x::=v']Γv  ⊢wf c[x::=v']cv › using wfS-assertI wf-subst1 by auto
     show ‹ Θ ; B ; Γ[x::=v']Γv ⊢wf Δ[x::=v']Δv › using wfS-assertI wf-subst1 by auto
     show ‹atom x2 ♯ (Φ, Θ, B, Γ[x::=v']Γv, Δ[x::=v']Δv, c[x::=v']cv, b, s[x::=v']sv)›
      apply(unfold fresh-prodN,intro conjI)
      apply(simp add: wfS-assertI )+
      apply(metis fresh-subst-gv-if wfS-assertI)
      apply(simp add: fresh-prodN fresh-subst-dv-if wfS-assertI)
      apply(simp add: fresh-prodN fresh-subst-v-if subst-v-e-def wfS-assertI)
      apply(simp add: fresh-prodN fresh-subst-v-if subst-v-τ-def wfS-assertI)
      by(simp add: fresh-prodN fresh-subst-v-if subst-v-s-def wfS-assertI)
  qed
next
  case (wfS-letI Θ Φ B Γ Δ e b1 y s b2)
  have Θ ; Φ  ; B ; Γ[x::=v']Γv ; Δ[x::=v']Δv ⊢wf LET y = (e[x::=v']ev) IN (s[x::=v']sv) : b2
  proof
    show ‹ Θ ; Φ  ; B ; Γ[x::=v']Γv ; Δ[x::=v']Δv ⊢wf e[x::=v']ev : b1 › using wfS-letI by auto
    have  ‹ Θ ; Φ  ; B ; ((y, b1, TRUE)  #Γ Γ)[x::=v']Γv ; Δ[x::=v']Δv ⊢wf s[x::=v']sv : b2 ›
      using wfS-letI(6) wfS-letI append-g.simps by metis
    thus ‹ Θ ; Φ  ; B ; (y, b1, TRUE)  #Γ Γ[x::=v']Γv ; Δ[x::=v']Δv ⊢wf s[x::=v']sv : b2 ›
      using wfS-letI subst-gv.simps by auto
    show ‹ Θ ; B ; Γ[x::=v']Γv ⊢wf Δ[x::=v']Δv › using wfS-letI by auto
```

show ‹*atom y* ♯ (Φ, Θ, $\mathcal{B}$, Γ[*x*::=*v'*]$_{\Gamma v}$, Δ[*x*::=*v'*]$_{\Delta v}$, *e*[*x*::=*v'*]$_{ev}$, *b2*)›
apply(*unfold fresh-prodN*,*intro conjI*)
apply(*simp add*: *wfS-letI*)+
apply(*metis fresh-subst-gv-if wfS-letI*)
apply(*simp add*: *fresh-prodN fresh-subst-dv-if wfS-letI*)
apply(*simp add*: *fresh-prodN fresh-subst-v-if subst-v-e-def wfS-letI*)
apply(*simp add*: *fresh-prodN fresh-subst-v-if subst-v-τ-def wfS-letI*)
done
qed
thus *?case* **using** *subst-sv.simps wfS-letI* **by** *auto*
next
case (*wfS-let2I* Θ Φ $\mathcal{B}$ Γ Δ *s1* τ *y s2 b*)
have Θ ; Φ ; $\mathcal{B}$; Γ[*x*::=*v'*]$_{\Gamma v}$; Δ[*x*::=*v'*]$_{\Delta v}$ ⊢$_{wf}$ *LET y* : τ[*x*::=*v'*]$_{\tau v}$ = (*s1*[*x*::=*v'*]$_{sv}$) *IN* (*s2*[*x*::=*v'*]$_{sv}$) : *b*
proof
show ‹ Θ ; Φ ; $\mathcal{B}$; Γ[*x*::=*v'*]$_{\Gamma v}$; Δ[*x*::=*v'*]$_{\Delta v}$ ⊢$_{wf}$ *s1*[*x*::=*v'*]$_{sv}$: *b-of* (τ[*x*::=*v'*]$_{\tau v}$) › **using** *wfS-let2I b-of-subst* **by** *simp*
have ‹ Θ ; Φ ; $\mathcal{B}$; ((*y*, *b-of* τ, *TRUE*) #$_\Gamma$ Γ)[*x*::=*v'*]$_{\Gamma v}$; Δ[*x*::=*v'*]$_{\Delta v}$ ⊢$_{wf}$ *s2*[*x*::=*v'*]$_{sv}$: *b* ›
using *wfS-let2I append-g.simps* **by** *metis*
thus ‹ Θ ; Φ ; $\mathcal{B}$; (*y*, *b-of* τ[*x*::=*v'*]$_{\tau v}$, *TRUE*) #$_\Gamma$ Γ[*x*::=*v'*]$_{\Gamma v}$; Δ[*x*::=*v'*]$_{\Delta v}$ ⊢$_{wf}$ *s2*[*x*::=*v'*]$_{sv}$: *b* ›
using *wfS-let2I subst-gv.simps append-g.simps* **using** *b-of-subst* **by** *simp*
show ‹ Θ ; $\mathcal{B}$; Γ[*x*::=*v'*]$_{\Gamma v}$ ⊢$_{wf}$ τ[*x*::=*v'*]$_{\tau v}$ › **using** *wfS-let2I wf-subst1* **by** *metis*
show ‹*atom y* ♯ (Φ, Θ, $\mathcal{B}$, Γ[*x*::=*v'*]$_{\Gamma v}$, Δ[*x*::=*v'*]$_{\Delta v}$, *s1*[*x*::=*v'*]$_{sv}$, *b*, τ[*x*::=*v'*]$_{\tau v}$)›
apply(*unfold fresh-prodN*,*intro conjI*)
apply(*simp add*: *wfS-let2I*)+
apply(*metis fresh-subst-gv-if wfS-let2I*)
apply(*simp add*: *fresh-prodN fresh-subst-dv-if wfS-let2I*)
apply(*simp add*: *fresh-prodN fresh-subst-v-if subst-v-e-def wfS-let2I*)
apply(*simp add*: *fresh-prodN fresh-subst-v-if subst-v-τ-def wfS-let2I*)+
done
qed
thus *?case* **using** *subst-sv.simps*(*3*) *subst-tv.simps wfS-let2I* **by** *auto*
next
case (*wfS-varI* Θ $\mathcal{B}$ Γ τ *v u* Φ Δ *b s*)
show *?case* **proof**(*subst subst-sv.simps*, *auto simp add*: *u-fresh-xv*,*rule*)
show ‹ Θ ; $\mathcal{B}$; Γ[*x*::=*v'*]$_{\Gamma v}$ ⊢$_{wf}$ τ[*x*::=*v'*]$_{\tau v}$ › **using** *wfS-varI wf-subst1* **by** *auto*
have *b-of* (τ[*x*::=*v'*]$_{\tau v}$) = *b-of* τ **using** *b-of-subst* **by** *auto*
thus ‹ Θ ; $\mathcal{B}$; Γ[*x*::=*v'*]$_{\Gamma v}$ ⊢$_{wf}$ *v*[*x*::=*v'*]$_{vv}$: *b-of* τ[*x*::=*v'*]$_{\tau v}$ › **using** *wfS-varI wf-subst1* **by** *auto*
have ∗:*atom u* ♯ *v'* **using** *wfV-supp wfS-varI fresh-def* **by** *metis*
show ‹*atom u* ♯ (Φ, Θ, $\mathcal{B}$, Γ[*x*::=*v'*]$_{\Gamma v}$, Δ[*x*::=*v'*]$_{\Delta v}$, τ[*x*::=*v'*]$_{\tau v}$, *v*[*x*::=*v'*]$_{vv}$, *b*)›
unfolding *fresh-prodN* **apply**(*auto simp add*: *wfS-varI*)
using *wfS-varI fresh-subst-gv* ∗ *fresh-subst-dv* **by** *metis*+
show ‹ Θ ; Φ ; $\mathcal{B}$; Γ[*x*::=*v'*]$_{\Gamma v}$; (*u*, τ[*x*::=*v'*]$_{\tau v}$) #$_\Delta$ Δ[*x*::=*v'*]$_{\Delta v}$ ⊢$_{wf}$ *s*[*x*::=*v'*]$_{sv}$: *b* › **using** *wfS-varI* **by** *auto*
qed
next
case (*wfS-assignI u* τ Δ Θ $\mathcal{B}$ Γ Φ *v*)
show *?case* **proof**(*subst subst-sv.simps*, *rule wf-intros*)
show ‹(*u*, τ[*x*::=*v'*]$_{\tau v}$) ∈ *setD* Δ[*x*::=*v'*]$_{\Delta v}$› **using** *subst-dv-iff wfS-assignI* **using** *subst-dv-fst-eq*
using *subst-dv-member* **by** *auto*
show ‹ Θ ; $\mathcal{B}$; Γ[*x*::=*v'*]$_{\Gamma v}$ ⊢$_{wf}$ Δ[*x*::=*v'*]$_{\Delta v}$ › **using** *wfS-assignI* **by** *auto*

show ‹ Θ ; $\mathcal{B}$; $\Gamma[x::=v']_{\Gamma v} \vdash_{wf} v[x::=v']_{vv}$: *b-of* $\tau[x::=v']_{\tau v}$ › **using** *wfS-assignI b-of-subst wf-subst1*
by *auto*
show $\Theta \vdash_{wf} \Phi$ **using** *wfS-assignI* **by** *auto*
qed
next

case (*wfS-matchI* Θ $\mathcal{B}$ Γ *v tid dclist* Δ Φ *cs b*)
show *?case* **proof**(*subst subst-sv.simps*, *rule wf-intros*)
show ‹ Θ ; $\mathcal{B}$; $\Gamma[x::=v']_{\Gamma v} \vdash_{wf} v[x::=v']_{vv}$: *B-id tid* › **using** *wfS-matchI wf-subst1* **by** *auto*
show ‹*AF-typedef tid dclist* ∈ *set* Θ› **using** *wfS-matchI* **by** *auto*
show ‹ Θ ; Φ ; $\mathcal{B}$; $\Gamma[x::=v']_{\Gamma v}$; $\Delta[x::=v']_{\Delta v}$; *tid* ; *dclist* $\vdash_{wf}$ *subst-branchlv cs x v'* : *b* › **using**
wfS-matchI **by** *simp*
show Θ ; $\mathcal{B}$; $\Gamma[x::=v']_{\Gamma v} \vdash_{wf} \Delta[x::=v']_{\Delta v}$ **using** *wfS-matchI* **by** *auto*
show $\Theta \vdash_{wf} \Phi$ **using** *wfS-matchI* **by** *auto*
qed
next
case (*wfS-branchI* Θ Φ $\mathcal{B}$ *y* τ Γ Δ *s b tid dc*)
have Θ ; Φ ; $\mathcal{B}$; $\Gamma[x::=v']_{\Gamma v}$; $\Delta[x::=v']_{\Delta v}$; *tid* ; *dc* ; $\tau \vdash_{wf}$ *dc y* ⇒ $(s[x::=v']_{sv})$: *b*
proof
have ‹ Θ ; Φ ; $\mathcal{B}$; $((y,$ *b-of* $\tau,$ *TRUE*$)$ $\#_\Gamma$ $\Gamma)[x::=v']_{\Gamma v}$; $\Delta[x::=v']_{\Delta v} \vdash_{wf} s[x::=v']_{sv}$: *b* ›
using *wfS-branchI append-g.simps* **by** *metis*
thus ‹ Θ ; Φ ; $\mathcal{B}$; $(y,$ *b-of* $\tau,$ *TRUE*$)$ $\#_\Gamma$ $\Gamma[x::=v']_{\Gamma v}$; $\Delta[x::=v']_{\Delta v} \vdash_{wf} s[x::=v']_{sv}$: *b* ›
using *subst-gv.simps b-of-subst wfS-branchI* **by** *simp*
show ‹*atom y* ♯ $(\Phi,$ $\Theta,$ $\mathcal{B},$ $\Gamma[x::=v']_{\Gamma v},$ $\Delta[x::=v']_{\Delta v},$ $\Gamma[x::=v']_{\Gamma v},$ $\tau)$›
apply(*unfold fresh-prodN*,*intro conjI*)
apply(*simp add*: *wfS-branchI*)+
apply(*metis fresh-subst-gv-if wfS-branchI*)
apply(*simp add*: *fresh-prodN fresh-subst-dv-if wfS-branchI*)
apply(*metis fresh-subst-gv-if wfS-branchI*)+
done
show ‹ Θ ; $\mathcal{B}$; $\Gamma[x::=v']_{\Gamma v} \vdash_{wf} \Delta[x::=v']_{\Delta v}$ › **using** *wfS-branchI* **by** *auto*
qed
thus *?case* **using** *subst-branchv.simps wfS-branchI* **by** *auto*

next
case (*wfS-finalI* Θ Φ $\mathcal{B}$ Γ Δ *tid dclist' cs b dclist*)
then show *?case* **using** *subst-branchlv.simps wf-intros* **by** *metis*
next
case (*wfS-cons* Θ Φ $\mathcal{B}$ Γ Δ *tid dclist' cs b css dclist*)
then show *?case* **using** *subst-branchlv.simps wf-intros* **by** *metis*

qed(*metis subst-sv.simps wf-subst1 wf-intros*)+

lemmas *wf-subst* = *wf-subst1 wf-subst2*

lemma *wfG-subst-wfV*:
assumes Θ ; $\mathcal{B} \vdash_{wf} \Gamma'$ @ $(x,$ $b,$ *c0*$[z0::=V\text{-}var\ x]_{cv})$ $\#_\Gamma$ Γ **and** *wfV* Θ $\mathcal{B}$ Γ *v b*
shows Θ ; $\mathcal{B} \vdash_{wf} \Gamma'[x::=v]_{\Gamma v}$ @ Γ
using *assms wf-subst subst-g-inside-simple* **by** *auto*

lemma *wfG-member-subst*:

assumes $(x1,b1,c1) \in setG\ (\Gamma'@\Gamma)$ **and** $wfG\ \Theta\ \mathcal{B}\ (\Gamma'@((x,b,c)\ \#_\Gamma\Gamma))$ **and** $x \neq x1$
shows $\exists\, c1'.\ (x1,b1,c1') \in setG\ ((\Gamma'[x::=v]_{\Gamma v})@\Gamma)$
proof −
consider $(lhs)\ (x1,b1,c1) \in setG\ \Gamma'\ \mid\ (rhs)\ (x1,b1,c1) \in setG\ \Gamma$ **using** *append-g-setGU assms* **by** *auto*
thus *?thesis* **proof**(*cases*)
case *lhs*
hence $(x1,b1,c1[x::=v]_{cv}) \in setG\ (\Gamma'[x::=v]_{\Gamma v})$ **using** *wfG-inside-fresh*[*THEN subst-gv-member-iff*[*OF lhs*]] *assms* **by** *metis*
hence $(x1,b1,c1[x::=v]_{cv}) \in setG\ (\Gamma'[x::=v]_{\Gamma v}@\Gamma)$ **using** *append-g-setGU* **by** *auto*
then show *?thesis* **by** *auto*
next
case *rhs*
hence $(x1,b1,c1) \in setG\ (\Gamma'[x::=v]_{\Gamma v}@\Gamma)$ **using** *append-g-setGU* **by** *auto*
then show *?thesis* **by** *auto*
qed
qed

lemma *wfG-member-subst2*:
assumes $(x1,b1,c1) \in setG\ (\Gamma'@((x,b,c)\ \#_\Gamma\Gamma))$ **and** $wfG\ \Theta\ \mathcal{B}\ (\Gamma'@((x,b,c)\ \#_\Gamma\Gamma))$ **and** $x \neq x1$
shows $\exists\, c1'.\ (x1,b1,c1') \in setG\ ((\Gamma'[x::=v]_{\Gamma v})@\Gamma)$
proof −
consider $(lhs)\ (x1,b1,c1) \in setG\ \Gamma'\ \mid\ (rhs)\ (x1,b1,c1) \in setG\ \Gamma$ **using** *append-g-setGU assms* **by** *auto*
thus *?thesis* **proof**(*cases*)
case *lhs*
hence $(x1,b1,c1[x::=v]_{cv}) \in setG\ (\Gamma'[x::=v]_{\Gamma v})$ **using** *wfG-inside-fresh*[*THEN subst-gv-member-iff*[*OF lhs*]] *assms* **by** *metis*
hence $(x1,b1,c1[x::=v]_{cv}) \in setG\ (\Gamma'[x::=v]_{\Gamma v}@\Gamma)$ **using** *append-g-setGU* **by** *auto*
then show *?thesis* **by** *auto*
next
case *rhs*
hence $(x1,b1,c1) \in setG\ (\Gamma'[x::=v]_{\Gamma v}@\Gamma)$ **using** *append-g-setGU* **by** *auto*
then show *?thesis* **by** *auto*
qed
qed

lemma *wbc-subst*:
fixes $\Gamma::\Gamma$ **and** $\Gamma'::\Gamma$ **and** $v::v$
assumes $wfC\ \Theta\ \mathcal{B}\ (\Gamma'@((x,b,c')\ \#_\Gamma\Gamma))\ c$ **and** $\Theta\ ;\ \mathcal{B}\ ;\ \Gamma \vdash_{wf} v : b$
shows $\Theta\ ;\ \mathcal{B}\ ;\ ((\Gamma'[x::=v]_{\Gamma v})@\Gamma) \vdash_{wf} c[x::=v]_{cv}$
proof −
have $(\Gamma'@((x,b,c')\ \#_\Gamma\Gamma))[x::=v]_{\Gamma v} = ((\Gamma'[x::=v]_{\Gamma v})@\Gamma)$ **using** *assms subst-g-inside-simple wfC-wf* **by** *metis*
thus *?thesis* **using** *wf-subst1*(*2*)[*OF assms*(*1*) - *assms*(*2*)] **by** *metis*
qed

lemma *wfG-inside-fresh-suffix*:
assumes $wfG\ P\ B\ (\Gamma'@(x,b,c)\ \#_\Gamma\Gamma)$
shows $atom\ x\ \sharp\ \Gamma$
proof −
have $wfG\ P\ B\ ((x,b,c)\ \#_\Gamma\Gamma)$ **using** *wfG-suffix assms* **by** *auto*

thus *?thesis* **using** *wfG-elims* **by** *metis*
qed

lemmas *wf-b-subst-lemmas* = *subst-eb.simps wf-intros*
forget-subst subst-b-b-def subst-b-v-def subst-b-ce-def fresh-e-opp-all subst-bb.simps wfV-b-fresh ms-fresh-all(*6*)

lemma *wf-b-subst1*:
fixes Γ::Γ **and** Γ'::Γ **and** *v*::*v* **and** *e*::*e* **and** *c*::*c* **and** τ::τ **and** *ts*::(*string*∗τ) *list* **and** Δ::Δ **and** *b*::*b* **and** *ftq*::*fun-typ-q* **and** *ft*::*fun-typ* **and** *s*::*s* **and** *b'*::*b* **and** *ce*::*ce* **and** *td*::*type-def*
and *cs*::*branch-s* **and** *css*::*branch-list*
shows Θ ; B' ; Γ $\vdash_{wf}$ *v* : *b'* $\Longrightarrow$ {|*bv*|} = B' $\Longrightarrow$ Θ ; B $\vdash_{wf}$ *b* $\Longrightarrow$ Θ ; B ; Γ[*bv*::=*b*]$_{\Gamma b}$ $\vdash_{wf}$ *v*[*bv*::=*b*]$_{vb}$: *b'*[*bv*::=*b*]$_{bb}$ **and**
Θ ; B' ; Γ $\vdash_{wf}$ *c* $\Longrightarrow$ {|*bv*|} = B' $\Longrightarrow$ Θ ; B $\vdash_{wf}$ *b* $\Longrightarrow$ Θ ; B ; Γ[*bv*::=*b*]$_{\Gamma b}$ $\vdash_{wf}$ *c*[*bv*::=*b*]$_{cb}$ **and**
Θ ; B' $\vdash_{wf}$ Γ $\Longrightarrow$ {|*bv*|} = B' $\Longrightarrow$ Θ ; B $\vdash_{wf}$ *b* $\Longrightarrow$ Θ ; B $\vdash_{wf}$ Γ[*bv*::=*b*]$_{\Gamma b}$ **and**
Θ ; B' ; Γ $\vdash_{wf}$ τ $\Longrightarrow$ {|*bv*|} = B' $\Longrightarrow$ Θ ; B $\vdash_{wf}$ *b* $\Longrightarrow$ Θ ; B ; Γ[*bv*::=*b*]$_{\Gamma b}$ $\vdash_{wf}$ τ[*bv*::=*b*]$_{\tau b}$ **and**
Θ ; $\mathcal{B}$; Γ $\vdash_{wf}$ *ts* $\Longrightarrow$ *True* **and**
$\vdash_{wf}$ Θ $\Longrightarrow$ *True* **and**
Θ ; B' $\vdash_{wf}$ *b'* $\Longrightarrow$ {|*bv*|} = B' $\Longrightarrow$ Θ ; B $\vdash_{wf}$ *b* $\Longrightarrow$ Θ ; B $\vdash_{wf}$ *b'*[*bv*::=*b*]$_{bb}$ **and**
Θ ; B' ; Γ $\vdash_{wf}$ *ce* : *b'* $\Longrightarrow$ {|*bv*|} = B' $\Longrightarrow$ Θ ; B $\vdash_{wf}$ *b* $\Longrightarrow$ Θ ; B ; Γ[*bv*::=*b*]$_{\Gamma b}$ $\vdash_{wf}$ *ce*[*bv*::=*b*]$_{ceb}$: *b'*[*bv*::=*b*]$_{bb}$ **and**
Θ $\vdash_{wf}$ *td* $\Longrightarrow$ *True*
proof(*nominal-induct*
b' **and** *c* **and** Γ **and** τ **and** *ts* **and** Θ **and** *b'* **and** *b'* **and** *td*
avoiding: *bv b B*
rule:*wfV-wfC-wfG-wfT-wfTs-wfTh-wfB-wfCE-wfTD.strong-induct*)
case (*wfB-intI* Θ $\mathcal{B}$)
then show *?case* **using** *subst-bb.simps wf-intros wfX-wfY* **by** *metis*
next
case (*wfB-boolI* Θ $\mathcal{B}$)
then show *?case* **using** *subst-bb.simps wf-intros wfX-wfY* **by** *metis*
next
case (*wfB-unitI* Θ $\mathcal{B}$)
then show *?case* **using** *subst-bb.simps wf-intros wfX-wfY* **by** *metis*
next
case (*wfB-bitvecI* Θ $\mathcal{B}$)
then show *?case* **using** *subst-bb.simps wf-intros wfX-wfY* **by** *metis*
next
case (*wfB-pairI* Θ $\mathcal{B}$ *b1 b2*)
then show *?case* **using** *subst-bb.simps wf-intros wfX-wfY* **by** *metis*
next
case (*wfB-consI* Θ *s dclist* $\mathcal{B}$)
then show *?case* **using** *subst-bb.simps Wellformed.wfB-consI* **by** *simp*
next
case (*wfB-appI* Θ *ba s bva dclist* $\mathcal{B}$)
then show *?case* **using** *subst-bb.simps Wellformed.wfB-appI forget-subst wfB-supp*
by (*metis bot.extremum-uniqueI ex-in-conv fresh-def subst-b-b-def supp-empty-fset*)

next
case (*wfV-varI* Θ $\mathcal{B}1$ Γ *b1 c x*)
show *?case* **unfolding** *subst-vb.simps* **proof**
show Θ ; *B* $\vdash_{wf}$ $\Gamma[bv::=b]_{\Gamma b}$ **using** *wfV-varI* **by** *auto*
show *Some* $(b1[bv::=b]_{bb}, c[bv::=b]_{cb})$ = *lookup* $\Gamma[bv::=b]_{\Gamma b}$ *x* **using** *subst-b-lookup wfV-varI* **by** *simp*
qed
next
case (*wfV-litI* Θ $\mathcal{B}$ Γ *l*)
then show *?case* **using** *Wellformed.wfV-litI subst-b-base-for-lit* **by** *simp*
next
case (*wfV-pairI* Θ $\mathcal{B}1$ Γ *v1 b1 v2 b2*)
show *?case* **unfolding** *subst-vb.simps* **proof**(*subst subst-bb.simps*,*rule*)
show Θ ; *B* ; $\Gamma[bv::=b]_{\Gamma b}$ $\vdash_{wf}$ *v1*$[bv::=b]_{vb}$: *b1*$[bv::=b]_{bb}$ **using** *wfV-pairI* **by** *simp*
show Θ ; *B* ; $\Gamma[bv::=b]_{\Gamma b}$ $\vdash_{wf}$ *v2*$[bv::=b]_{vb}$: *b2*$[bv::=b]_{bb}$ **using** *wfV-pairI* **by** *simp*
qed
next
case (*wfV-consI s dclist* Θ *dc x b' c* $\mathcal{B}'$ Γ *v*)
show *?case* **unfolding** *subst-vb.simps* **proof**(*subst subst-bb.simps*, *rule Wellformed.wfV-consI*)
show *1*:*AF-typedef s dclist* $\in$ *set* Θ **using** *wfV-consI* **by** *auto*
show *2*:(*dc*, $\{\!|$ *x* : *b'* | *c* $|\!\}$) $\in$ *set dclist* **using** *wfV-consI* **by** *auto*
have Θ ; *B* ; $\Gamma[bv::=b]_{\Gamma b}$ $\vdash_{wf}$ *v*$[bv::=b]_{vb}$: *b'*$[bv::=b]_{bb}$ **using** *wfV-consI* **by** *auto*
moreover hence *supp b'* = {} **using** *1 2 wfTh-lookup-supp-empty* τ*.supp wfX-wfY* **by** *blast*
moreover hence *b'*$[bv::=b]_{bb}$ = *b'* **using** *forget-subst subst-bb-def fresh-def* **by** (*metis empty-iff subst-b-b-def*)
ultimately show Θ ; *B* ; $\Gamma[bv::=b]_{\Gamma b}$ $\vdash_{wf}$ *v*$[bv::=b]_{vb}$: *b'* **using** *wfV-consI* **by** *simp*
qed
next

case (*wfV-conspI s bva dclist* Θ *dc x b' c* $\mathcal{B}'$ *ba* Γ *v*)
have $*$:*atom bv* $\sharp$ *b'* **using** *wfTh-poly-supp-b*[*of s bva dclist* Θ *dc x b' c*] *fresh-def wfX-wfY* ‹*atom bva* $\sharp$ *bv*›
by (*metis insert-iff not-self-fresh singleton-insert-inj-eq' subsetI subset-antisym wfV-conspI wfV-conspI.hyps*(*4*) *wfV-conspI.prems*(*2*))
show *?case* **unfolding** *subst-vb.simps subst-bb.simps* **proof**
show ‹*AF-typedef-poly s bva dclist* $\in$ *set* Θ› **using** *wfV-conspI* **by** *auto*
show ‹(*dc*, $\{\!|$ *x* : *b'* | *c* $|\!\}$) $\in$ *set dclist*› **using** *wfV-conspI* **by** *auto*

thus ‹ Θ ; *B* $\vdash_{wf}$ *ba*$[bv::=b]_{bb}$ › **using** *wfV-conspI* **by** *metis*
have *atom bva* $\sharp$ $\Gamma[bv::=b]_{\Gamma b}$ **using** *fresh-subst-if subst-b-*Γ*-def wfV-conspI* **by** *metis*
moreover have *atom bva* $\sharp$ *ba*$[bv::=b]_{bb}$ **using** *fresh-subst-if subst-b-b-def wfV-conspI* **by** *metis*
moreover have *atom bva* $\sharp$ *v*$[bv::=b]_{vb}$ **using** *fresh-subst-if subst-b-v-def wfV-conspI* **by** *metis*
ultimately show ‹*atom bva* $\sharp$ (Θ, *B*, $\Gamma[bv::=b]_{\Gamma b}$, *ba*$[bv::=b]_{bb}$, *v*$[bv::=b]_{vb}$)›
unfolding *fresh-prodN* **using** *wfV-conspI fresh-def supp-fset* **by** *auto*
show ‹ Θ ; *B* ; $\Gamma[bv::=b]_{\Gamma b}$ $\vdash_{wf}$ *v*$[bv::=b]_{vb}$: *b'*$[bva::=ba[bv::=b]_{bb}]_{bb}$ ›
using *wfV-conspI subst-bb-commute*[*of bv b' bva ba b*] $*$ *wfV-conspI* **by** *metis*
qed
next
case (*wfTI z* Θ $\mathcal{B}'$ Γ' *b' c*)
show *?case* **proof**(*subst subst-tb.simps*, *rule Wellformed.wfTI*)
show *atom z* $\sharp$ (Θ, *B*, $\Gamma'[bv::=b]_{\Gamma b}$) **using** *wfTI subst-g-b-x-fresh* **by** *simp*

show Θ ; $B \vdash_{wf} b'[bv::=b]_{bb}$ **using** *wfTI* **by** *auto*
show Θ ; B ; $(z, b'[bv::=b]_{bb}, TRUE) \#_\Gamma \Gamma'[bv::=b]_{\Gamma b} \vdash_{wf} c[bv::=b]_{cb}$ **using** *wfTI* **by** *simp*
qed
next
case (*wfC-eqI* Θ $\mathcal{B}'$ Γ *e1* b' *e2*)
thus *?case* **using** *Wellformed.wfC-eqI subst-db.simps subst-cb.simps wfC-eqI* **by** *metis*
next
case (*wfG-nilI* Θ $\mathcal{B}'$)
then show *?case* **using** *Wellformed.wfG-nilI subst-gb.simps* **by** *simp*
next
case (*wfG-cons1I* c' Θ $\mathcal{B}'$ Γ' x b')
show *?case* **proof**(*subst subst-gb.simps*, *rule Wellformed.wfG-cons1I*)
show $c'[bv::=b]_{cb} \notin \{TRUE, FALSE\}$ **using** *wfG-cons1I*(*1*)
by(*nominal-induct* c' *rule*: *c.strong-induct*,*auto*+)
show Θ ; $B \vdash_{wf} \Gamma'[bv::=b]_{\Gamma b}$ **using** *wfG-cons1I* **by** *auto*
show *atom* $x \sharp \Gamma'[bv::=b]_{\Gamma b}$ **using** *wfG-cons1I subst-g-b-x-fresh* **by** *auto*
show Θ ; B ; $(x, b'[bv::=b]_{bb}, TRUE) \#_\Gamma \Gamma'[bv::=b]_{\Gamma b} \vdash_{wf} c'[bv::=b]_{cb}$ **using** *wfG-cons1I* **by** *auto*
show Θ ; $B \vdash_{wf} b'[bv::=b]_{bb}$ **using** *wfG-cons1I* **by** *auto*
qed
next
case (*wfG-cons2I* c' Θ $\mathcal{B}'$ Γ' x b')
show *?case* **proof**(*subst subst-gb.simps*, *rule Wellformed.wfG-cons2I*)
show $c'[bv::=b]_{cb} \in \{TRUE, FALSE\}$ **using** *wfG-cons2I* **by** *auto*
show Θ ; $B \vdash_{wf} \Gamma'[bv::=b]_{\Gamma b}$ **using** *wfG-cons2I* **by** *auto*
show *atom* $x \sharp \Gamma'[bv::=b]_{\Gamma b}$ **using** *wfG-cons2I subst-g-b-x-fresh* **by** *auto*
show Θ ; $B \vdash_{wf} b'[bv::=b]_{bb}$ **using** *wfG-cons2I* **by** *auto*
qed
next

case (*wfCE-valI* Θ $\mathcal{B}$ Γ v b)
then show *?case* **using** *subst-ceb.simps wf-intros wfX-wfY*
by (*metis wf-b-subst-lemmas wfCE-b-fresh*)
next
case (*wfCE-plusI* Θ $\mathcal{B}$ Γ *v1 v2*)
then show *?case* **using** *subst-bb.simps subst-ceb.simps wf-intros wfX-wfY*
by *metis*

next
case (*wfCE-leqI* Θ $\mathcal{B}$ Γ *v1 v2*)
then show *?case* **using** *subst-bb.simps subst-ceb.simps wf-intros wfX-wfY*
by *metis*

next
case (*wfCE-fstI* Θ $\mathcal{B}$ Γ *v1 b1 b2*)
then show *?case*
by (*metis* (*no-types*) *subst-bb.simps*(*5*) *subst-ceb.simps*(*3*) *wfCE-fstI.hyps*(*2*)
wfCE-fstI.prems(*1*) *wfCE-fstI.prems*(*2*) *Wellformed.wfCE-fstI*)

next
case (*wfCE-sndI* Θ $\mathcal{B}$ Γ *v1 b1 b2*)
then show *?case*

by (*metis* (*no-types*) *subst-bb.simps*(5) *subst-ceb.simps wfCE-sndI.hyps*(2)
wfCE-sndI wfCE-sndI.prems(2) *Wellformed.wfCE-sndI*)
next
case (*wfCE-concatI* Θ $\mathcal{B}$ Γ *v1 v2*)
then show *?case* **using** *subst-bb.simps subst-ceb.simps wf-intros wfX-wfY wf-b-subst-lemmas wfCE-b-fresh*

proof −
show *?thesis*
using *wfCE-concatI.hyps*(2) *wfCE-concatI.hyps*(4) *wfCE-concatI.prems*(1) *wfCE-concatI.prems*(2)

Wellformed.wfCE-concatI **by** *auto*
qed
next
case (*wfCE-lenI* Θ $\mathcal{B}$ Γ *v1*)
then show *?case* **using** *subst-bb.simps subst-ceb.simps wf-intros wfX-wfY wf-b-subst-lemmas wfCE-b-fresh*
by *metis*
qed(*auto simp add*: *wf-intros*)

lemma *wf-b-subst2*:
fixes Γ::Γ **and** Γ′::Γ **and** *v*::*v* **and** *e*::*e* **and** *c*::*c* **and** τ::τ **and** *ts*::(*string*∗τ) *list* **and** Δ::Δ **and** *b*::*b* **and** *ftq*::*fun-typ-q* **and** *ft*::*fun-typ* **and** *s*::*s* **and** *b′*::*b* **and** *ce*::*ce* **and** *td*::*type-def*
and *cs*::*branch-s* **and** *css*::*branch-list*
shows Θ ; Φ ; B' ; Γ ; Δ $\vdash_{wf}$ *e* : *b′* $\Longrightarrow$ {|*bv*|} = B' $\Longrightarrow$ Θ ; *B* $\vdash_{wf}$ *b* $\Longrightarrow$ Θ ; Φ ; *B* ; Γ[*bv*::=*b*]$_{\Gamma b}$; Δ[*bv*::=*b*]$_{\Delta b}$ $\vdash_{wf}$ *e*[*bv*::=*b*]$_{eb}$: *b′*[*bv*::=*b*]$_{bb}$ **and**
Θ ; Φ ; $\mathcal{B}$; Γ ; Δ $\vdash_{wf}$ *s* : *b* $\Longrightarrow$ *True* **and**
Θ ; Φ ; $\mathcal{B}$; Γ ; Δ ; *tid* ; *dc* ; *t* $\vdash_{wf}$ *cs* : *b* $\Longrightarrow$ *True* **and**
Θ ; Φ ; $\mathcal{B}$; Γ ; Δ ; *tid* ; *dclist* $\vdash_{wf}$ *css* : *b* $\Longrightarrow$ *True* **and**
Θ $\vdash_{wf}$ (Φ::Φ) $\Longrightarrow$ *True* **and**
Θ ; *B′* ; Γ $\vdash_{wf}$ Δ $\Longrightarrow$ {|*bv*|} = B' $\Longrightarrow$ Θ ; *B* $\vdash_{wf}$ *b* $\Longrightarrow$ Θ ; *B* ; Γ[*bv*::=*b*]$_{\Gamma b}$ $\vdash_{wf}$ Δ[*bv*::=*b*]$_{\Delta b}$ **and**
Θ ; Φ $\vdash_{wf}$ *ftq* $\Longrightarrow$ *True* **and**
Θ ; Φ ; $\mathcal{B}$ $\vdash_{wf}$ *ft* $\Longrightarrow$ *True*
proof(*nominal-induct*
b′ **and** *b* **and** *b* **and** *b* **and** Φ **and** Δ **and** *ftq* **and** *ft*
avoiding: *bv b B*
rule:*wfE-wfS-wfCS-wfCSS-wfPhi-wfD-wfFTQ-wfFT.strong-induct*)
case (*wfE-valI* Θ′ Φ′ $\mathcal{B}'$ Γ′ Δ′ *v′ b′*)
then show *?case* **unfolding** *subst-vb.simps subst-eb.simps* **using** *wf-b-subst1*(1) *Wellformed.wfE-valI*
by *auto*
next
case (*wfE-plusI* Θ Φ $\mathcal{B}$ Γ Δ *v1 v2*)
then show *?case* **unfolding** *subst-eb.simps*
using *wf-b-subst-lemmas wf-b-subst1*(1) *Wellformed.wfE-plusI*
proof −
have ∀ *b ba v g f ts*. ((*ts* ; *f* ; *g*[*bv*::=*ba*]$_{\Gamma b}$ $\vdash_{wf}$ *v*[*bv*::=*ba*]$_{vb}$: *b*[*bv*::=*ba*]$_{bb}$) ∨ ¬ *ts* ; $\mathcal{B}$; *g* $\vdash_{wf}$ *v* : *b*) ∨ ¬ *ts* ; *f* $\vdash_{wf}$ *ba*
using *wfE-plusI.prems*(1) *wf-b-subst1*(1) **by** *force*
then show Θ ; Φ ; *B* ; Γ[*bv*::=*b*]$_{\Gamma b}$; Δ[*bv*::=*b*]$_{\Delta b}$ $\vdash_{wf}$ [*plus v1*[*bv*::=*b*]$_{vb}$ *v2*[*bv*::=*b*]$_{vb}$]^{e} : *B-int*[*bv*::=*b*]$_{bb}$
by (*metis* (*full-types*) *wfE-plusI.hyps*(1) *wfE-plusI.hyps*(4) *wfE-plusI.hyps*(5) *wfE-plusI.hyps*(6)
wfE-plusI.prems(1) *wfE-plusI.prems*(2) *wfE-wfS-wfCS-wfCSS-wfPhi-wfD-wfFTQ-wfFT.wfE-plusI wf-b-subst-lemmas*(84

qed
next
case (*wfE-leqI* Θ Φ $\mathcal{B}$ Γ Δ *v1 v2*)
then show *?case* **unfolding** *subst-eb.simps*
using *wf-b-subst-lemmas*(*81*) *wf-b-subst1*(*1*) *Wellformed.wfE-leqI*
by (*metis wf-b-subst-lemmas*(*84*) *wf-b-subst-lemmas*(*85*))
next
case (*wfE-fstI* Θ Φ $\mathcal{B}$ Γ Δ *v1 b1 b2*)
then show *?case* **unfolding** *subst-eb.simps* **using** *wf-b-subst-lemmas*(*84*) *wf-b-subst1*(*1*) *Wellformed.wfE-fstI*

by (*metis wf-b-subst-lemmas*(*87*))

next
case (*wfE-sndI* Θ Φ $\mathcal{B}$ Γ Δ *v1 b1 b2*)
then show *?case* **unfolding** *subst-eb.simps* **using** *wf-b-subst-lemmas*(*86*) *wf-b-subst1*(*1*) *Wellformed.wfE-sndI*
by (*metis wf-b-subst-lemmas*(*87*))
next
case (*wfE-concatI* Θ Φ $\mathcal{B}$ Γ Δ *v1 v2*)
then show *?case* **unfolding** *subst-eb.simps* **using** *wf-b-subst-lemmas*(*86*) *wf-b-subst1*(*1*) *Wellformed.wfE-concatI*
by (*metis wf-b-subst-lemmas*(*89*))

next
case (*wfE-splitI* Θ Φ $\mathcal{B}$ Γ Δ *v1 v2*)
then show *?case* **unfolding** *subst-eb.simps* **using** *wf-b-subst-lemmas*(*86*) *wf-b-subst1*(*1*) *Wellformed.wfE-splitI*
by (*metis wf-b-subst-lemmas*(*84*) *wf-b-subst-lemmas*(*87*) *wf-b-subst-lemmas*(*89*))

next
case (*wfE-lenI* Θ Φ $\mathcal{B}$ Γ Δ *v1*)
then show *?case* **unfolding** *subst-eb.simps* **using** *wf-b-subst-lemmas*(*86*) *wf-b-subst1*(*1*) *Wellformed.wfE-lenI*
by (*metis wf-b-subst-lemmas*(*84*) *wf-b-subst-lemmas*(*89*))

next
case (*wfE-appI* Θ Φ $\mathcal{B}'$ Γ Δ *f x b' c* τ *s v*)
hence *bf*: *atom bv* $\sharp$ *b'* **using** *wfPhi-f-simple-wfT wfT-supp bv-not-in-dom-g wfPhi-f-simple-supp-b fresh-def* **by** *fast*
hence *bseq*: $b'[bv::=b]_{bb} = b'$ **using** *subst-bb.simps wf-b-subst-lemmas* **by** *metis*
have Θ ; Φ ; B ; $\Gamma[bv::=b]_{\Gamma b}$; $\Delta[bv::=b]_{\Delta b} \vdash_{wf}$ (*AE-app f* ($v[bv::=b]_{vb}$)) : (*b-of* ($\tau[bv::=b]_{\tau b}$))
proof
show $\Theta \vdash_{wf} \Phi$ **using** *wfE-appI* **by** *auto*
show Θ ; B ; $\Gamma[bv::=b]_{\Gamma b} \vdash_{wf} \Delta[bv::=b]_{\Delta b}$ **using** *wfE-appI* **by** *simp*
have *atom bv* $\sharp$ τ **using** *wfPhi-f-simple-wfT*[*OF wfE-appI*(*5*) *wfE-appI*(*1*), *THEN wfT-supp*] *bv-not-in-dom-g fresh-def* **by** *force*
hence $\tau[bv::=b]_{\tau b} = \tau$ **using** *forget-subst subst-b-τ-def* **by** *metis*
thus *Some* (*AF-fundef f* (*AF-fun-typ-none* (*AF-fun-typ x b' c* $\tau[bv::=b]_{\tau b}$ *s*))) = *lookup-fun* Φ *f* **using** *wfE-appI* **by** *simp*
show Θ ; B ; $\Gamma[bv::=b]_{\Gamma b} \vdash_{wf} v[bv::=b]_{vb}$: b' **using** *wfE-appI bseq wf-b-subst1* **by** *metis*

qed
then show *?case* **using** *subst-eb.simps b-of-subst-bb-commute* **by** *simp*
next

case (*wfE-appPI* Θ Φ $\mathcal{B}$ Γ Δ b' *bv1 v1* $\tau 1$ *f x1 b1 c1 s1*)
then have $*$: *atom bv* $\sharp$ *b1* **using** *wfPhi-f-supp*(*1*) *wfE-appPI*(*7*,*11*)
by (*metis fresh-def fresh-finsert singleton-iff subsetD fresh-def supp-at-base wfE-appPI.hyps*(*1*))
thm *Wellformed.wfE-appPI*
have Θ ; Φ ; B ; $\Gamma[bv::=b]_{\Gamma b}$; $\Delta[bv::=b]_{\Delta b} \vdash_{wf}$ *AE-appP f* $b'[bv::=b]_{bb}$ $(v1[bv::=b]_{vb})$: $(b\text{-}of\ \tau 1)[bv1::=b'[bv::=b]_{bb}]_b$
proof
show ‹ $\Theta \vdash_{wf} \Phi$ › **using** *wfE-appPI* **by** *auto*
show ‹ Θ ; B ; $\Gamma[bv::=b]_{\Gamma b} \vdash_{wf} \Delta[bv::=b]_{\Delta b}$ › **using** *wfE-appPI* **by** *auto*
show ‹ Θ ; $B \vdash_{wf} b'[bv::=b]_{bb}$ › **using** *wfE-appPI wf-b-subst1* **by** *auto*
have *atom bv1* $\sharp$ $\Gamma[bv::=b]_{\Gamma b}$ **using** *fresh-subst-if subst-b-*Γ*-def wfE-appPI* **by** *metis*
moreover have *atom bv1* $\sharp$ $b'[bv::=b]_{bb}$ **using** *fresh-subst-if subst-b-b-def wfE-appPI* **by** *metis*
moreover have *atom bv1* $\sharp$ $v1[bv::=b]_{vb}$ **using** *fresh-subst-if subst-b-v-def wfE-appPI* **by** *metis*
moreover have *atom bv1* $\sharp$ $\Delta[bv::=b]_{\Delta b}$ **using** *fresh-subst-if subst-b-*Δ*-def wfE-appPI* **by** *metis*
moreover have *atom bv1* $\sharp$ $(b\text{-}of\ \tau 1)[bv1::=b'[bv::=b]_{bb}]_{bb}$ **using** *fresh-subst-if subst-b-b-def wfE-appPI* **by** *metis*
ultimately show *atom bv1* $\sharp$ (Φ, Θ, B, $\Gamma[bv::=b]_{\Gamma b}$, $\Delta[bv::=b]_{\Delta b}$, $b'[bv::=b]_{bb}$, $v1[bv::=b]_{vb}$, $(b\text{-}of\ \tau 1)[bv1::=b'[bv::=b]_{bb}]_b$)
using *wfE-appPI* **using** *fresh-def fresh-prodN subst-b-b-def* **by** *metis*
show ‹*Some* (*AF-fundef f* (*AF-fun-typ-some bv1* (*AF-fun-typ x1 b1 c1* $\tau 1$ *s1*))) = *lookup-fun* Φ *f*› **using** *wfE-appPI* **by** *auto*

have ‹ Θ ; B ; $\Gamma[bv::=b]_{\Gamma b} \vdash_{wf} v1[bv::=b]_{vb}$: $b1[bv1::=b']_b[bv::=b]_{bb}$ ›
using *wfE-appPI subst-b-b-def* $*$ *wf-b-subst1* **by** *metis*
thus ‹ Θ ; B ; $\Gamma[bv::=b]_{\Gamma b} \vdash_{wf} v1[bv::=b]_{vb}$: $b1[bv1::=b'[bv::=b]_{bb}]_b$ ›
using *subst-bb-commute subst-b-b-def* $*$ **by** *auto*
qed
moreover have *atom bv* $\sharp$ *b-of* $\tau 1$ **proof** $-$
have *supp* (*b-of* $\tau 1$) $\subseteq$ { *atom bv1* } **using** *wfPhi-f-poly-supp-b-of-t*
using *b-of.simps wfE-appPI wfPhi-f-supp*(*5*) **by** *simp*
thus *?thesis* **using** *wfE-appPI*
fresh-def fresh-finsert singleton-iff subsetD fresh-def supp-at-base wfE-appPI.hyps **by** *metis*
qed
ultimately show *?case* **using** *subst-eb.simps*(*3*) *subst-bb-commute subst-b-b-def* $*$ **by** *simp*
next
case (*wfE-mvarI* Θ Φ $\mathcal{B}'$ Γ Δ *u* τ)

have Θ ; Φ ; B ; *subst-gb* Γ *bv b* ; *subst-db* Δ *bv b* $\vdash_{wf}$ (*AE-mvar u*)$[bv::=b]_{eb}$: $(b\text{-}of\ (\tau[bv::=b]_{\tau b}))$

proof(*subst subst-eb.simps*,*rule Wellformed.wfE-mvarI*)
show $\Theta \vdash_{wf} \Phi$ **using** *wfE-mvarI* **by** *simp*
show Θ ; B ; $\Gamma[bv::=b]_{\Gamma b} \vdash_{wf} \Delta[bv::=b]_{\Delta b}$ **using** *wfE-mvarI* **by** *metis*
show $(u, \tau[bv::=b]_{\tau b}) \in$ *setD* $\Delta[bv::=b]_{\Delta b}$
using *wfE-mvarI subst-db.simps set-insert subst-d-b-member* **by** *simp*
qed
thus *?case* **using** *b-of-subst-bb-commute* **by** *auto*

next

case (*wfS-seqI* Θ Φ $\mathcal{B}$ Γ Δ *s1 s2 b*)
then show *?case* **using** *subst-bb.simps wf-intros wfX-wfY* **by** *metis*

next
case (*wfD-emptyI* Θ $\mathcal{B}'$ Γ)
then show *?case* **using** *subst-db.simps Wellformed.wfD-emptyI wf-b-subst1* **by** *simp*
next
case (*wfD-cons* Θ $\mathcal{B}'$ Γ' Δ τ *u*)
show *?case* **proof**(*subst subst-db.simps*, *rule Wellformed.wfD-cons*)
show Θ ; B ; $\Gamma'[bv::=b]_{\Gamma b} \vdash_{wf} \Delta[bv::=b]_{\Delta b}$ **using** *wfD-cons* **by** *auto*
show Θ ; B ; $\Gamma'[bv::=b]_{\Gamma b} \vdash_{wf} \tau[bv::=b]_{\tau b}$ **using** *wfD-cons wf-b-subst1* **by** *auto*
show $u \notin$ *fst* ‘ *setD* $\Delta[bv::=b]_{\Delta b}$ **using** *wfD-cons subst-b-lookup-d* **by** *metis*
qed
next
case (*wfS-assertI* Θ Φ $\mathcal{B}$ *x c* Γ Δ *s b*)
show *?case* **by** *auto*
qed(*auto*)

lemmas *wf-b-subst* = *wf-b-subst1 wf-b-subst2*

lemma *wfT-subst-wfT*:
fixes τ::τ **and** b'::b **and** bv::bv
assumes Θ ; {|bv|} ; (x,b,c) #$_\Gamma$ *GNil* $\vdash_{wf} \tau$ **and** Θ ; $B \vdash_{wf} b'$
shows Θ ; B ; (x,$b[bv::=b']_{bb}$,$c[bv::=b']_{cb}$) #$_\Gamma$ *GNil* $\vdash_{wf}$ ($\tau[bv::=b']_{\tau b}$)
proof −
have Θ ; B ; ((x,b,c) #$_\Gamma$ *GNil*)$[bv::=b']_{\Gamma b} \vdash_{wf}$ ($\tau[bv::=b']_{\tau b}$)
using *wf-b-subst assms* **by** *metis*
thus *?thesis* **using** *subst-gb.simps wf-b-subst-lemmas wfCE-b-fresh* **by** *simp*
qed

lemma *wf-trans*:
fixes Γ::Γ **and** Γ'::Γ **and** v::v **and** e::e **and** c::c **and** τ::τ **and** ts::(*string*∗τ) *list* **and** Δ::Δ **and** b::b **and** *ftq*::*fun-typ-q* **and** *ft*::*fun-typ* **and** *ce*::*ce* **and** *td*::*type-def* **and** *s*::*s*
and *cs*::*branch-s* **and** *css*::*branch-list* **and** Θ::Θ
shows Θ ; $\mathcal{B}$; $\Gamma \vdash_{wf} v : b' \Longrightarrow \Gamma = (x, b, c2)$ #$_\Gamma$ $G \Longrightarrow \Theta$; $\mathcal{B}$; $(x, b, c1)$ #$_\Gamma$ $G \vdash_{wf} c2 \Longrightarrow \Theta$; $\mathcal{B}$; $(x, b, c1)$ #$_\Gamma$ $G \vdash_{wf} v : b'$ **and**
Θ ; $\mathcal{B}$; $\Gamma \vdash_{wf} c \Longrightarrow \Gamma = (x, b, c2)$ #$_\Gamma$ $G \Longrightarrow \Theta$; $\mathcal{B}$; $(x, b, c1)$ #$_\Gamma$ $G \vdash_{wf} c2 \Longrightarrow \Theta$; $\mathcal{B}$; $(x, b, c1)$ #$_\Gamma$ $G \vdash_{wf} c$ **and**
Θ ; $\mathcal{B} \vdash_{wf} \Gamma \Longrightarrow$ *True* **and**
Θ ; $\mathcal{B}$; $\Gamma \vdash_{wf} \tau \Longrightarrow$ *True* **and**
Θ ; $\mathcal{B}$; $\Gamma \vdash_{wf} ts \Longrightarrow$ *True* **and**
$\vdash_{wf} \Theta \Longrightarrow$ *True* **and**
Θ ; $\mathcal{B} \vdash_{wf} b \Longrightarrow$ *True* **and**
Θ ; $\mathcal{B}$; $\Gamma \vdash_{wf} ce : b' \Longrightarrow \Gamma = (x, b, c2)$ #$_\Gamma$ $G \Longrightarrow \Theta$; $\mathcal{B}$; $(x, b, c1)$ #$_\Gamma$ $G \vdash_{wf} c2 \Longrightarrow \Theta$; $\mathcal{B}$; $(x, b, c1)$ #$_\Gamma$ $G \vdash_{wf} ce : b'$ **and**
$\Theta \vdash_{wf} td \Longrightarrow$ *True*
proof(*nominal-induct*
b' **and** c **and** Γ **and** τ **and** *ts* **and** Θ **and** b **and** b' **and** *td*
avoiding: *c1*
arbitrary: Γ_1 **and** Γ_1 **and** Γ_1 **and** Γ_1 **and** Γ_1 **and** Γ_1 **and** Γ_1 **and** Γ_1 **and** Γ_1 **and** Γ_1 **and** Γ_1 **and** Γ_1 **and** Γ_1 **and** Γ_1 **and** Γ_1
rule:*wfV-wfC-wfG-wfT-wfTs-wfTh-wfB-wfCE-wfTD.strong-induct*)

case (*wfV-varI* Θ $\mathcal{B}$ Γ b' c' x')
have *wbg*: Θ ; $\mathcal{B}$ $\vdash_{wf}$ (x, b, *c1*) $\#_\Gamma$ G **using** *wfC-wf wfV-varI* **by** *simp*
show *?case* **proof**(*cases* x=x')
case *True*
have *Some* (b', *c1*) = *lookup* ((x, b, *c1*) $\#_\Gamma$ G) x' **using** *lookup.simps wfV-varI* **using** *True* **by** *auto*
then show *?thesis* **using** *Wellformed.wfV-varI wbg* **by** *simp*
next
case *False*
then have *Some* (b', c') = *lookup* ((x, b, *c1*) $\#_\Gamma$ G) x' **using** *lookup.simps wfV-varI*
by *simp*
then show *?thesis* **using** *Wellformed.wfV-varI wbg* **by** *simp*
qed
next
case (*wfV-conspI s bv dclist* Θ *dc x1* b' *c* $\mathcal{B}$ *b1* Γ *v*)
show *?case* **proof**
show ‹*AF-typedef-poly s bv dclist* ∈ *set* Θ› **using** *wfV-conspI* **by** *auto*
show ‹(*dc*, {| *x1* : b' | *c* |}) ∈ *set dclist*› **using** *wfV-conspI* **by** *auto*
show ‹ Θ ; $\mathcal{B}$ $\vdash_{wf}$ *b1* › **using** *wfV-conspI* **by** *auto*
show ‹*atom bv* ♯ (Θ, $\mathcal{B}$, (x, b, *c1*) $\#_\Gamma$ G, *b1*, *v*)› **unfolding** *fresh-prodN fresh-GCons* **using** *wfV-conspI fresh-prodN fresh-GCons* **by** *simp*
show ‹ Θ ; $\mathcal{B}$; (x, b, *c1*) $\#_\Gamma$ G $\vdash_{wf}$ *v* : b'[*bv*::=*b1*]$_{bb}$ › **using** *wfV-conspI* **by** *auto*
qed
qed((*auto* | *metis wfC-wf wf-intros*) +)

end

Chapter 9

Type System

9.1 Subtyping

Subtyping is defined on top of SMT logic. A subtyping check is converted into an SMT validity check.

inductive *subtype* :: $\Theta \Rightarrow \mathcal{B} \Rightarrow \Gamma \Rightarrow \tau \Rightarrow \tau \Rightarrow$ *bool* (- ; - ; - $\vdash$ - $\lesssim$ - [*50*, *50*, *50*] *50*) **where**
subtype-baseI: ⟦
  atom $x \sharp (\Theta, \mathcal{B}, \Gamma, z, c, z', c')$;
  Θ ; $\mathcal{B}$; $\Gamma \vdash_{wf} \{\!|\, z : b \mid c \,|\!\}$;
  Θ ; $\mathcal{B}$; $\Gamma \vdash_{wf} \{\!|\, z' : b \mid c' \,|\!\}$;
  Θ ; $\mathcal{B}$; $(x, b, c[z::=[x]^v]_v)$ $\#_\Gamma$ $\Gamma \models c'[z'::=[x]^v]_v$
⟧ ⟹
  Θ ; $\mathcal{B}$; $\Gamma \vdash \{\!|\, z : b \mid c \,|\!\} \lesssim \{\!|\, z' : b \mid c' \,|\!\}$

equivariance *subtype*
nominal-inductive *subtype*
  avoids *subtype-baseI*: *x*
proof(*goal-cases*)
  case (*1* Θ $\mathcal{B}$ Γ *z b c z' c' x*)
  then show *?case* **using** *fresh-star-def 1* **by** *force*
next
  case (*2* Θ $\mathcal{B}$ Γ *z b c z' c' x*)
  then show *?case* **by** *auto*
qed

inductive-cases *subtype-elims*:
  Θ ; $\mathcal{B}$; $\Gamma \vdash \{\!|\, z : b \mid c \,|\!\} \lesssim \{\!|\, z' : b \mid c' \,|\!\}$
  Θ ; $\mathcal{B}$; $\Gamma \vdash \tau_1 \lesssim \tau_2$

9.2 Literals

The type synthesised has the constraint that z equates to the literal

inductive *infer-l* :: $l \Rightarrow \tau \Rightarrow$ *bool* ($\vdash$ - $\Rightarrow$ - [*50*, *50*] *50*) **where**
  infer-trueI: $\vdash$ *L-true* $\Rightarrow \{\!|\, z : \textit{B-bool} \mid [[z]^v]^{ce} == [[\textit{L-true}]^v]^{ce} \,|\!\}$
| *infer-falseI*: $\vdash$ *L-false* $\Rightarrow \{\!|\, z : \textit{B-bool} \mid [[z]^v]^{ce} == [[\textit{L-false}]^v]^{ce} \,|\!\}$
| *infer-natI*: $\vdash$ *L-num n* $\Rightarrow \{\!|\, z : \textit{B-int} \mid [[z]^v]^{ce} == [[\textit{L-num } n]^v]^{ce} \,|\!\}$

| *infer-unitI*: $\vdash$ *L-unit* $\Rightarrow$ {| z : *B-unit* | $[[z]^v]^{ce} == [[\text{L-unit}]^v]^{ce}$ |}
| *infer-bitvecI*: $\vdash$ *L-bitvec bv* $\Rightarrow$ {| z : *B-bitvec* | $[[z]^v]^{ce} == [[\text{L-bitvec } bv]^v]^{ce}$ |}

nominal-inductive *infer-l* **.**
equivariance *infer-l*

inductive-cases *infer-l-elims*[*elim*!]:
$\vdash$ *L-true* $\Rightarrow \tau$
$\vdash$ *L-false* $\Rightarrow \tau$
$\vdash$ *L-num n* $\Rightarrow \tau$
$\vdash$ *L-unit* $\Rightarrow \tau$
$\vdash$ *L-bitvec x* $\Rightarrow \tau$
$\vdash l \Rightarrow \tau$

lemma *infer-l-form2*[*simp*]:
shows $\exists z. \vdash l \Rightarrow (\{| z : \textit{base-for-lit } l \mid [[z]^v]^{ce} == [[l]^v]^{ce} |\})$
proof (*nominal-induct l rule*: *l.strong-induct*)
case (*L-num x*)
then show *?case* **using** *infer-l.intros base-for-lit.simps has-fresh-z* **by** *metis*
next
case *L-true*
then show *?case* **using** *infer-l.intros base-for-lit.simps has-fresh-z* **by** *metis*
next
case *L-false*
then show *?case* **using** *infer-l.intros base-for-lit.simps has-fresh-z* **by** *metis*
next
case *L-unit*
then show *?case* **using** *infer-l.intros base-for-lit.simps has-fresh-z* **by** *metis*
next
case (*L-bitvec x*)
then show *?case* **using** *infer-l.intros base-for-lit.simps has-fresh-z* **by** *metis*
qed

9.3 Values

inductive *infer-v* :: $\Theta \Rightarrow \mathcal{B} \Rightarrow \Gamma \Rightarrow v \Rightarrow \tau \Rightarrow$ *bool* (- ; - ; - $\vdash$ - $\Rightarrow$ - [*50*, *50*, *50*] *50*) **where**

infer-v-varI: $[\![$
Θ ; $\mathcal{B} \vdash_{wf} \Gamma$;
Some (*b*,*c*) = *lookup* Γ *x*;
atom z $\sharp$ *x* ; *atom z* $\sharp$ Γ
$]\!] \Longrightarrow$
Θ ; $\mathcal{B}$; Γ $\vdash [x]^v \Rightarrow$ {| z : b | $[[z]^v]^{ce} == [[x]^v]^{ce}$ |}

| *infer-v-litI*: $[\![$
Θ ; $\mathcal{B} \vdash_{wf} \Gamma$;
$\vdash l \Rightarrow \tau$
$]\!] \Longrightarrow$
Θ ; $\mathcal{B}$; $\Gamma \vdash [l]^v \Rightarrow \tau$

| *infer-v-pairI*: $[\![$

$atom\ z \sharp (v1,\ v2);\ atom\ z \sharp \Gamma;$
$\Theta\ ;\ \mathcal{B}\ ;\ \Gamma \vdash (v1::v) \Rightarrow (\{\!|\ z1 : b1 \mid c1\ |\!\})\ ;$
$\Theta\ ;\ \mathcal{B}\ ;\ \Gamma \vdash (v2::v) \Rightarrow (\{\!|\ z2 : b2 \mid c2\ |\!\})$
$\rrbracket \Longrightarrow$
$\Theta\ ;\ \mathcal{B}\ ;\ \Gamma \vdash V\text{-}pair\ v1\ v2 \Rightarrow (\{\!|\ z : B\text{-}pair\ b1\ b2\ \mid [[z]^v]^{ce} == [[v1,v2]^v]^{ce}\ |\!\})$

| *infer-v-consI*: ⟦
$AF\text{-}typedef\ s\ dclist \in set\ \Theta;$
$(dc,\ \{\!|\ x : b\ \mid c\ |\!\}) \in set\ dclist\ ;$
$\Theta\ ;\ \mathcal{B}\ ;\ \Gamma \vdash v \Rightarrow (\{\!|\ z' : b \mid c'\ |\!\})\ ;$
$\Theta\ ;\ \mathcal{B}\ ;\ \Gamma \vdash \{\!|\ z' : b \mid c'\ |\!\} \lesssim \{\!|\ x : b \mid c\ |\!\}\ ;$
$atom\ z \sharp v\ ;\ atom\ z \sharp \Gamma$
⟧ ⟹
$\Theta\ ;\ \mathcal{B}\ ;\ \Gamma\ \vdash V\text{-}cons\ s\ dc\ v \Rightarrow (\{\!|\ z : B\text{-}id\ s\ \mid\ [[z]^v]^{ce} == [\ V\text{-}cons\ s\ dc\ v\ ]^{ce}\ |\!\})$

| *infer-v-conspI*: ⟦
$AF\text{-}typedef\text{-}poly\ s\ bv\ dclist \in set\ \Theta;$
$(dc,\ tc) \in set\ dclist\ ;$
$\Theta\ ;\ \mathcal{B}\ ;\ \Gamma \vdash v \Rightarrow tv;$
$\Theta\ ;\ \mathcal{B}\ ;\ \Gamma \vdash tv \lesssim tc[bv::=b]_{\tau b}\ ;$
$atom\ z \sharp (\Theta,\ \mathcal{B},\ \Gamma,\ v,\ b);$
$atom\ bv \sharp (\Theta,\ \mathcal{B},\ \Gamma,\ v,\ b);$
$\Theta\ ;\ \mathcal{B}\ \vdash_{wf} b$
⟧ ⟹
$\Theta\ ;\ \mathcal{B}\ ;\ \Gamma\ \vdash V\text{-}consp\ s\ dc\ b\ v \Rightarrow (\{\!|\ z : B\text{-}app\ s\ b\ \mid\ [[z]^v]^{ce} == (CE\text{-}val\ (V\text{-}consp\ s\ dc\ b\ v))\ |\!\})$

equivariance *infer-v*
nominal-inductive *infer-v*
avoids *infer-v-conspI*: *bv* **and** *z*
proof(*goal-cases*)
case (*1 s bv dclist* Θ *dc tc* $\mathcal{B}$ Γ *v tv b z*)
hence $atom\ bv \sharp V\text{-}consp\ s\ dc\ b\ v$ **using** *v.fresh fresh-prodN pure-fresh* **by** *metis*
moreover then have $atom\ bv \sharp \{\!|\ z : B\text{-}id\ s\ \mid [\ [\ z\ ]^v\ ]^{ce}\ ==\ [\ V\text{-}consp\ s\ dc\ b\ v\ ]^{ce}\ |\!\}$
using *τ.fresh ce.fresh v.fresh* **by** *auto*
moreover have $atom\ z \sharp V\text{-}consp\ s\ dc\ b\ v$ **using** *v.fresh fresh-prodN pure-fresh 1* **by** *metis*
moreover then have $atom\ z \sharp \{\!|\ z : B\text{-}id\ s\ \mid [\ [\ z\ ]^v\ ]^{ce}\ ==\ [\ V\text{-}consp\ s\ dc\ b\ v\ ]^{ce}\ |\!\}$
using *τ.fresh ce.fresh v.fresh* **by** *auto*
ultimately show *?case* **using** *fresh-star-def 1* **by** *force*
next
case (*2 s bv dclist* Θ *dc tc* $\mathcal{B}$ Γ *v tv b z*)
then show *?case* **by** *auto*
qed

inductive-cases *infer-v-elims*[*elim*!]:
$\Theta\ ;\ \mathcal{B}\ ;\ \Gamma \vdash V\text{-}var\ x \Rightarrow \tau$
$\Theta\ ;\ \mathcal{B}\ ;\ \Gamma \vdash V\text{-}lit\ l \Rightarrow \tau$
$\Theta\ ;\ \mathcal{B}\ ;\ \Gamma \vdash V\text{-}pair\ v1\ v2 \Rightarrow \tau$
$\Theta\ ;\ \mathcal{B}\ ;\ \Gamma \vdash V\text{-}cons\ s\ dc\ v \Rightarrow \tau$
$\Theta\ ;\ \mathcal{B}\ ;\ \Gamma \vdash V\text{-}pair\ v1\ v2 \Rightarrow (\{\!|\ z : b\ \mid\ c\ |\!\})$
$\Theta\ ;\ \mathcal{B}\ ;\ \Gamma \vdash V\text{-}pair\ v1\ v2 \Rightarrow (\{\!|\ z : [\ b1\ ,\ b2\ ]^b\ \mid\ [[z]^v]^{ce} == [[v1,v2]^v]^{ce}\ |\!\})$
$\Theta\ ;\ \mathcal{B}\ ;\ \Gamma \vdash V\text{-}consp\ s\ dc\ b\ v\ \Rightarrow \tau$

9.4 Introductions

inductive *check-v* :: $\Theta \Rightarrow \mathcal{B} \Rightarrow \Gamma \Rightarrow v \Rightarrow \tau \Rightarrow$ *bool* (- ; - ; - $\vdash$ - $\Leftarrow$ - [*50*, *50*, *50*] *50*) **where**
check-v-subtypeI: $[\![\ \Theta\ ;\ \mathcal{B}\ ;\ \Gamma \vdash \tau 1 \lesssim \tau 2;\ \Theta\ ;\ \mathcal{B}\ ;\ \Gamma \vdash v \Rightarrow \tau 1\ ]\!] \Longrightarrow \Theta\ ;\ \mathcal{B}\ ;\ \ \Gamma \vdash\ \ v \Leftarrow \tau 2$
equivariance *check-v*
nominal-inductive *check-v* **.**

inductive-cases *check-v-elims*[*elim*!]:
$\Theta;\ \mathcal{B}\ ;\ \Gamma \vdash v \Leftarrow \tau$

9.5 Expressions

Type synthesis for expressions

inductive *infer-e* :: $\Theta \Rightarrow \Phi \Rightarrow \mathcal{B} \Rightarrow \Gamma \Rightarrow \Delta \Rightarrow e \Rightarrow \tau \Rightarrow$ *bool* (- ; - ; - ; - ; - $\vdash$ - $\Rightarrow$ - [*50*, *50*, *50*,*50*] *50*) **where**

infer-e-valI: $[\![$
$(\Theta\ ;\ \mathcal{B}\ ;\Gamma \vdash_{wf} \Delta)\ ;$
$(\Theta\ \vdash_{wf} (\Phi::\Phi))\ ;$
$(\Theta\ ;\ \mathcal{B}\ ;\ \Gamma \vdash v \Rightarrow \tau)\ ]\!] \Longrightarrow$
$\Theta\ ;\ \Phi\ ;\ \mathcal{B}\ ;\ \Gamma\ ;\ \Delta \vdash (AE\text{-}val\ v) \Rightarrow \tau$

| *infer-e-plusI*: $[\![$
$\Theta\ ;\ \mathcal{B}\ ;\Gamma \vdash_{wf} \Delta\ ;$
$\Theta\ \vdash_{wf} (\Phi::\Phi)\ ;$
$\Theta\ ;\ \mathcal{B}\ ;\ \Gamma \vdash v1 \Rightarrow \{\!|\ z1 : B\text{-}int \mid c1\ |\!\}\ ;$
$\Theta\ ;\ \mathcal{B}\ ;\ \Gamma \vdash\ \ v2 \Rightarrow \{\!|\ z2 : B\text{-}int \mid c2\ |\!\};$
$atom\ z3 \sharp (AE\text{-}op\ Plus\ v1\ v2);\ atom\ z3 \sharp \Gamma\ ]\!] \Longrightarrow$
$\Theta\ ;\ \Phi\ ;\ \mathcal{B}\ ;\ \Gamma\ ;\ \Delta \vdash AE\text{-}op\ Plus\ v1\ v2 \Rightarrow \{\!|\ z3 : B\text{-}int \mid [[z3]^v]^{ce} == (CE\text{-}op\ Plus\ [v1]^{ce}\ [v2]^{ce})\ |\!\}$

| *infer-e-leqI*: $[\![$
$\Theta\ ;\ \mathcal{B}\ ;\Gamma \vdash_{wf} \Delta;$
$\Theta\ \vdash_{wf} (\Phi::\Phi)\ ;$
$\Theta\ ;\ \mathcal{B}\ ;\ \Gamma \vdash v1 \Rightarrow \{\!|\ z1 : B\text{-}int \mid c1\ |\!\}\ ;$
$\Theta\ ;\ \mathcal{B}\ ;\ \Gamma \vdash v2 \Rightarrow \{\!|\ z2 : B\text{-}int \mid c2\ |\!\};$
$atom\ z3 \sharp (AE\text{-}op\ LEq\ v1\ v2);\ atom\ z3 \sharp \Gamma$
$]\!] \Longrightarrow$
$\Theta\ ;\ \Phi\ ;\ \mathcal{B}\ ;\ \Gamma\ ;\ \Delta \vdash AE\text{-}op\ LEq\ v1\ v2 \Rightarrow\ \{\!|\ z3 : B\text{-}bool \mid [[z3]^v]^{ce} == (CE\text{-}op\ LEq\ [v1]^{ce}\ [v2]^{ce})\ |\!\}$

| *infer-e-appI*: $[\![$
$\Theta\ ;\ \mathcal{B}\ ;\Gamma \vdash_{wf} \Delta\ ;$
$\Theta\ \vdash_{wf} (\Phi::\Phi)\ ;$
$Some\ (AF\text{-}fundef\ f\ (AF\text{-}fun\text{-}typ\text{-}none\ (AF\text{-}fun\text{-}typ\ x\ b\ c\ \tau'\ s'))) = lookup\text{-}fun\ \Phi\ f;$
$\Theta\ ;\ \mathcal{B}\ ;\ \Gamma \vdash v \Leftarrow \{\!|\ \ x : b \mid c\ |\!\};\ atom\ x \sharp \Gamma;$
$\tau'[x::=v]_v = \tau$
$]\!] \Longrightarrow$
$\Theta\ ;\ \Phi\ ;\ \mathcal{B}\ ;\ \Gamma\ ;\ \Delta \vdash AE\text{-}app\ f\ v \Rightarrow \tau$

| *infer-e-appPI*: $[\![$
$\Theta\ ;\ \mathcal{B}\ ;\Gamma \vdash_{wf} \Delta\ ;$
$\Theta\ \vdash_{wf} (\Phi::\Phi)\ ;$

Θ ; $\mathcal{B} \vdash_{wf} b'$;
Some (*AF-fundef f* (*AF-fun-typ-some bv* (*AF-fun-typ x b c* τ' s'))) = *lookup-fun* Φ *f*;
Θ ; $\mathcal{B}$; $\Gamma \vdash v \Leftarrow \{\!|\ x : b[bv::=b']_b \mid c[bv::=b']_b\ |\!\}$; *atom x* $\sharp$ Γ;
$(\tau'[bv::=b']_b[x::=v]_v) = \tau$;
atom bv $\sharp$ (Θ, Φ, $\mathcal{B}$, Γ, Δ, b', v, τ)
$]\!] \Longrightarrow$
Θ ; Φ ; $\mathcal{B}$; Γ ; $\Delta \vdash$ *AE-appP f b' v* $\Rightarrow \tau$

| *infer-e-fstI*: $[\![$
Θ ; $\mathcal{B}$;$\Gamma \vdash_{wf} \Delta$;
$\Theta \vdash_{wf} (\Phi::\Phi)$;
Θ ; $\mathcal{B}$; $\Gamma \vdash v \Rightarrow \{\!|\ z' : [b1,b2]^b \mid c\ |\!\}$;
atom z $\sharp$ *AE-fst v* ; *atom z* $\sharp$ Γ $]\!] \Longrightarrow$
Θ ; Φ ; $\mathcal{B}$; Γ ; $\Delta \vdash$ *AE-fst v* $\Rightarrow \{\!|\ z : b1 \mid [[z]^v]^{ce} == ((CE\text{-}fst\ [v]^{ce}))\ |\!\}$

| *infer-e-sndI*: $[\![$
Θ ; $\mathcal{B}$;$\Gamma \vdash_{wf} \Delta$;
$\Theta \vdash_{wf} (\Phi::\Phi)$;
Θ ; $\mathcal{B}$; $\Gamma \vdash v \Rightarrow \{\!|\ z' : [\ b1,\ b2]^b \mid c\ |\!\}$;
atom z $\sharp$ *AE-snd v* ; *atom z* $\sharp$ Γ $]\!] \Longrightarrow$
Θ ; Φ ; $\mathcal{B}$; Γ ; $\Delta \vdash$ *AE-snd v* $\Rightarrow \{\!|\ z : b2 \mid [[z]^v]^{ce} == ((CE\text{-}snd\ [v]^{ce}))\ |\!\}$

| *infer-e-lenI*: $[\![$
Θ ; $\mathcal{B}$;$\Gamma \vdash_{wf} \Delta$;
$\Theta \vdash_{wf} (\Phi::\Phi)$;
Θ ; $\mathcal{B}$; $\Gamma \vdash v \Rightarrow \{\!|\ z' : B\text{-}bitvec \mid c\ |\!\}$;
atom z $\sharp$ *AE-len v* ; *atom z* $\sharp$ $\Gamma]\!] \Longrightarrow$
Θ ; Φ ; $\mathcal{B}$; Γ ; $\Delta \vdash$ *AE-len v* $\Rightarrow \{\!|\ z : B\text{-}int \mid [[z]^v]^{ce} == ((CE\text{-}len\ [v]^{ce}))\ |\!\}$

| *infer-e-mvarI*: $[\![$
Θ ; $\mathcal{B} \vdash_{wf} \Gamma$;
$\Theta \vdash_{wf} (\Phi::\Phi)$;
Θ ; $\mathcal{B}$; $\Gamma \vdash_{wf} \Delta$;
$(u,\tau) \in$ *setD* Δ $]\!] \Longrightarrow$
Θ ; Φ ; $\mathcal{B}$; Γ ; $\Delta \vdash$ *AE-mvar u* $\Rightarrow \tau$

| *infer-e-concatI*: $[\![$
Θ ; $\mathcal{B}$;$\Gamma \vdash_{wf} \Delta$;
$\Theta \vdash_{wf} (\Phi::\Phi)$;
Θ ; $\mathcal{B}$; $\Gamma \vdash$ *v1* $\Rightarrow \{\!|$ *z1* : *B-bitvec* $\mid$ *c1* $|\!\}$;
Θ ; $\mathcal{B}$; $\Gamma \vdash$ *v2* $\Rightarrow \{\!|$ *z2* : *B-bitvec* $\mid$ *c2* $|\!\}$;
atom z3 $\sharp$ (*AE-concat v1 v2*); *atom z3* $\sharp$ Γ $]\!] \Longrightarrow$
Θ ; Φ ; $\mathcal{B}$; Γ ; $\Delta \vdash$ *AE-concat v1 v2* $\Rightarrow \{\!|$ *z3* : *B-bitvec* $\mid [[z3]^v]^{ce} == (CE\text{-}concat\ [v1]^{ce}\ [v2]^{ce})\ |\!\}$

| *infer-e-splitI*: $[\![$
Θ ; $\mathcal{B}$;$\Gamma \vdash_{wf} \Delta$;
$\Theta \vdash_{wf} (\Phi::\Phi)$;
infer-v Θ $\mathcal{B}$ Γ *v1* $\{\!|$ *z1* : *B-bitvec* $\mid$ *c1* $|\!\}$;
check-v Θ $\mathcal{B}$ Γ *v2* $\{\!|$ *z2* : *B-int* $\mid$ (*CE-op LEq* (*CE-val* (*V-lit* (*L-num 0*))) (*CE-val* (*V-var z2*))) == (*CE-val* (*V-lit L-true*)) *AND*
(*CE-op LEq* (*CE-val* (*V-var z2*)) (*CE-len* (*CE-val* (*v1*)))) == (*CE-val*

$(V\text{-}lit\ L\text{-}true))\ \}$;
$atom\ z1\ \sharp\ (AE\text{-}split\ v1\ v2)$; $atom\ z1\ \sharp\ \Gamma$;
$atom\ z2\ \sharp\ (AE\text{-}split\ v1\ v2)$; $atom\ z2\ \sharp\ \Gamma$;
$atom\ z3\ \sharp\ (AE\text{-}split\ v1\ v2)$; $atom\ z3\ \sharp\ \Gamma$
$\rrbracket \Longrightarrow$
$infer\text{-}e\ \Theta\ \Phi\ \mathcal{B}\ \Gamma\ \ \Delta\ (AE\text{-}split\ \ v1\ v2)\ \{\!|\ z3 : B\text{-}pair\ B\text{-}bitvec\ B\text{-}bitvec\ \mid$
$((CE\text{-}val\ v1) == (CE\text{-}concat\ (CE\text{-}fst\ (CE\text{-}val\ (V\text{-}var\ z3)))\ (CE\text{-}snd\ (CE\text{-}val\ (V\text{-}var\ z3)))))$
$AND\ (((CE\text{-}len\ (CE\text{-}fst\ (CE\text{-}val\ (V\text{-}var\ z3))))) == (CE\text{-}val\ (\ v2))) \ |\!\}$

equivariance *infer-e*
nominal-inductive *infer-e*
avoids *infer-e-appPI*: *bv* | *infer-e-splitI*: *z3* **and** *z1* **and** *z2*
proof(*goal-cases*)
case (*1* $\Theta\ \mathcal{B}\ \Gamma\ \Delta\ \Phi\ b'\ f\ bv\ x\ b\ c\ \tau'\ s'\ v\ \tau$)
moreover hence $atom\ bv\ \sharp\ AE\text{-}appP\ f\ b'\ v$ **using** *fresh-prodN pure-fresh e.fresh* **by** *force*
ultimately show *?case* **unfolding** *fresh-star-def* **using** *fresh-prodN e.fresh pure-fresh fresh-Pair* **by** *auto*
next
case (*2* $\Theta\ \mathcal{B}\ \Gamma\ \Delta\ \Phi\ b'\ f\ bv\ x\ b\ c\ \tau'\ s'\ v\ \tau$)
then show *?case* **by** *auto*
next
case (*3* $\Theta\ \mathcal{B}\ \Gamma\ \Delta\ \Phi\ v1\ z1\ c1\ v2\ z2\ z3$)
have $atom\ z3\ \sharp\ \{\!|\ z3 : [\ B\text{-}bitvec\ ,\ B\text{-}bitvec\ ]^b\ \mid [\ v1\ ]^{ce} == [\ [\#1[\ [\ z3\ ]^v\ ]^{ce}]^{ce}\ @@\ [\#2[\ [\ z3\ ]^v\ ]^{ce}]^{ce}\ ]^{ce}\ \ AND\ [|\ [\#1[\ [\ z3\ ]^v\ ]^{ce}]^{ce}\ |]^{ce} == [\ v2\ ]^{ce}\ \ |\!\}$
using $\tau.fresh$ **by** *simp*
then show *?case* **unfolding** *fresh-star-def fresh-prod7* **using** *wfG-fresh-x2 3* **by** *auto*
next
case (*4* $\Theta\ \mathcal{B}\ \Gamma\ \Delta\ \Phi\ v1\ z1\ c1\ v2\ z2\ z3$)
then show *?case* **by** *auto*
qed

inductive-cases *infer-e-elims*[*elim*!]:
$\Theta\ ;\ \Phi\ ;\ \mathcal{B}\ ;\ \Gamma\ ;\ \Delta \vdash (AE\text{-}op\ Plus\ v1\ v2) \Rightarrow \{\!|\ z3 : B\text{-}int \mid [[z3]^v]^{ce} == (CE\text{-}op\ Plus\ [v1]^{ce}\ [v2]^{ce})\ |\!\}$
$\Theta\ ;\ \Phi\ ;\ \mathcal{B}\ ;\ \Gamma\ ;\ \Delta \vdash (AE\text{-}op\ LEq\ v1\ v2) \Rightarrow \{\!|\ z3 : B\text{-}bool \mid [[z3]^v]^{ce} == (CE\text{-}op\ LEq\ [v1]^{ce}\ [v2]^{ce})\ |\!\}$
$\Theta\ ;\ \Phi\ ;\ \mathcal{B}\ ;\ \Gamma\ ;\ \Delta \vdash (AE\text{-}op\ Plus\ v1\ v2) \Rightarrow \{\!|\ z3 : B\text{-}int \mid c\ |\!\}$
$\Theta\ ;\ \Phi\ ;\ \mathcal{B}\ ;\ \Gamma\ ;\ \Delta \vdash (AE\text{-}op\ Plus\ v1\ v2) \Rightarrow \{\!|\ z3 : b \mid c\ |\!\}$
$\Theta\ ;\ \Phi\ ;\ \mathcal{B}\ ;\ \Gamma\ ;\ \Delta \vdash (AE\text{-}op\ LEq\ v1\ v2) \Rightarrow \{\!|\ z3 : b \mid c\ |\!\}$
$\Theta\ ;\ \Phi\ ;\ \mathcal{B}\ ;\ \Gamma\ ;\ \Delta \vdash (AE\text{-}app\ f\ v\ ) \Rightarrow \tau$
$\Theta\ ;\ \Phi\ ;\ \mathcal{B}\ ;\ \Gamma\ ;\ \Delta \vdash (AE\text{-}val\ v) \Rightarrow \tau$
$\Theta\ ;\ \Phi\ ;\ \mathcal{B}\ ;\ \Gamma\ ;\ \Delta \vdash (AE\text{-}fst\ v) \Rightarrow \tau$
$\Theta\ ;\ \Phi\ ;\ \mathcal{B}\ ;\ \Gamma\ ;\ \Delta \vdash (AE\text{-}snd\ v) \Rightarrow \tau$
$\Theta\ ;\ \Phi\ ;\ \mathcal{B}\ ;\ \Gamma\ ;\ \Delta \vdash (AE\text{-}mvar\ u) \Rightarrow \tau$
$\Theta\ ;\ \Phi\ ;\ \mathcal{B}\ ;\ \Gamma\ ;\ \Delta \vdash (AE\text{-}op\ Plus\ v1\ v2) \Rightarrow \tau$
$\Theta\ ;\ \Phi\ ;\ \mathcal{B}\ ;\ \Gamma\ ;\ \Delta \vdash (AE\text{-}op\ LEq\ v1\ v2) \Rightarrow \tau$
$\Theta\ ;\ \Phi\ ;\ \mathcal{B}\ ;\ \Gamma\ ;\ \Delta \vdash (AE\text{-}op\ LEq\ v1\ v2) \Rightarrow \{\!|\ z3 : B\text{-}bool \mid c\ |\!\}$
$\Theta\ ;\ \Phi\ ;\ \mathcal{B}\ ;\ \Gamma\ ;\ \Delta \vdash (AE\text{-}app\ f\ v\ ) \Rightarrow \tau[x::=v]_{\tau v}$
$\Theta\ ;\ \Phi\ ;\ \mathcal{B}\ ;\ \Gamma\ ;\ \Delta \vdash (AE\text{-}op\ opp\ v1\ v2) \Rightarrow \tau$
$\Theta\ ;\ \Phi\ ;\ \mathcal{B}\ ;\ \Gamma\ ;\ \Delta \vdash (AE\text{-}len\ v) \Rightarrow \tau$
$\Theta\ ;\ \Phi\ ;\ \mathcal{B}\ ;\ \Gamma\ ;\ \Delta \vdash (AE\text{-}len\ v) \Rightarrow \{\!|\ z : B\text{-}int \mid [[z]^v]^{ce} == ((CE\text{-}len\ [v]^{ce}))|\!\}$
$\Theta\ ;\ \Phi\ ;\ \mathcal{B}\ ;\ \Gamma\ ;\ \Delta \vdash AE\text{-}concat\ v1\ v2 \Rightarrow \tau$
$\Theta\ ;\ \Phi\ ;\ \mathcal{B}\ ;\ \Gamma\ ;\ \Delta \vdash AE\text{-}concat\ v1\ v2 \Rightarrow (\{\!|\ z : b \mid\ c\ |\!\})$

Θ ; Φ ; $\mathcal{B}$; Γ ; $\Delta \vdash$ *AE-concat v1 v2* $\Rightarrow$ ($\{\!|$ z : *B-bitvec* | $[[z]^v]^{ce}$ == (*CE-concat* $[v1]^{ce}$ $[v1]^{ce}$) $|\!\}$)
Θ ; Φ ; $\mathcal{B}$; Γ ; $\Delta \vdash$ (*AE-appP f b v*) $\Rightarrow \tau$
Θ ; Φ ; $\mathcal{B}$; Γ ; $\Delta \vdash$ *AE-split v1 v2* $\Rightarrow \tau$
nominal-termination (*eqvt*) **by** *lexicographic-order*

9.6 Statements

inductive *check-s* :: $\Theta \Rightarrow \Phi \Rightarrow \mathcal{B} \Rightarrow \Gamma \Rightarrow \Delta \Rightarrow s \Rightarrow \tau \Rightarrow$ *bool* (- ; - ; - ; - ; - $\vdash$ - $\Leftarrow$ - [*50*, *50*, *50*,*50*,*50*] *50*) **and**
check-branch-s :: $\Theta \Rightarrow \Phi \Rightarrow \mathcal{B} \Rightarrow \Gamma \Rightarrow \Delta \Rightarrow$ *tyid* $\Rightarrow$ *string* $\Rightarrow \tau \Rightarrow v \Rightarrow$ *branch-s* $\Rightarrow \tau \Rightarrow$ *bool* (- ; - ; - ; - ; - ; - ; - ; - ; - ; - $\vdash$ - $\Leftarrow$ - [*50*, *50*, *50*,*50*,*50*] *50*) **and**
check-branch-list :: $\Theta \Rightarrow \Phi \Rightarrow \mathcal{B} \Rightarrow \Gamma \Rightarrow \Delta \Rightarrow$ *tyid* $\Rightarrow$ (*string* $*$ τ) *list* $\Rightarrow v \Rightarrow$ *branch-list* $\Rightarrow \tau \Rightarrow$ *bool* (- ; - ; - ; - ; - ; - ; - ; - $\vdash$ - $\Leftarrow$ - [*50*, *50*, *50*,*50*,*50*] *50*) **where**
check-valI: $[\![$
Θ ; $\mathcal{B}$;$\Gamma \vdash_{wf} \Delta$;
$\Theta \vdash_{wf} \Phi$;
Θ ; $\mathcal{B}$; $\Gamma \vdash v \Rightarrow \tau'$;
Θ ; $\mathcal{B}$; $\Gamma \vdash \tau' \lesssim \tau$ $]\!] \Longrightarrow$
Θ ; Φ ; $\mathcal{B}$; Γ ; $\Delta \vdash$ (*AS-val v*) $\Leftarrow \tau$

| *check-letI*: $[\![$
atom $x \sharp (\Theta, \Phi, \mathcal{B}, \Gamma, \Delta, e, \tau)$;
atom $z \sharp (x, \Theta, \Phi, \mathcal{B}, \Gamma, \Delta, e, \tau, s)$;
Θ ; Φ ; $\mathcal{B}$; Γ ; $\Delta \vdash e \Rightarrow \{\!|$ z : b | c $|\!\}$;
Θ ; Φ ; $\mathcal{B}$; $((x,b,c[z::=V\text{-}var\ x]_v)\#_\Gamma\Gamma)$; $\Delta \vdash s \Leftarrow \tau$
$]\!] \Longrightarrow$
Θ ; Φ ; $\mathcal{B}$; Γ ; $\Delta \vdash$ (*AS-let x e s*) $\Leftarrow \tau$

| *check-assertI*: $[\![$
atom $x \sharp (\Theta, \Phi, \mathcal{B}, \Gamma, \Delta, c, \tau, s)$;
Θ ; Φ ; $\mathcal{B}$; $((x,B\text{-}bool,c)\#_\Gamma\Gamma)$; $\Delta \vdash s \Leftarrow \tau$;
Θ ; $\mathcal{B}$; $\Gamma \models c$;
Θ ; $\mathcal{B}$;$\Gamma \vdash_{wf} \Delta$
$]\!] \Longrightarrow$
Θ ; Φ ; $\mathcal{B}$; Γ ; $\Delta \vdash$ (*AS-assert c s*) $\Leftarrow \tau$

|*check-branch-s-branchI*: $[\![$
Θ ; $\mathcal{B}$;$\Gamma \vdash_{wf} \Delta$;
$\vdash_{wf} \Theta$;
Θ ; $\mathcal{B}$;$\Gamma \vdash_{wf} \tau$;
Θ ; $\{||\}$; *GNil* $\vdash_{wf}$ *const*;
atom $x \sharp (\Theta, \Phi, \mathcal{B}, \Gamma, \Delta, tid, cons\ , const, v, \tau)$;
Θ ; Φ ; $\mathcal{B}$; $((x,b\text{-}of\ const,\ ([v]^{ce} == [\ V\text{-}cons\ tid\ cons\ [x]^v]^{ce}\ )\ AND\ (c\text{-}of\ const\ x))\#_\Gamma\Gamma)$; $\Delta \vdash s \Leftarrow \tau$
$]\!] \Longrightarrow$
Θ ; Φ ; $\mathcal{B}$; Γ ; Δ ; *tid* ; *cons* ; *const* ; $v \vdash$ (*AS-branch cons x s*) $\Leftarrow \tau$

|*check-branch-list-consI*: $[\![$
Θ ; Φ ; $\mathcal{B}$; Γ ; Δ; *tid* ; *cons* ; *const* ; $v \vdash cs \Leftarrow \tau$;
Θ ; Φ ; $\mathcal{B}$; Γ ; Δ; *tid* ; *dclist* ; $v \vdash css \Leftarrow \tau$
$]\!] \Longrightarrow$

Θ ; Φ ; $\mathcal{B}$; Γ ; Δ ; *tid* ; (*cons*,*const*)#*dclist* ; $v \vdash$ *AS-cons cs css* $\Leftarrow \tau$

|*check-branch-list-finalI*: $[\![$
Θ ; Φ ; $\mathcal{B}$; Γ ; Δ; *tid* ; *cons* ; *const* ; $v \vdash$ *cs* $\Leftarrow \tau$
$]\!] \Longrightarrow$
Θ ; Φ ; $\mathcal{B}$; Γ ; Δ ; *tid* ; [(*cons*,*const*)] ; $v \vdash$ *AS-final cs* $\Leftarrow \tau$

| *check-ifI*: $[\![$
atom $z \sharp (\Theta, \Phi, \mathcal{B}, \Gamma, \Delta, v$, *s1* , *s2* , τ);
$(\Theta$; $\mathcal{B}$; $\Gamma \vdash v \Leftarrow (\{\!|\ z$: *B-bool* | *TRUE* $|\!\}))$;
Θ ; Φ ; $\mathcal{B}$; Γ ; $\Delta \vdash$ *s1* $\Leftarrow (\{\!|\ z$: *b-of* τ | $([v]^{ce} == [[$*L-true*$]^v]^{ce})$ *IMP* (*c-of* τ *z*) $|\!\})$;
Θ ; Φ ; $\mathcal{B}$; Γ ; $\Delta \vdash$ *s2* $\Leftarrow (\{\!|\ z$: *b-of* τ | $([v]^{ce} == [[$*L-false*$]^v]^{ce})$ *IMP* (*c-of* τ *z*) $|\!\})$
$]\!] \Longrightarrow$
Θ ; Φ ; $\mathcal{B}$; Γ ; $\Delta \vdash$ *IF v THEN s1 ELSE s2* $\Leftarrow \tau$

| *check-let2I*: $[\![$
atom $x \sharp (\Theta, \Phi, \mathcal{B}, G, \Delta, t,$ *s1*, $\tau)$;
Θ ; Φ ; $\mathcal{B}$; G; $\Delta \vdash$ *s1* $\Leftarrow t$;
Θ ; Φ ; $\mathcal{B}$; $((x$,*b-of* t,*c-of* $t\ x)\#_\Gamma G)$; $\Delta \vdash$ *s2* $\Leftarrow \tau$
$]\!] \Longrightarrow$
Θ ; Φ ; $\mathcal{B}$; G ; $\Delta \vdash$ (*LET* $x : t$ = *s1 IN s2*) $\Leftarrow \tau$

| *check-varI*: $[\![$
atom $u \sharp (\Theta, \Phi, \mathcal{B}, \Gamma, \Delta, \tau', v, \tau)$;
Θ ; $\mathcal{B}$; $\Gamma \vdash\ v \Leftarrow \tau'$;
Θ ; Φ ; $\mathcal{B}$; Γ ; $((u,\tau')\ \#_\Delta\ \Delta)\ \vdash s \Leftarrow \tau$
$]\!] \Longrightarrow$
Θ ; Φ ; $\mathcal{B}$; Γ ; $\Delta \vdash$ (*VAR* $u : \tau' = v$ *IN* s) $\Leftarrow \tau$

| *check-assignI*: $[\![$
$\Theta \vdash_{wf} \Phi$;
Θ ; $\mathcal{B}$;$\Gamma \vdash_{wf} \Delta$;
$(u,\tau) \in$ *setD* Δ ;
Θ ; $\mathcal{B}$; $\Gamma \vdash\ v \Leftarrow \tau$;
Θ ; $\mathcal{B}$; $\Gamma \vdash (\{\!|\ z$: *B-unit* | *TRUE* $|\!\}) \lesssim \tau'$
$]\!] \Longrightarrow$
Θ ; Φ ; $\mathcal{B}$; Γ ; $\Delta \vdash\ (u ::= v) \Leftarrow \tau'$

| *check-whileI*: $[\![$
Θ ; Φ ; $\mathcal{B}$; Γ ; $\Delta \vdash$ *s1* $\Leftarrow \{\!|\ z$: *B-bool* | *TRUE* $|\!\}$;
Θ ; Φ ; $\mathcal{B}$; Γ ; $\Delta \vdash$ *s2* $\Leftarrow \{\!|\ z$: *B-unit* | *TRUE* $|\!\}$;
Θ ; $\mathcal{B}$; $\Gamma \vdash (\{\!|\ z$: *B-unit* | *TRUE* $|\!\}) \lesssim \tau'$
$]\!] \Longrightarrow$
Θ ; Φ ; $\mathcal{B}$; Γ ; $\Delta \vdash$ *WHILE s1 DO* { *s2* } $\Leftarrow \tau'$

| *check-seqI*: $[\![$
Θ ; Φ ; $\mathcal{B}$; Γ ; $\Delta \vdash$ *s1* $\Leftarrow \{\!|\ z$: *B-unit* | *TRUE* $|\!\}$;
Θ ; Φ ; $\mathcal{B}$; Γ ; $\Delta \vdash$ *s2* $\Leftarrow \tau$
$]\!] \Longrightarrow$
Θ ; Φ ; $\mathcal{B}$; Γ ; $\Delta \vdash$ *s1* ;; *s2* $\Leftarrow \tau$

| *check-caseI*: $[\![$

Θ ; Φ ; $\mathcal{B}$; Γ ; Δ; *tid* ; *dclist* ; $v \vdash$ *cs* $\Leftarrow \tau$;
(*AF-typedef tid dclist*) $\in$ *set* Θ ;
Θ ; $\mathcal{B}$; $\Gamma \vdash v \Leftarrow \{\!|\ z : \textit{B-id tid} \mid \textit{TRUE}\ |\!\}$;
$\vdash_{wf} \Theta$
$]\!] \Longrightarrow$
Θ ; Φ ; $\mathcal{B}$; Γ ; $\Delta \vdash$ *AS-match v cs* $\Leftarrow \tau$

equivariance *check-s*

We only need avoidance for cases where a variable is added to a context

nominal-inductive *check-s*
avoids *check-letI*: *x* **and** *z* | *check-branch-s-branchI*: *x* | *check-let2I*: *x* | *check-varI*: *u* | *check-ifI*: *z* | *check-assertI*: *x*
proof(*goal-cases*)
case (*1 x* Θ Φ $\mathcal{B}$ Γ Δ *e* τ *z s b c*)
hence *atom x* $\sharp$ *AS-let x e s* **using** *s-branch-s-branch-list.fresh*(*2*) **by** *auto*
moreover have *atom z* $\sharp$ *AS-let x e s* **using** *s-branch-s-branch-list.fresh*(*2*) *1 fresh-prod8* **by** *auto*
then show *?case* **using** *fresh-star-def 1* **by** *force*
next
case (*3 x* Θ Φ $\mathcal{B}$ Γ Δ *c* τ *s*)
hence *atom x* $\sharp$ *AS-assert c s* **using** *fresh-prodN s-branch-s-branch-list.fresh pure-fresh* **by** *auto*
then show *?case* **using** *fresh-star-def 3* **by** *force*
next
case (*5* Θ $\mathcal{B}$ Γ Δ τ *const x* Φ *tid cons v s*)
hence *atom x* $\sharp$ *AS-branch cons x s* **using** *fresh-prodN s-branch-s-branch-list.fresh pure-fresh* **by** *auto*

then show *?case* **using** *fresh-star-def 5* **by** *force*
next
case (*7 z* Θ Φ $\mathcal{B}$ Γ Δ *v s1 s2* τ)
hence *atom z* $\sharp$ *AS-if v s1 s2* **using** *s-branch-s-branch-list.fresh* **by** *auto*
then show *?case* **using** *7 fresh-prodN fresh-star-def* **by** *fastforce*
next
case (*9 x* Θ Φ $\mathcal{B}$ *G* Δ *t s1* τ *s2*)
hence *atom x* $\sharp$ *AS-let2 x t s1 s2* **using** *s-branch-s-branch-list.fresh* **by** *auto*
thus *?case* **using** *fresh-star-def 9* **by** *force*
next
case (*11 u* Θ Φ $\mathcal{B}$ Γ Δ τ' *v* τ *s*)
hence *atom u* $\sharp$ *AS-var u* τ' *v s* **using** *s-branch-s-branch-list.fresh* **by** *auto*
then show *?case* **using** *fresh-star-def 11* **by** *force*

qed(*auto*+)

inductive-cases *check-s-elims*[*elim*!]:
Θ ; Φ ; $\mathcal{B}$; Γ ; $\Delta \vdash$ *AS-val v* $\Leftarrow \tau$
Θ ; Φ ; $\mathcal{B}$; Γ ; $\Delta \vdash$ *AS-let x e s* $\Leftarrow \tau$
Θ ; Φ ; $\mathcal{B}$; Γ ; $\Delta \vdash$ *AS-if v s1 s2* $\Leftarrow \tau$
Θ ; Φ ; $\mathcal{B}$; Γ ; $\Delta \vdash$ *AS-let2 x t s1 s2* $\Leftarrow \tau$
Θ ; Φ ; $\mathcal{B}$; Γ ; $\Delta \vdash$ *AS-while s1 s2* $\Leftarrow \tau$
Θ ; Φ ; $\mathcal{B}$; Γ ; $\Delta \vdash$ *AS-var u t v s* $\Leftarrow \tau$
Θ ; Φ ; $\mathcal{B}$; Γ ; $\Delta \vdash$ *AS-seq s1 s2* $\Leftarrow \tau$
Θ ; Φ ; $\mathcal{B}$; Γ ; $\Delta \vdash$ *AS-assign u v* $\Leftarrow \tau$

Θ ; Φ ; $\mathcal{B}$; Γ ; $\Delta \vdash$ *AS-match v cs* $\Leftarrow \tau$
Θ ; Φ ; $\mathcal{B}$; Γ ; $\Delta \vdash$ *AS-assert c s* $\Leftarrow \tau$

inductive-cases *check-branch-s-elims*[*elim*!]:
Θ ; Φ ; $\mathcal{B}$; Γ ; Δ; *tid* ; *dclist* ; $v \vdash$ (*AS-final cs*) $\Leftarrow \tau$
Θ ; Φ ; $\mathcal{B}$; Γ ; Δ; *tid* ; *dclist* ; $v \vdash$ (*AS-cons cs css*) $\Leftarrow \tau$
Θ ; Φ ; $\mathcal{B}$; Γ ; Δ; *tid* ; *cons* ; *const* ; $v \vdash$ (*AS-branch dc x s*) $\Leftarrow \tau$

9.7 Programs

inductive *check-funtyp* :: $\Theta \Rightarrow \Phi \Rightarrow \mathcal{B} \Rightarrow$ *fun-typ* $\Rightarrow$ *bool* **where**
check-funtypI: ⟦
atom x ♯ (Θ, Φ, *B* , *b*);
Θ ; Φ ; *B* ; ((*x*,*b*,*c*) $\#_\Gamma$ *GNil*) ; $[]_\Delta \vdash s \Leftarrow \tau$
⟧ $\Longrightarrow$
check-funtyp Θ Φ *B* (*AF-fun-typ x b c* τ *s*)

equivariance *check-funtyp*
nominal-inductive *check-funtyp*
avoids *check-funtypI*: *x*
proof(*goal-cases*)
case (*1 x* Θ Φ *B b c s* τ)
hence *atom x* ♯ (*AF-fun-typ x b c* τ *s*) **using** *fun-def.fresh fun-typ-q.fresh fun-typ.fresh* **by** *simp*
then show *?case* **using** *fresh-star-def 1 fresh-prodN* **by** *fastforce*
next
case (*2* Θ Φ *x b c s* τ *f*)
then show *?case* **by** *auto*
qed

inductive *check-funtypq* :: $\Theta \Rightarrow \Phi \Rightarrow$ *fun-typ-q* $\Rightarrow$ *bool* **where**
check-fundefq-simpleI: ⟦
check-funtyp Θ Φ {||} (*AF-fun-typ x b c t s*)
⟧ $\Longrightarrow$
check-funtypq Θ Φ ((*AF-fun-typ-none* (*AF-fun-typ x b c t s*)))

|*check-funtypq-polyI*: ⟦
atom bv ♯ (Θ, Φ, (*AF-fun-typ x b c t s*));
check-funtyp Θ Φ {|*bv*|} (*AF-fun-typ x b c t s*)
⟧ $\Longrightarrow$
check-funtypq Θ Φ (*AF-fun-typ-some bv* (*AF-fun-typ x b c t s*))

equivariance *check-funtypq*
nominal-inductive *check-funtypq*
avoids *check-funtypq-polyI*: *bv*
proof(*goal-cases*)
case (*1 bv* Θ Φ *x b c t s*)
hence *atom bv* ♯ (*AF-fun-typ-some bv* (*AF-fun-typ x b c t s*)) **using** *fun-def.fresh fun-typ-q.fresh fun-typ.fresh* **by** *simp*
thus *?case* **using** *fresh-star-def 1 fresh-prodN* **by** *fastforce*
next
case (*2 bv* Θ Φ *ft*)
then show *?case* **by** *auto*

qed

inductive *check-fundef* :: $\Theta \Rightarrow \Phi \Rightarrow$ *fun-def* $\Rightarrow$ *bool* **where**
check-fundefI: $[\![$
  check-funtypq Θ Φ *ft*
$]\!] \Longrightarrow$
  check-fundef Θ Φ ((*AF-fundef f ft*))

equivariance *check-fundef*
nominal-inductive *check-fundef* **.**

Temporarily remove this simproc as it produces untidy eliminations

declare[[*simproc del*: *alpha-lst*]]

inductive-cases *check-funtyp-elims*[*elim*!]:
  check-funtyp Θ Φ *B ft*

inductive-cases *check-funtypq-elims*[*elim*!]:
  check-funtypq Θ Φ (*AF-fun-typ-none* (*AF-fun-typ x b c* τ *s*))
  check-funtypq Θ Φ (*AF-fun-typ-some bv* (*AF-fun-typ x b c* τ *s*))

inductive-cases *check-fundef-elims*[*elim*!]:
  check-fundef Θ Φ (*AF-fundef f ftq*)

declare[[*simproc add*: *alpha-lst*]]

end

Chapter 10

Operational Semantics

Here we define the operational semantics in terms of a small-step reduction relation.

10.1 Reduction Rules

The store for mutable variables

type-synonym $\delta = (u*v)\ list$

nominal-function *update-d* :: $\delta \Rightarrow u \Rightarrow v \Rightarrow \delta$ **where**
update-d [] - - = []
| *update-d* ((*u'*,*v'*)#δ) *u v* = (*if u* = *u' then* ((*u*,*v*)#δ) *else* ((*u'*,*v'*)# (*update-d* δ *u v*)))
by(*auto*,*simp add*: *eqvt-def update-d-graph-aux-def* ,*metis neq-Nil-conv old.prod.exhaust*)
nominal-termination (*eqvt*) **by** *lexicographic-order*

Relates constructor to the branch in the case and binding variable and statement

inductive *find-branch* :: *dc* $\Rightarrow$ *branch-list* $\Rightarrow$ *branch-s* $\Rightarrow$ *bool* **where**
find-branch-finalI: *dc'* = *dc* $\Longrightarrow$ *find-branch dc'* (*AS-final* (*AS-branch dc x s*)) (*AS-branch dc x s*)
| *find-branch-branch-eqI*: *dc'* = *dc* $\Longrightarrow$ *find-branch dc'* (*AS-cons* (*AS-branch dc x s*) *css*) (*AS-branch dc x s*)
| *find-branch-branch-neqI*: $[\![$ *dc* $\neq$ *dc'*; *find-branch dc' css cs* $]\!] \Longrightarrow$ *find-branch dc'* (*AS-cons* (*AS-branch dc x s*) *css*) *cs*
equivariance *find-branch*
nominal-inductive *find-branch* .

inductive-cases *find-branch-elims*[*elim*!]:
find-branch dc (*AS-final cs'*) *cs*
find-branch dc (*AS-cons cs' css*) *cs*

nominal-function *lookup-branch* :: *dc* $\Rightarrow$ *branch-list* $\Rightarrow$ *branch-s option* **where**
lookup-branch dc (*AS-final* (*AS-branch dc' x s*)) = (*if dc* = *dc' then* (*Some* (*AS-branch dc' x s*)) *else None*)
| *lookup-branch dc* (*AS-cons* (*AS-branch dc' x s*) *css*) = (*if dc* = *dc' then* (*Some* (*AS-branch dc' x s*)) *else lookup-branch dc css*)
apply(*auto*,*simp add*: *eqvt-def lookup-branch-graph-aux-def*)

by(*metis neq-Nil-conv old.prod.exhaust s-branch-s-branch-list.strong-exhaust*)
nominal-termination (*eqvt*) **by** *lexicographic-order*

value *take 1* [*1*::*nat*,*2*]

Reduction rules

inductive *reduce-stmt* :: $\Phi \Rightarrow \delta \Rightarrow s \Rightarrow \delta \Rightarrow s \Rightarrow$ *bool* (- $\vdash$ $\langle$ - , - $\rangle \longrightarrow \langle$ - , - $\rangle$ [*50*, *50*, *50*] *50*)
where
reduce-if-trueI: $\Phi \vdash \langle\ \delta$, *AS-if* $[$*L-true*$]^v$ *s1 s2* $\rangle \longrightarrow \langle\ \delta$, *s1* $\rangle$
| *reduce-if-falseI*: $\Phi \vdash \langle\ \delta$, *AS-if* $[$*L-false*$]^v$ *s1 s2* $\rangle \longrightarrow \langle\ \delta$, *s2* $\rangle$
| *reduce-let-valI*: $\Phi \vdash \langle\ \delta$, *AS-let x* (*AE-val v*) *s* $\rangle \longrightarrow \langle\ \delta$, $s[x::=v]_{sv}$ $\rangle$
| *reduce-let-plusI*: $\Phi \vdash$ $\langle\ \delta$, *AS-let x* (*AE-op Plus* ((*V-lit* (*L-num n1*))) ((*V-lit* (*L-num n2*)))) *s* $\rangle \longrightarrow$

$\langle\ \delta$, *AS-let x* (*AE-val* (*V-lit* (*L-num* (((*n1*)+(*n2*)))))) *s* $\rangle$
| *reduce-let-leqI*: *b* = (*if* (*n1* $\leq$ *n2*) *then L-true else L-false*) $\Longrightarrow$
$\Phi \vdash \langle\ \delta$, *AS-let x* ((*AE-op LEq* (*V-lit* (*L-num n1*)) (*V-lit* (*L-num n2*)))) *s* $\rangle \longrightarrow$
$\langle\ \delta$, *AS-let x* (*AE-val* (*V-lit b*)) *s* $\rangle$
| *reduce-let-appI*: *Some* (*AF-fundef f* (*AF-fun-typ-none* (*AF-fun-typ z b c* τ *s'*))) = *lookup-fun* Φ *f* $\Longrightarrow$

$\Phi \vdash \langle\ \delta$, *AS-let x* ((*AE-app f v*)) *s* $\rangle \longrightarrow \langle\ \delta$, *AS-let2 x* $\tau[z::=v]_{\tau v}$ $s'[z::=v]_{sv}$ *s* $\rangle$
| *reduce-let-appPI*: *Some* (*AF-fundef f* (*AF-fun-typ-some bv* (*AF-fun-typ z b c* τ *s'*))) = *lookup-fun* Φ *f* $\Longrightarrow$
$\Phi \vdash \langle\ \delta$, *AS-let x* ((*AE-appP f b' v*)) *s* $\rangle \longrightarrow \langle\ \delta$, *AS-let2 x* $\tau[bv::=b']_{\tau b}[z::=v]_{\tau v}$ $s'[bv::=b']_{sb}[z::=v]_{sv}$ *s* $\rangle$
| *reduce-let-fstI*: $\Phi \vdash \langle\ \delta$, *AS-let x* (*AE-fst* (*V-pair v1 v2*)) *s* $\rangle \longrightarrow \langle\ \delta$, *AS-let x* (*AE-val v1*) *s* $\rangle$
| *reduce-let-sndI*: $\Phi \vdash \langle\ \delta$, *AS-let x* (*AE-snd* (*V-pair v1 v2*)) *s* $\rangle \longrightarrow \langle\ \delta$, *AS-let x* (*AE-val v2*) *s* $\rangle$
| *reduce-let-concatI*: $\Phi \vdash \langle\ \delta$, *AS-let x* (*AE-concat* (*V-lit* (*L-bitvec v1*)) (*V-lit* (*L-bitvec v2*))) *s* $\rangle$
$\longrightarrow$
$\langle\ \delta$, *AS-let x* (*AE-val* (*V-lit* (*L-bitvec* (*v1*@*v2*)))) *s* $\rangle$
| *reduce-let-splitI*: *split n v* (*v1* , *v2*) $\Longrightarrow \Phi \vdash \langle\ \delta$, *AS-let x* (*AE-split* (*V-lit* (*L-bitvec v*)) (*V-lit* (*L-num n*))) *s* $\rangle \longrightarrow$
$\langle\ \delta$, *AS-let x* (*AE-val* (*V-pair* (*V-lit* (*L-bitvec v1*)) (*V-lit* (*L-bitvec v2*)))) *s* $\rangle$
| *reduce-let-lenI*: $\Phi \vdash \langle\ \delta$, *AS-let x* (*AE-len* (*V-lit* (*L-bitvec v*))) *s* $\rangle \longrightarrow$
$\langle\ \delta$, *AS-let x* (*AE-val* (*V-lit* (*L-num* (*int* (*List.length v*))))) *s* $\rangle$
| *reduce-let-mvar*: (*u*,*v*) $\in$ *set* $\delta \Longrightarrow \Phi \vdash \langle\ \delta$, *AS-let x* (*AE-mvar u*) *s* $\rangle \longrightarrow \langle\ \delta$, *AS-let x* (*AE-val v*) *s* $\rangle$
| *reduce-assert1I*: $\Phi \vdash \langle\ \delta$, *AS-assert c* (*AS-val v*) $\rangle \longrightarrow \langle\ \delta$, *AS-val v* $\rangle$
| *reduce-assert2I*: $\Phi \vdash \langle\ \delta$, *s* $\rangle \longrightarrow \langle\ \delta'$, *s'* $\rangle \Longrightarrow \Phi \vdash \langle\ \delta$, *AS-assert c s* $\rangle \longrightarrow \langle\ \delta'$, *AS-assert c s'* $\rangle$
| *reduce-varI*: *atom u* $\sharp$ $\delta \Longrightarrow \Phi \vdash \langle\ \delta$, *AS-var u* τ *v s* $\rangle \longrightarrow \langle$ $((u,v)\#\delta)$, *s* $\rangle$
| *reduce-assignI*: $\Phi \vdash \langle\ \delta$, *AS-assign u v* $\rangle \longrightarrow \langle$ *update-d* δ *u v* , *AS-val* (*V-lit L-unit*) $\rangle$
| *reduce-seq1I*: $\Phi \vdash \langle\ \delta$, *AS-seq* (*AS-val* (*V-lit L-unit*)) *s* $\rangle \longrightarrow \langle\ \delta$, *s* $\rangle$
| *reduce-seq2I*: $[\![$*s1* $\neq$ *AS-val v* ; $\Phi \vdash \langle\ \delta$, *s1* $\rangle \longrightarrow \langle\ \delta'$, *s1'* $\rangle\ ]\!] \Longrightarrow$
$\Phi \vdash \langle\ \delta$, *AS-seq s1 s2* $\rangle \longrightarrow \langle\ \delta'$, *AS-seq s1' s2* $\rangle$
| *reduce-let2-valI*: $\Phi \vdash \langle\ \delta$, *AS-let2 x t* (*AS-val v*) *s* $\rangle \longrightarrow \langle\ \delta$, $s[x::=v]_{sv}$ $\rangle$
| *reduce-let2I*: $\Phi \vdash \langle\ \delta$, *s1* $\rangle \longrightarrow \langle\ \delta'$, *s1'* $\rangle \Longrightarrow \Phi \vdash \langle\ \delta$, *AS-let2 x t s1 s2* $\rangle \longrightarrow \langle\ \delta'$, *AS-let2 x t s1' s2* $\rangle$

| *reduce-caseI*: $[\![$ *Some* (*AS-branch dc x' s'*) = *lookup-branch dc cs* $]\!] \Longrightarrow \Phi \vdash \langle\ \delta$, *AS-match* (*V-cons tyid dc v*) *cs* $\rangle \longrightarrow \langle\ \delta$, $s'[x'::=v]_{sv}$ $\rangle$
| *reduce-whileI*: $[\![$ *atom x* $\sharp$ (*s1*,*s2*); *atom z* $\sharp$ *x* $]\!] \Longrightarrow$
$\Phi \vdash \langle\ \delta$, *AS-while s1 s2* $\rangle \longrightarrow$

$\langle\ \delta\ ,\ AS\text{-}let2\ x\ (\{\!|\ z : B\text{-}bool\ |\ TRUE\ |\!\})\ s1\ (AS\text{-}if\ (V\text{-}var\ x)\ (AS\text{-}seq\ s2\ (AS\text{-}while\ s1\ s2))\ (AS\text{-}val\ (V\text{-}lit\ L\text{-}unit)))\ \rangle$

equivariance *reduce-stmt*
nominal-inductive *reduce-stmt* **.**

inductive-cases *reduce-stmt-elims*[*elim*!]:
$\Phi \vdash \langle\ \delta\ ,\ AS\text{-}if\ (V\text{-}lit\ L\text{-}true)\ s1\ s2\ \rangle \longrightarrow \langle \delta\ ,\ s1\ \rangle$
$\Phi \vdash \langle\ \delta\ ,\ AS\text{-}if\ (V\text{-}lit\ L\text{-}false)\ s1\ s2\ \rangle \longrightarrow \langle\ \delta\ ,s2\ \rangle$
$\Phi \vdash \langle\ \delta\ ,\ AS\text{-}let\ x\ (AE\text{-}val\ v)\ \ s\ \rangle \longrightarrow \langle\ \delta\ ,s'\ \rangle$
$\Phi \vdash \langle\ \delta\ ,\ AS\text{-}let\ x\ \ (AE\text{-}op\ Plus\ ((V\text{-}lit\ (L\text{-}num\ n1)))\ ((V\text{-}lit\ (L\text{-}num\ n2))))\ \ s\ \rangle \longrightarrow$
$\langle\ \delta\ ,AS\text{-}let\ x\ \ (AE\text{-}val\ (V\text{-}lit\ (L\text{-}num\ (\ ((\ n1)+(n2))))))\ s\ \ \rangle$
$\Phi \vdash \langle\ \delta\ ,\ AS\text{-}let\ x\ \ ((AE\text{-}op\ LEq\ (V\text{-}lit\ (L\text{-}num\ n1))\ (V\text{-}lit\ (L\text{-}num\ n2))))\ s\ \rangle \longrightarrow \langle\ \delta\ ,\ AS\text{-}let\ x\ (AE\text{-}val\ (V\text{-}lit\ b))\ s\ \rangle$
$\Phi \vdash \langle\ \delta\ ,\ AS\text{-}let\ x\ \ ((AE\text{-}app\ f\ v))\ s\ \rangle \longrightarrow \langle\ \delta\ ,AS\text{-}let2\ x\ \tau\ (subst\text{-}sv\ s'\ x\ v\ )\ s\ \rangle$
$\Phi \vdash \langle\ \delta\ ,\ AS\text{-}let\ x\ \ ((AE\text{-}len\ v))\ s\ \rangle \longrightarrow \langle\ \delta\ ,AS\text{-}let\ x\ \ v'\ s\ \rangle$
$\Phi \vdash \langle\ \delta\ ,\ AS\text{-}let\ x\ \ ((AE\text{-}concat\ v1\ v2))\ s\ \rangle \longrightarrow \langle\ \delta\ ,AS\text{-}let\ x\ \ v'\ s\ \rangle$
$\Phi \vdash \langle\ \delta\ ,\ AS\text{-}seq\ s1\ s2\ \rangle \longrightarrow \langle\ \delta'\ ,s'\ \rangle$
$\Phi \vdash \langle\ \delta\ ,\ AS\text{-}let\ x\ \ ((AE\text{-}appP\ \ f\ b\ v))\ s\ \rangle \longrightarrow \langle\ \delta\ ,AS\text{-}let2\ x\ \tau\ (subst\text{-}sv\ s'\ z\ v)\ s\ \rangle$
$\Phi \vdash \langle\ \delta\ ,\ AS\text{-}let\ x\ \ ((AE\text{-}split\ v1\ v2))\ s\ \rangle \longrightarrow \langle\ \delta\ ,AS\text{-}let\ x\ \ v'\ s\ \rangle$
$\Phi \vdash \langle\ \delta\ ,\ AS\text{-}assert\ c\ s\ \ \rangle \longrightarrow \langle\ \delta\ ,\ s'\ \rangle$

inductive *reduce-stmt-many* :: $\Phi \Rightarrow \delta \Rightarrow s \Rightarrow \delta \Rightarrow s \Rightarrow bool$ $(\text{-} \vdash \langle\ \text{-}\ ,\ \text{-}\ \rangle \longrightarrow^* \langle\ \ \text{-}\ ,\ \text{-}\ \rangle\ [50,\ 50,\ 50]\ 50)$ **where**
reduce-stmt-many-oneI: $\Phi \vdash \langle\ \delta\ \ ,\ s\ \rangle \longrightarrow \langle\ \ \delta'\ ,\ s'\ \rangle \Longrightarrow \Phi \vdash \langle\ \delta\ \ ,\ s\ \rangle \longrightarrow^* \langle\ \ \delta'\ ,\ s'\ \rangle$
| *reduce-stmt-many-manyI*: $[\![\ \Phi \vdash \langle\ \delta\ \ ,\ s\ \rangle \longrightarrow\ \ \langle\ \ \delta'\ ,\ s'\ \rangle\ ;\ \Phi \vdash\ \langle\ \delta'\ \ ,\ s'\ \rangle \longrightarrow^* \langle\ \ \delta''\ ,\ s''\ \rangle\ ]\!] \Longrightarrow \Phi \vdash\ \langle\ \delta\ \ ,\ s\ \rangle \longrightarrow^* \langle\ \ \delta''\ ,\ s''\ \rangle$

nominal-function *convert-fds* :: *fun-def list* $\Rightarrow$ $(f{*}fun\text{-}def)$ *list* **where**
convert-fds $[] = []$
| *convert-fds* $((AF\text{-}fundef\ f\ (AF\text{-}fun\text{-}typ\text{-}none\ (AF\text{-}fun\text{-}typ\ x\ b\ c\ \tau\ s)))\#fs) = ((f,AF\text{-}fundef\ f\ (AF\text{-}fun\text{-}typ\text{-}none\ (AF\text{-}fun\text{-}typ\ x\ b\ c\ \tau\ s)))\#convert\text{-}fds\ fs)$
| *convert-fds* $((AF\text{-}fundef\ f\ \ (AF\text{-}fun\text{-}typ\text{-}some\ bv\ \ (AF\text{-}fun\text{-}typ\ \ x\ \ b\ \ c\ \ \tau\ \ s)))\#fs)\ =\ ((f,AF\text{-}fundef\ f\ (AF\text{-}fun\text{-}typ\text{-}some\ bv\ (AF\text{-}fun\text{-}typ\ x\ b\ c\ \tau\ s)))\#convert\text{-}fds\ fs)$
apply(*auto*)
apply (*simp add*: *eqvt-def convert-fds-graph-aux-def*)
using *fun-def.exhaust fun-typ.exhaust fun-typ-q.exhaust neq-Nil-conv*
by *metis*
nominal-termination (*eqvt*) **by** *lexicographic-order*

nominal-function *convert-tds* :: *type-def list* $\Rightarrow$ $(f{*}type\text{-}def)$ *list* **where**
convert-tds $[] = []$
| *convert-tds* $((AF\text{-}typedef\ s\ dclist)\#fs) = ((s,AF\text{-}typedef\ s\ dclist)\#convert\text{-}tds\ fs)$
| *convert-tds* $((AF\text{-}typedef\text{-}poly\ s\ bv\ dclist)\#fs) = ((s,AF\text{-}typedef\text{-}poly\ s\ bv\ dclist)\#convert\text{-}tds\ fs)$
apply(*auto*)
apply (*simp add*: *eqvt-def convert-tds-graph-aux-def*)
by (*metis type-def.exhaust neq-Nil-conv*)
nominal-termination (*eqvt*) **by** *lexicographic-order*

inductive *reduce-prog* :: $p \Rightarrow v \Rightarrow bool$ **where**

$[\![$ *reduce-stmt-many* Φ [] s δ (*AS-val* v) $]\!] \Longrightarrow$ *reduce-prog* (*AP-prog* Θ Φ [] s) v

10.2 Reduction Typing

Checks that the store is consistent with Δ

inductive *delta-sim* :: $\Theta \Rightarrow \delta \Rightarrow \Delta \Rightarrow$ *bool* (- $\vdash$ - $\sim$ - [50,50] 50) **where**
delta-sim-nilI: $\Theta \vdash [] \sim []_\Delta$
| *delta-sim-consI*: $[\![\Theta \vdash \delta \sim \Delta$; Θ ; {||} ; *GNil* $\vdash v \Leftarrow \tau$; $u \notin$ *fst* ' *set* δ $]\!] \Longrightarrow \Theta \vdash ((u,v)\#\delta) \sim ((u,\tau)\#_\Delta\Delta)$

equivariance *delta-sim*
nominal-inductive *delta-sim* .

inductive-cases *delta-sim-elims*[*elim*!]:
$\Theta \vdash [] \sim []_\Delta$
$\Theta \vdash ((u,v)\#ds) \sim (u,\tau) \#_\Delta D$
$\Theta \vdash ((u,v)\#ds) \sim D$

A typing judgement that combines typing of the statement, the store and the condition that functions are well-formed

inductive *config-type* :: $\Theta \Rightarrow \Phi \Rightarrow \Delta \Rightarrow \delta \Rightarrow s \Rightarrow \tau \Rightarrow$ *bool* (- ; - ; - $\vdash \langle$ - , - $\rangle \Leftarrow$ - [50, 50, 50] 50) **where**
config-typeI: $[\![\Theta$; Φ ; {||} ; *GNil* ; $\Delta \vdash s \Leftarrow \tau$;
($\forall$ *fd* $\in$ *set* Φ. *check-fundef* Θ Φ *fd*) ;
$\Theta \vdash \delta \sim \Delta]\!]$
$\Longrightarrow \Theta$; Φ ; $\Delta \vdash \langle \delta$, $s \rangle \Leftarrow \tau$
equivariance *config-type*
nominal-inductive *config-type* .

inductive-cases *config-type-elims* [*elim*!]:
Θ ; Φ ; $\Delta \vdash \langle \delta$, $s \rangle \Leftarrow \tau$

end

hide-const *Syntax.dom*

Chapter 11

Refinement Constraint Logic Lemmas

11.1 Lemmas

lemma *wfI-domi*:
assumes Θ ; $\Gamma \vdash i$
shows *fst* ‘ *setG* $\Gamma \subseteq$ *dom* i
using *wfI-def setG.simps assms* **by** *fastforce*

lemma *wfI-lookup*:
fixes G::Γ **and** b::b
assumes *Some* (b,c) = *lookup* G x **and** P ; $G \vdash i$ **and** *Some* $s = i\ x$ **and** P ; $B \vdash_{wf} G$
shows $P \vdash s : b$
proof −
have $(x,b,c) \in$ *setG* G **using** *lookup.simps assms*
using *lookup-in-g* **by** *blast*
then obtain s' **where** $*$:*Some* $s' = i\ x \wedge$ *wfRCV* $P\ s'\ b$ **using** *wfI-def assms* **by** *auto*
hence $s' = s$ **using** *assms* **by** (*metis option.inject*)
thus *?thesis* **using** $*$ **by** *auto*
qed

lemma *wfI-restrict-weakening*:
assumes *wfI* $\Theta\ \Gamma'\ i'$ **and** $i =$ *restrict-map* i' (*fst* ‘*setG* Γ) **and** *setG* $\Gamma \subseteq$ *setG* Γ'
shows Θ ; $\Gamma \vdash i$
proof −
{ fix x
assume $x \in$ *setG* Γ
have *case x of* $(x,\ b,\ c) \Rightarrow \exists s.$ *Some* $s = i\ x \wedge \Theta \vdash s : b$ **using** *assms wfI-def*
proof −
have *case x of* $(x,\ b,\ c) \Rightarrow \exists s.$ *Some* $s = i'\ x \wedge \Theta \vdash s : b$
using ‹$x \in$ *setG* Γ› *assms wfI-def* **by** *auto*
then have $\exists s.$ *Some* $s = i$ (*fst* x) $\wedge$ *wfRCV* $\Theta\ s$ (*fst* (*snd* x))
by (*simp add*: ‹$x \in$ *setG* Γ› *assms*(*2*) *case-prod-unfold*)
then show *?thesis*
by (*simp add*: *case-prod-unfold*)
qed
}

thus *?thesis* **using** *wfI-def assms* **by** *auto*
qed

lemma *wfI-suffix*:
fixes $G::\Gamma$
assumes *wfI P* $(G'@G)$ *i* **and** P ; $B \vdash_{wf} G$
shows P ; $G \vdash i$
using *wfI-def append-g.simps assms setG.simps* **by** *simp*

lemma *wfI-replace-inside*:
assumes *wfI* Θ $(\Gamma' @ (x, b, c) \#_\Gamma \Gamma)$ *i*
shows *wfI* Θ $(\Gamma' @ (x, b, c') \#_\Gamma \Gamma)$ *i*
using *wfI-def setG-splitP assms* **by** *simp*

11.2 Existence of evaluation

lemma *eval-l-base*:
$\Theta \vdash [\![l]\!]$: (*base-for-lit l*)
apply(*nominal-induct l rule*:*l.strong-induct*)
using *wfRCV.intros eval-l.simps base-for-lit.simps* **by** *auto+*

lemma *obtain-fresh-bv-dclist*:
fixes *tm*::*'a*::*fs*
assumes (*dc*, $\{\!|$ *x* : *b* | *c* $|\!\}$) ∈ *set dclist*
obtains *bv1*::*bv* **and** *dclist1 x1 b1 c1* **where** *AF-typedef-poly tyid bv dclist* = *AF-typedef-poly tyid bv1 dclist1*
∧ (*dc*, $\{\!|$ *x1* : *b1* | *c1* $|\!\}$) ∈ *set dclist1* ∧ *atom bv1* ♯ *tm*
proof −
obtain *bv1 dclist1* **where** *AF-typedef-poly tyid bv dclist* = *AF-typedef-poly tyid bv1 dclist1* ∧ *atom bv1* ♯ *tm*
using *obtain-fresh-bv* **by** *metis*
moreover hence [[*atom bv*]]*lst*. *dclist* = [[*atom bv1*]]*lst*. *dclist1* **using** *type-def.eq-iff* **by** *metis*
moreover then obtain *x1 b1 c1* **where** (*dc*, $\{\!|$ *x1* : *b1* | *c1* $|\!\}$) ∈ *set dclist1* **using** *td-lookup-eq-iff assms* **by** *metis*
ultimately show *?thesis* **using** *that* **by** *blast*
qed

lemma *obtain-fresh-bv-dclist-b-iff*:
fixes *tm*::*'a*::*fs*
assumes (*dc*, $\{\!|$ *x* : *b* | *c* $|\!\}$) ∈ *set dclist* **and** *AF-typedef-poly tyid bv dclist* ∈ *set P* **and** $\vdash_{wf} P$
obtains *bv1*::*bv* **and** *dclist1 x1 b1 c1* **where** *AF-typedef-poly tyid bv dclist* = *AF-typedef-poly tyid bv1 dclist1*
∧ (*dc*, $\{\!|$ *x1* : *b1* | *c1* $|\!\}$) ∈ *set dclist1* ∧ *atom bv1* ♯ *tm* ∧ $b[bv::=b']_{bb}$=$b1[bv1::=b']_{bb}$
proof −
obtain *bv1 dclist1 x1 b1 c1* **where** ∗:*AF-typedef-poly tyid bv dclist* = *AF-typedef-poly tyid bv1 dclist1* ∧ *atom bv1* ♯ *tm*
∧ (*dc*, $\{\!|$ *x1* : *b1* | *c1* $|\!\}$) ∈ *set dclist1* **using** *obtain-fresh-bv-dclist assms* **by** *metis*

hence *AF-typedef-poly tyid bv1 dclist1* ∈ *set P* **using** *assms* **by** *metis*

hence $b[bv::=b']_{bb}$ = $b1[bv1::=b']_{bb}$
using *wfTh-typedef-poly-b-eq-iff*[*OF assms*(*2*) *assms*(*1*) - - *assms*(*3*),*of bv1 dclist1 x1 b1 c1 b'*] ∗

by *metis*

from *this that* **show** *?thesis* **using** $*$ **by** *metis*
qed

lemma *eval-v-exist*:
fixes Γ::Γ **and** v::v **and** b::b
assumes P ; Γ $\vdash$ i **and** P ; B ; $\Gamma \vdash_{wf} v : b$
shows $\exists s.\ i \llbracket v \rrbracket^{\sim}\ s \wedge P \vdash s : b$
using *assms* **proof**(*nominal-induct v arbitrary*: b *rule*:*v.strong-induct*)
case (*V-lit x*)
then show *?case* **using** *eval-l-base eval-v.intros eval-l.simps wfV-elims rcl-val.supp pure-supp* **by** *metis*
next
case (*V-var x*)
then obtain c **where** $*$:*Some* (b,c) = *lookup* Γ x **using** *wfV-elims* **by** *metis*
hence $x \in$ *fst* ' *setG* Γ **using** *lookup-x* **by** *blast*
hence $x \in$ *dom* i **using** *wfI-domi* **using** *assms* **by** *blast*
then obtain s **where** $i\ x$ = *Some* s **by** *auto*
moreover hence $P \vdash s : b$ **using** *wfRCV.intros wfI-lookup* $*$ *V-var*
by (*metis wfV-wf*)

ultimately show *?case* **using** *eval-v.intros rcl-val.supp b.supp* **by** *metis*
next
case (*V-pair v1 v2*)
then obtain *b1* **and** *b2* **where** $*$:P ; B ; $\Gamma \vdash_{wf} v1 : b1 \wedge$ P ; B ; $\Gamma \vdash_{wf} v2 : b2 \wedge b$ = *B-pair b1 b2* **using** *wfV-elims* **by** *metis*
then obtain *s1* **and** *s2* **where** *eval-v i v1 s1* $\wedge$ *wfRCV P s1 b1* $\wedge$ *eval-v i v2 s2* $\wedge$ *wfRCV P s2 b2* **using** *V-pair* **by** *metis*
thus *?case* **using** *eval-v.intros wfRCV.intros* $*$ **by** *metis*
next
case (*V-cons tyid dc v*)
then obtain s **and** b'::b **and** *dclist* **and** x::x **and** c::c **where** (*wfV P B* Γ v b') $\wedge$ $i \llbracket v \rrbracket^{\sim}$ $s \wedge$ $P \vdash s : b' \wedge b$ = *B-id tyid* $\wedge$
AF-typedef tyid dclist $\in$ *set* $P \wedge$ $(dc, \{\!|\ x : b'\ |\ c\ |\!\}) \in$ *set dclist* **using** *wfV-elims*(*4*) **by** *metis*
then show *?case* **using** *eval-v.intros*(*4*) *wfRCV.intros*(*5*) *V-cons* **by** *metis*
next
case (*V-consp tyid dc b' v*)

obtain $b'a$::b **and** *bv* **and** *dclist* **and** x::x **and** c::c **where** $*$:(*wfV P B* Γ v $b'a[bv::=b']_{bb}$) $\wedge$ b = *B-app tyid b'* $\wedge$
AF-typedef-poly tyid bv dclist $\in$ *set* $P \wedge$ $(dc, \{\!|\ x : b'a\ |\ c\ |\!\}) \in$ *set dclist* $\wedge$
atom bv $\sharp$ (P, *B-app tyid b'*,B) **using** *wf-strong-elim*(*1*)[*OF V-consp*(*3*)] **by** *metis*

then obtain s **where** $**$:$i \llbracket v \rrbracket^{\sim}$ s $\wedge$ $P \vdash s : b'a[bv::=b']_{bb}$ **using** *V-consp* **by** *auto*

have $\vdash_{wf} P$ **using** *wfX-wfY V-consp* **by** *metis*
then obtain *bv1*::*bv* **and** *dclist1 x1 b1 c1* **where** *3*:*AF-typedef-poly tyid bv dclist* = *AF-typedef-poly tyid bv1 dclist1*
$\wedge$ $(dc, \{\!|\ x1 : b1\ |\ c1\ |\!\}) \in$ *set dclist1* $\wedge$ *atom bv1* $\sharp$ (P, *SConsp tyid dc b' s*, *B-app tyid b'*)

```
        ∧ b'a[bv::=b']_bb = b1[bv1::=b']_bb
    using ∗ obtain-fresh-bv-dclist-b-iff by blast

  have i ⟦ V-consp tyid dc b' v ⟧ ~ SConsp tyid dc b' s using eval-v.intros by (simp add: ∗∗)

  moreover have P ⊢ SConsp tyid dc b' s : B-app tyid b' proof
    show ‹AF-typedef-poly tyid bv1 dclist1 ∈ set P› using 3 ∗ by metis
    show ‹(dc, {| x1 : b1 | c1 |}) ∈ set dclist1› using 3 by auto
    thus ‹atom bv1 ♯ (P, SConsp tyid dc b' s, B-app tyid b')› using ∗ 3 fresh-prodN by metis
    show ‹ P ⊢ s : b1[bv1::=b']_bb› using 3 ∗∗ by auto
  qed

  ultimately show ?case using eval-v.intros wfRCV.intros V-consp ∗ by auto
qed

lemma eval-v-uniqueness:
  fixes v::v
  assumes i ⟦ v ⟧ ~ s and i ⟦ v ⟧ ~ s'
  shows s=s'
using assms proof(nominal-induct v arbitrary: s s' rule:v.strong-induct)
  case (V-lit x)
  then show ?case using eval-v-elims eval-l.simps by metis
next
  case (V-var x)
  then show ?case using eval-v-elims  by (metis option.inject)
next
  case (V-pair v1 v2)
  obtain s1 and s2 where s:i ⟦ v1 ⟧ ~ s1 ∧ i ⟦ v2 ⟧ ~ s2 ∧ s = SPair s1 s2 using eval-v-elims
V-pair by metis
  obtain s1' and s2' where s':i ⟦ v1 ⟧ ~ s1' ∧ i ⟦ v2 ⟧ ~ s2' ∧ s' = SPair s1' s2' using eval-v-elims
V-pair by metis
  then show ?case using eval-v-elims using V-pair s s' by auto
next
  case (V-cons tyid dc v1)
  obtain sv1 where 1:i ⟦ v1 ⟧ ~ sv1 ∧ s = SCons tyid dc sv1 using eval-v-elims V-cons by metis
  moreover obtain sv2 where 2:i ⟦ v1 ⟧ ~ sv2 ∧ s' = SCons tyid dc sv2 using eval-v-elims V-cons
by metis
  ultimately have sv1 = sv2 using V-cons by auto
  then show ?case using 1 2 by auto
next
  case (V-consp tyid dc b v1)
  then show ?case using eval-v-elims by metis

qed

lemma eval-v-base:
  fixes Γ::Γ and v::v and b::b
  assumes P ; Γ ⊢ i and P ; B ; Γ ⊢_wf v : b and i ⟦ v ⟧ ~ s
  shows P ⊢ s : b
  using eval-v-exist eval-v-uniqueness assms by metis
```

lemma *eval-e-uniqueness*:
fixes *e*::*ce*
assumes $i \llbracket e \rrbracket \sim s$ **and** $i \llbracket e \rrbracket \sim s'$
shows $s{=}s'$
using *assms* **proof**(*nominal-induct e arbitrary*: *s s' rule*:*ce.strong-induct*)
case (*CE-val x*)
then show *?case* **using** *eval-v-uniqueness eval-e-elims* **by** *metis*
next
case (*CE-op opp x1 x2*)
consider $opp = Plus \mid opp = LEq$ **using** *opp.exhaust* **by** *metis*
thus *?case* **proof**(*cases*)
case *1*
hence *a1*:*eval-e i* (*CE-op Plus x1 x2*) *s* **and** *a2*:*eval-e i* (*CE-op Plus x1 x2*) s' **using** *CE-op* **by** *auto*
then show *?thesis* **using** *eval-e-elims*(*2*)[*OF a1*] *eval-e-elims*(*2*)[*OF a2*] *CE-op eval-e-plusI*
by (*metis rcl-val.eq-iff*(*2*))
next
case *2*
hence *a1*:*eval-e i* (*CE-op LEq x1 x2*) *s* **and** *a2*:*eval-e i* (*CE-op LEq x1 x2*) s' **using** *CE-op* **by** *auto*
thm *eval-e-elims*(*2*)
then show *?thesis* **using** *eval-v-uniqueness eval-e-elims*(*3*)[*OF a1*] *eval-e-elims*(*3*)[*OF a2*] *CE-op eval-e-plusI*
by (*metis rcl-val.eq-iff*(*2*))
qed
next
case (*CE-concat x1 x2*)
hence *a1*:*eval-e i* (*CE-concat x1 x2*) *s* **and** *a2*:*eval-e i* (*CE-concat x1 x2*) s' **using** *CE-concat* **by** *auto*
show *?case* **using** *eval-e-elims*(*6*)[*OF a1*] *eval-e-elims*(*6*)[*OF a2*] *CE-concat eval-e-concatI rcl-val.eq-iff*

proof −
assume $\bigwedge P.\ (\bigwedge bv1\ bv2.\ [\![s' = SBitvec\ (bv1\ @\ bv2);\ i \llbracket x1 \rrbracket \sim SBitvec\ bv1\ ;\ i \llbracket x2 \rrbracket \sim SBitvec\ bv2]\!] \Longrightarrow P) \Longrightarrow P$
obtain *bbs* :: *bit list* **and** *bbsa* :: *bit list* **where**
$i \llbracket x2 \rrbracket \sim SBitvec\ bbs \wedge i \llbracket x1 \rrbracket \sim SBitvec\ bbsa \wedge SBitvec\ (bbsa\ @\ bbs) = s'$
by (*metis* ‹$\bigwedge P.\ (\bigwedge bv1\ bv2.\ [\![s' = SBitvec\ (bv1\ @\ bv2);\ i \llbracket x1 \rrbracket \sim SBitvec\ bv1\ ;\ i \llbracket x2 \rrbracket \sim SBitvec\ bv2]\!] \Longrightarrow P) \Longrightarrow P$›)
then have $s' = s$
by (*metis* (*no-types*) ‹$\bigwedge P.\ (\bigwedge bv1\ bv2.\ [\![s = SBitvec\ (bv1\ @\ bv2);\ i \llbracket x1 \rrbracket \sim SBitvec\ bv1\ ;\ i \llbracket x2 \rrbracket \sim SBitvec\ bv2]\!] \Longrightarrow P) \Longrightarrow P$› ‹$\bigwedge s'\ s.\ [\![i \llbracket x1 \rrbracket \sim s\ ;\ i \llbracket x1 \rrbracket \sim s']\!] \Longrightarrow s = s'$› ‹$\bigwedge s'\ s.\ [\![i \llbracket x2 \rrbracket \sim s\ ;\ i \llbracket x2 \rrbracket \sim s']\!] \Longrightarrow s = s'$› *rcl-val.eq-iff*(*1*))
then show *?thesis*
by *metis*
qed

next
case (*CE-fst x*)
then show *?case* **using** *eval-v-uniqueness* **by** (*meson eval-e-elims rcl-val.eq-iff*)
next
case (*CE-snd x*)

then show *?case* **using** *eval-v-uniqueness* **by** (*meson eval-e-elims rcl-val.eq-iff*)
next
case (*CE-len x*)
then show *?case* **using** *eval-e-elims rcl-val.eq-iff*
proof −
obtain *bbs* :: *rcl-val* ⇒ *ce* ⇒ (*x* ⇒ *rcl-val option*) ⇒ *bit list* **where**
∀ *x0 x1 x2*. (∃ *v3*. *x0* = *SNum* (*int* (*length v3*)) ∧ *x2* ⟦ *x1* ⟧ ~ *SBitvec v3*) = (*x0* = *SNum* (*int* (*length* (*bbs x0 x1 x2*))) ∧ *x2* ⟦ *x1* ⟧ ~ *SBitvec* (*bbs x0 x1 x2*))
by *moura*
then have ∀ *f c r*. ¬ *f* ⟦ [| *c* |]^{ce} ⟧ ~ *r* ∨ *r* = *SNum* (*int* (*length* (*bbs r c f*))) ∧ *f* ⟦ *c* ⟧ ~ *SBitvec* (*bbs r c f*)
by (*meson eval-e-elims*(*7*))
then show *?thesis*
by (*metis* (*no-types*) *CE-len.hyps CE-len.prems*(*1*) *CE-len.prems*(*2*) *rcl-val.eq-iff*(*1*))
qed

qed

lemma *wfV-eval-bitvec*:
fixes *v*::*v*
assumes *P* ; *B* ; Γ ⊢$_{wf}$ *v* : *B-bitvec* **and** *P* ; Γ ⊢ *i*
shows ∃ *bv*. *eval-v i v* (*SBitvec bv*)
proof −
obtain *s* **where** *i* ⟦ *v* ⟧ ~ *s* ∧ *wfRCV P s B-bitvec* **using** *eval-v-exist assms* **by** *metis*
moreover then obtain *bv* **where** *s* = *SBitvec bv* **using** *wfRCV-elims*(*1*)[*of P s*] **by** *metis*
ultimately show *?thesis* **by** *metis*
qed

lemma *wfV-eval-pair*:
fixes *v*::*v*
assumes *P* ; *B* ; Γ ⊢$_{wf}$ *v* : *B-pair b1 b2* **and** *P* ; Γ ⊢ *i*
shows ∃ *s1 s2*. *eval-v i v* (*SPair s1 s2*)
proof −
obtain *s* **where** *i* ⟦ *v* ⟧ ~ *s* ∧ *wfRCV P s* (*B-pair b1 b2*) **using** *eval-v-exist assms* **by** *metis*
moreover then obtain *s1* **and** *s2* **where** *s* = *SPair s1 s2* **using** *wfRCV-elims*(*2*)[*of P s*] **by** *metis*
ultimately show *?thesis* **by** *metis*
qed

lemma *wfV-eval-int*:
fixes *v*::*v*
assumes *P* ; *B* ; Γ ⊢$_{wf}$ *v* : *B-int* **and** *P* ; Γ ⊢ *i*
shows ∃ *n*. *eval-v i v* (*SNum n*)
proof −
obtain *s* **where** *i* ⟦ *v* ⟧ ~ *s* ∧ *wfRCV P s* (*B-int*) **using** *eval-v-exist assms* **by** *metis*
moreover then obtain *n* **where** *s* = *SNum n* **using** *wfRCV-elims*(*3*)[*of P s*] **by** *metis*
ultimately show *?thesis* **by** *metis*
qed

Well sorted value with a well sorted valuation evaluates

lemma *wfI-wfV-eval-v*:
fixes *v*::*v* **and** *b*::*b*

assumes Θ ; B ; $\Gamma \vdash_{wf} v : b$ **and** *wfI* Θ Γ i
shows $\exists s.\ i \llbracket v \rrbracket \sim s \wedge \Theta \vdash s : b$
using *eval-v-exist assms* **by** *auto*

lemma *wfI-wfCE-eval-e*:
fixes *e*::*ce* **and** *b*::*b*
assumes *wfCE P B G e b* **and** *P* ; *G* $\vdash$ *i*
shows $\exists s.\ i \llbracket e \rrbracket \sim s \wedge P \vdash s : b$
using *assms* **proof**(*nominal-induct e arbitrary*: *b rule*: *ce.strong-induct*)
case (*CE-val v*)
obtain *s* **where** $i \llbracket v \rrbracket \sim s \wedge P \vdash s : b$ **using** *wfI-wfV-eval-v*[*of P B G v b i*] *assms wfCE-elims*(*1*)[*of P B G v b*] *CE-val* **by** *auto*
then show *?case* **using** *CE-val eval-e.intros*(*1*)[*of i v s*] **by** *auto*
next
case (*CE-op opp v1 v2*)
hence *wfCE P B G v1 B-int* $\wedge$ *wfCE P B G v2 B-int* **using** *wfCE-elims*
by (*metis* (*full-types*) *opp.strong-exhaust*)
then obtain *s1* **and** *s2* **where** $*$: *eval-e i v1 s1* $\wedge$ *wfRCV P s1 B-int* $\wedge$ *eval-e i v2 s2* $\wedge$ *wfRCV P s2 B-int*
using *wfI-wfV-eval-v CE-op* **by** *metis*
then obtain *n1* **and** *n2* **where** $**$:*s2*=*SNum n2* $\wedge$ *s1* = *SNum n1* **using** *wfRCV-elims* **by** *meson*
consider *opp* =*Plus* | *opp*=*LEq* **using** *opp.exhaust* **by** *auto*

thus *?case* **proof**(*cases*)
case *1*
hence *eval-e i* (*CE-op Plus v1 v2*) (*SNum* (*n1*+*n2*)) **using** *eval-e-plusI* $*$ $**$ **by** *simp*
moreover have *wfRCV P* (*SNum* (*n1*+*n2*)) *B-int* **using** *wfRCV.intros* **by** *auto*
ultimately show *?thesis* **using** *1*
using *CE-op.prems*(*1*) *wfCE-elims*(*2*) **by** *blast*
next
case *2*
hence *eval-e i* (*CE-op LEq v1 v2*) (*SBool* (*n1* $\leq$ *n2*)) **using** *eval-e-leqI* $*$ $**$ **by** *simp*
moreover have *wfRCV P* (*SBool* (*n1*$\leq$*n2*)) *B-bool* **using** *wfRCV.intros* **by** *auto*
ultimately show *?thesis* **using** *2*
using *CE-op.prems wfCE-elims* **by** *metis*
qed
next
case (*CE-concat v1 v2*)
then obtain *s1* **and** *s2* **where** $*$:*b* = *B-bitvec* $\wedge$ *eval-e i v1 s1* $\wedge$ *eval-e i v2 s2* $\wedge$
wfRCV P s1 B-bitvec $\wedge$ *wfRCV P s2 B-bitvec* **using**
CE-concat
by (*meson wfCE-elims*(*6*))
thus *?case* **using** *eval-e-concatI wfRCV.intros*(*1*) *wfRCV-elims*
proof −
obtain *bbs* :: *type-def list* $\Rightarrow$ *rcl-val* $\Rightarrow$ *bit list* **where**
$\forall$ *ts s*. $\neg$ *ts* $\vdash$ *s* : *B-bitvec* $\vee$ *s* = *SBitvec* (*bbs ts s*)
using *wfRCV-elims*(*1*) **by** *moura*
then show *?thesis*
by (*metis* (*no-types*) *local.*$*$ *wfRCV-BBitvecI eval-e-concatI*)
qed
next
case (*CE-fst v1*)

thus *?case* **using** *eval-e-fstI wfRCV.intros wfCE-elims wfI-wfV-eval-v*
by (*metis wfRCV-elims(2) rcl-val.eq-iff(4)*)
next
case (*CE-snd v1*)
thus *?case* **using** *eval-e-sndI wfRCV.intros wfCE-elims wfI-wfV-eval-v*
by (*metis wfRCV-elims(2) rcl-val.eq-iff(4)*)
next
case (*CE-len x*)
thus *?case* **using** *eval-e-lenI wfRCV.intros wfCE-elims wfI-wfV-eval-v wfV-eval-bitvec*
by (*metis wfRCV-elims(1)*)
qed

lemma *eval-e-exist*:
fixes Γ::Γ **and** *e*::*ce*
assumes P ; Γ ⊢ *i* **and** P ; B ; Γ ⊢$_{wf}$ *e* : *b*
shows ∃ *s*. *i* ⟦ *e* ⟧ ~ *s*
using *assms* **proof**(*nominal-induct e arbitrary*: *b rule*:*ce.strong-induct*)
case (*CE-val v*)
then show *?case* **using** *eval-v-exist wfCE-elims eval-e.intros* **by** *metis*
next
case (*CE-op op v1 v2*)

show *?case* **proof**(*rule opp.exhaust*)
assume ‹*op* = *Plus*›
hence P ; B ; Γ ⊢$_{wf}$ *v1* : *B-int* ∧ P ; B ; Γ ⊢$_{wf}$ *v2* : *B-int* ∧ *b* = *B-int* **using** *wfCE-elims CE-op* **by** *metis*
then obtain *n1* **and** *n2* **where** *eval-e i v1* (*SNum n1*) ∧ *eval-e i v2* (*SNum n2*) **using** *CE-op eval-v-exist wfV-eval-int*
by (*metis wfI-wfCE-eval-e wfRCV-elims(3)*)
then show ‹∃ *a*. *eval-e i* (*CE-op op v1 v2*) *a*› **using** *eval-e-plusI*[*of i v1 - v2*] ‹*op*=*Plus*› **by** *auto*
next
assume ‹*op* = *LEq*›
hence P ; B ; Γ ⊢$_{wf}$ *v1* : *B-int* ∧ P ; B ; Γ ⊢$_{wf}$ *v2* : *B-int* ∧ *b* = *B-bool* **using** *wfCE-elims CE-op* **by** *metis*
then obtain *n1* **and** *n2* **where** *eval-e i v1* (*SNum n1*) ∧ *eval-e i v2* (*SNum n2*) **using** *CE-op eval-v-exist wfV-eval-int*
by (*metis wfI-wfCE-eval-e wfRCV-elims(3)*)
then show ‹∃ *a*. *eval-e i* (*CE-op op v1 v2*) *a*› **using** *eval-e-leqI*[*of i v1 - v2*] *eval-v-exist* ‹*op*=*LEq*› *CE-op* **by** *auto*
qed
next
case (*CE-concat v1 v2*)
then obtain *bv1* **and** *bv2* **where** *eval-e i v1* (*SBitvec bv1*) ∧ *eval-e i v2* (*SBitvec bv2*)
using *wfV-eval-bitvec wfCE-elims(6)*
by (*meson eval-e-elims(6) wfI-wfCE-eval-e*)
then show *?case* **using** *eval-e.intros* **by** *metis*
next
case (*CE-fst ce*)
then obtain *b2* **where** *b*:*P* ; *B* ; Γ ⊢$_{wf}$ *ce* : *B-pair b b2* **using** *wfCE-elims* **by** *metis*
then obtain *s* **where** *s*:*i* ⟦ *ce* ⟧ ~ *s* **using** *CE-fst* **by** *auto*
then obtain *s1* **and** *s2* **where** *s* = (*SPair s1 s2*) **using** *eval-e-elims(4) CE-fst wfI-wfCE-eval-e*[*of P B Γ ce B-pair b b2 i*,*OF b*] *wfRCV-elims(2)*[*of P s b b2*]

by (*metis eval-e-uniqueness*)
then show *?case* **using** *s eval-e.intros* **by** *metis*
next
case (*CE-snd ce*)
then obtain *b1* **where** *b*:*P* ; *B* ; Γ ⊢$_{wf}$ *ce* : *B-pair b1 b* **using** *wfCE-elims* **by** *metis*
then obtain *s* **where** *s*:*i* ⟦ *ce* ⟧ ~ *s* **using** *CE-snd* **by** *auto*
then obtain *s1* **and** *s2* **where** *s* = (*SPair s1 s2*)
using *eval-e-elims*(*5*) *CE-snd wfI-wfCE-eval-e*[*of P B* Γ *ce B-pair b1 b i*,*OF b*] *wfRCV-elims*(*2*)[*of P s b b1*]
eval-e-uniqueness
by (*metis wfRCV-elims*(*2*))
then show *?case* **using** *s eval-e.intros* **by** *metis*
next
case (*CE-len v1*)
then obtain *bv1* **where** *eval-e i v1* (*SBitvec bv1*)
using *wfV-eval-bitvec CE-len wfCE-elims eval-e-uniqueness*
by (*metis eval-e-elims*(*6*) *wfCE-concatI wfI-wfCE-eval-e*)
then show *?case* **using** *eval-e.intros* **by** *metis*
qed

lemma *eval-c-exist*:
fixes Γ::Γ **and** *c*::*c*
assumes *P* ; Γ ⊢ *i* **and** *P* ; *B* ; Γ ⊢$_{wf}$ *c*
shows ∃ *s*. *i* ⟦ *c* ⟧ ~ *s*
using *assms* **proof**(*nominal-induct c rule*: *c.strong-induct*)
case *C-true*
then show *?case* **using** *eval-c.intros wfC-elims* **by** *metis*
next
case *C-false*
then show *?case* **using** *eval-c.intros wfC-elims* **by** *metis*
next
case (*C-conj c1 c2*)
then show *?case* **using** *eval-c.intros wfC-elims* **by** *metis*
next
case (*C-disj x1 x2*)
then show *?case* **using** *eval-c.intros wfC-elims* **by** *metis*
next
case (*C-not x*)
then show *?case* **using** *eval-c.intros wfC-elims* **by** *metis*
next
case (*C-imp x1 x2*)
then show *?case* **using** *eval-c.intros eval-e-exist wfC-elims* **by** *metis*
next
case (*C-eq x1 x2*)
then show *?case* **using** *eval-c.intros eval-e-exist wfC-elims* **by** *metis*
qed

lemma *eval-c-uniqueness*:
fixes *c*::*c*
assumes *i* ⟦ *c* ⟧ ~ *s* **and** *i* ⟦ *c* ⟧ ~ *s*′
shows *s*=*s*′

using *assms* **proof**(*nominal-induct c arbitrary*: *s s'* *rule*:*c.strong-induct*)
case *C-true*
then show *?case* **using** *eval-c-elims* **by** *metis*
next
case *C-false*
then show *?case* **using** *eval-c-elims* **by** *metis*
next
case (*C-conj x1 x2*)
then show *?case* **using** *eval-c-elims*(*3*) **by** (*metis* (*full-types*))
next
case (*C-disj x1 x2*)
then show *?case* **using** *eval-c-elims*(*4*) **by** (*metis* (*full-types*))
next
case (*C-not x*)
then show *?case* **using** *eval-c-elims*(*6*) **by** (*metis* (*full-types*))
next
case (*C-imp x1 x2*)
then show *?case* **using** *eval-c-elims*(*5*) **by** (*metis* (*full-types*))
next
case (*C-eq x1 x2*)
then show *?case* **using** *eval-e-uniqueness eval-c-elims*(*7*) **by** *metis*
qed

lemma *wfI-wfC-eval-c*:
fixes *c*::*c*
assumes *wfC P B G c* **and** *P* ; *G* ⊢ *i*
shows ∃ *s*. *i* ⟦ *c* ⟧ ~ *s*
using *assms* **proof**(*nominal-induct c* *rule*: *c.strong-induct*)
qed(*metis wfC-elims wfI-wfCE-eval-e eval-c.intros*)+

11.3 Satisfiability

lemma *satis-reflI*:
fixes *c*::*c*
assumes *i* ⊨ ((*x*, *b*, *c*) #Γ *G*)
shows *i* ⊨ *c*
using *assms* **by** *auto*

lemma *is-satis-mp*:
fixes *c1*::*c* **and** *c2*::*c*
assumes *i* ⊨ (*c1 IMP c2*) **and** *i* ⊨ *c1*
shows *i* ⊨ *c2*
using *assms* **proof** −
have *eval-c i* (*c1 IMP c2*) *True* **using** *is-satis.simps* **using** *assms* **by** *blast*
then obtain *b1* **and** *b2* **where** *True* = (*b1* ⟶ *b2*) ∧ *eval-c i c1 b1* ∧ *eval-c i c2 b2*
using *eval-c-elims*(*5*) **by** *metis*
moreover have *eval-c i c1 True* **using** *is-satis.simps* **using** *assms* **by** *blast*
moreover have *b1* = *True* **using** *calculation eval-c-uniqueness* **by** *blast*
ultimately have *eval-c i c2 True* **by** *auto*
thus *?thesis* **using** *is-satis.intros* **by** *auto*
qed

lemma *is-satis-imp*:
fixes *c1*::*c* **and** *c2*::*c*
assumes *i* ⊨ *c1* ⟶ *i* ⊨ *c2* **and** *i* ⟦ *c1* ⟧ ~ *b1* **and** *i* ⟦ *c2* ⟧ ~ *b2*
shows *i* ⊨ (*c1 IMP c2*)
proof(*cases b1*)
case *True*
hence *i* ⊨ *c2* **using** *assms is-satis.simps* **by** *simp*
hence *b2* = *True* **using** *is-satis.simps assms*
using *eval-c-uniqueness* **by** *blast*
then show *?thesis* **using** *eval-c-impI is-satis.simps assms* **by** *force*
next
case *False*
then show *?thesis* **using** *assms eval-c-impI is-satis.simps* **by** *metis*
qed

lemma *is-satis-iff*:
i ⊨ *G* = (∀ *x b c*. (*x*,*b*,*c*) ∈ *setG G* ⟶ *i* ⊨ *c*)
by(*induct G*,*auto*)

lemma *is-satis-g-append*:
i ⊨ (*G1*@*G2*) = (*i* ⊨ *G1* ∧ *i* ⊨ *G2*)
using *is-satis-g.simps is-satis-iff* **by** *auto*

11.4 Substitution for Evaluation

lemma *eval-v-i-upd*:
fixes *v*::*v*
assumes *atom x* ♯ *v* **and** *i* ⟦ *v* ⟧ ~ *s'*
shows *eval-v* ((*i* (*x* ↦*s*))) *v s'*
using *assms* **proof**(*nominal-induct v arbitrary*: *s' rule*:*v.strong-induct*)
case (*V-lit x*)
then show *?case* **by** (*metis eval-v-elims*(*1*) *eval-v-litI*)
next
case (*V-var y*)
then obtain *s* **where** ∗: *Some s* = *i y* ∧ *s*=*s'* **using** *eval-v-elims* **by** *metis*
moreover have *x* ≠ *y* **using** ‹*atom x* ♯ *V-var y*› *v.supp* **by** *simp*
ultimately have (*i* (*x* ↦*s*)) *y* = *Some s*
by (*simp add*: ‹*Some s* = *i y* ∧ *s* = *s'*›)
then show *?case* **using** *eval-v-varI* ∗ ‹*x* ≠ *y*›
by (*simp add*: *eval-v.eval-v-varI*)
next
case (*V-pair v1 v2*)
hence *atom x* ♯ *v1* ∧ *atom x* ♯ *v2* **using** *v.supp* **by** *simp*
moreover obtain *s1* **and** *s2* **where** ∗: *eval-v i v1 s1* ∧ *eval-v i v2 s2* ∧ *s'* = *SPair s1 s2* **using** *eval-v-elims V-pair* **by** *metis*
ultimately have *eval-v* ((*i* (*x* ↦*s*))) *v1 s1* ∧ *eval-v* ((*i* (*x* ↦*s*))) *v2 s2* **using** *V-pair* **by** *blast*
thus *?case* **using** *eval-v-pairI* ∗ **by** *meson*
next
case (*V-cons tyid dc v1*)
hence *atom x* ♯ *v1* **using** *v.supp* **by** *simp*
moreover obtain *s1* **where** ∗: *eval-v i v1 s1* ∧ *s'* = *SCons tyid dc s1* **using** *eval-v-elims V-cons* **by**

metis
ultimately have *eval-v* ((*i* (*x* ↦*s*))) *v1 s1* **using** *V-cons* **by** *blast*
thus *?case* **using** *eval-v-consI* ∗ **by** *meson*
next
case (*V-consp tyid dc b1 v1*)

hence *atom x* ♯ *v1* **using** *v.supp* **by** *simp*
moreover obtain *s1* **where** ∗: *eval-v i v1 s1* ∧ *s′* = *SConsp tyid dc b1 s1* **using** *eval-v-elims V-consp*
by *metis*
ultimately have *eval-v* ((*i* (*x* ↦*s*))) *v1 s1* **using** *V-consp* **by** *blast*
thus *?case* **using** *eval-v-conspI* ∗ **by** *meson*
qed

lemma *eval-e-i-upd*:
fixes *e*::*ce*
assumes *i* ⟦ *e* ⟧ ~ *s′* **and** *atom x* ♯ *e*
shows (*i* (*x* ↦*s*)) ⟦ *e* ⟧ ~ *s′*
using *assms* **apply**(*induct rule*: *eval-e.induct*) **using** *eval-v-i-upd eval-e-elims*
by (*meson ce.fresh eval-e.intros*)+

lemma *eval-c-i-upd*:
fixes *c*::*c*
assumes *i* ⟦ *c* ⟧ ~ *s′* **and** *atom x* ♯ *c*
shows ((*i* (*x* ↦*s*))) ⟦ *c* ⟧ ~ *s′*
using *assms* **proof**(*induct rule*:*eval-c.induct*)
case (*eval-c-eqI i e1 sv1 e2 sv2*)
then show *?case* **using** *RCLogic.eval-c-eqI eval-e-i-upd c.fresh* **by** *metis*
qed(*simp add*: *eval-c.intros*)+

lemma *subst-v-eval-v*[*simp*]:
fixes *v*::*v* **and** *v′*::*v*
assumes *i* ⟦ *v* ⟧ ~ *s* **and** *i* ⟦ (*v′*[*x*::=*v*]$_{vv}$) ⟧ ~ *s′*
shows (*i* (*x* ↦ *s*)) ⟦ *v′* ⟧ ~ *s′*
using *assms* **proof**(*nominal-induct v′ arbitrary*: *s′ rule*:*v.strong-induct*)
case (*V-lit x*)
then show *?case* **using** *subst-vv.simps*
by (*metis eval-v-elims*(*1*) *eval-v-litI*)
next
case (*V-var x′*)
then show *?case* **proof**(*cases x*=*x′*)
case *True*
hence (*V-var x′*)[*x*::=*v*]$_{vv}$ = *v* **using** *subst-vv.simps* **by** *auto*
then show *?thesis* **using** *V-var eval-v-elims eval-v-varI eval-v-uniqueness True*
by (*simp add*: *eval-v.eval-v-varI*)
next
case *False*
hence *atom x* ♯ (*V-var x′*) **by** *simp*
then show *?thesis* **using** *eval-v-i-upd False V-var* **by** *fastforce*
qed
next
case (*V-pair v1 v2*)
then obtain *s1* **and** *s2* **where** ∗:*eval-v i* (*v1*[*x*::=*v*]$_{vv}$) *s1* ∧ *eval-v i* (*v2*[*x*::=*v*]$_{vv}$) *s2* ∧ *s′* = *SPair*

s1 s2 **using** *V-pair eval-v-elims subst-vv.simps* **by** *metis*
hence $(i\ (\ x \mapsto s\ ))\ [\![\ v1\ ]\!] \sim s1 \wedge (i\ (\ x \mapsto s\ ))\ [\![\ v2\ ]\!] \sim s2$ **using** *V-pair* **by** *metis*
thus *?case* **using** *eval-v-pairI subst-vv.simps* $*$ *V-pair* **by** *metis*
next
case (*V-cons tyid dc v1*)
then obtain *s1* **where** *eval-v i* $(v1[x::=v]_{vv})$ *s1* **using** *eval-v-elims subst-vv.simps* **by** *metis*
thus *?case* **using** *eval-v-consI V-cons*
by (*metis eval-v-elims subst-vv.simps*)
next
case (*V-consp tyid dc b1 v1*)

then obtain *s1* **where** $*$:*eval-v i* $(v1[x::=v]_{vv})$ *s1* $\wedge$ $s' = SConsp\ tyid\ dc\ b1\ s1$ **using** *eval-v-elims subst-vv.simps* **by** *metis*
hence $i\ (\ x \mapsto s\ )\ [\![\ v1\ ]\!] \sim s1$ **using** *V-consp* **by** *metis*
thus *?case* **using** $*$ *eval-v-conspI* **by** *metis*
qed

lemma *subst-e-eval-v*[*simp*]:
fixes *y*::*x* **and** *e*::*ce* **and** *v*::*v* **and** *e'*::*ce*
assumes $i\ [\![\ e'\ ]\!] \sim s'$ **and** $e'=(e[y::=v]_{cev})$ **and** $i\ [\![\ v\ ]\!] \sim s$
shows $(i\ (\ y \mapsto s\ ))\ [\![\ e\ ]\!] \sim s'$
using *assms* **proof**(*induct arbitrary*: *e rule*: *eval-e.induct*)
case (*eval-e-valI i v1 sv*)
then obtain *v1'* **where** $*$:$e = CE\text{-}val\ v1' \wedge v1 = v1'[y::=v]_{vv}$
using *assms* **by**(*nominal-induct e rule*:*ce.strong-induct*,*simp*+)
hence *eval-v i* $(v1'[y::=v]_{vv})$ *sv* **using** *eval-e-valI* **by** *simp*
hence *eval-v* $(i\ (\ y \mapsto s\ ))$ *v1' sv* **using** *subst-v-eval-v eval-e-valI* **by** *simp*
then show *?case* **using** *RCLogic.eval-e-valI* $*$ **by** *meson*
next
case (*eval-e-plusI i v1 n1 v2 n2*)
then obtain *v1'* **and** *v2'* **where** $*$:$e = CE\text{-}op\ Plus\ v1'\ v2' \wedge v1 = v1'[y::=v]_{cev} \wedge v2 = v2'[y::=v]_{cev}$
using *assms* **by**(*nominal-induct e rule*:*ce.strong-induct*,*simp*+)
hence *eval-e i* $(v1'[y::=v]_{cev})$ $(SNum\ n1)$ $\wedge$ *eval-e i* $(v2'[y::=v]_{cev})$ $(SNum\ n2)$ **using** *eval-e-plusI* **by** *simp*
hence *eval-e* $(i\ (\ y \mapsto s\ ))$ *v1'* $(SNum\ n1)$ $\wedge$ *eval-e* $(i\ (\ y \mapsto s\ ))$ *v2'* $(SNum\ n2)$ **using** *subst-v-eval-v eval-e-plusI*
using *local.*$*$ **by** *blast*
then show *?case* **using** *RCLogic.eval-e-plusI* $*$ **by** *meson*
next
case (*eval-e-leqI i v1 n1 v2 n2*)
then obtain *v1'* **and** *v2'* **where** $*$:$e = CE\text{-}op\ LEq\ v1'\ v2' \wedge v1 = v1'[y::=v]_{cev} \wedge v2 = v2'[y::=v]_{cev}$
using *assms* **by**(*nominal-induct e rule*:*ce.strong-induct*,*simp*+)
hence *eval-e i* $(v1'[y::=v]_{cev})$ $(SNum\ n1)$ $\wedge$ *eval-e i* $(v2'[y::=v]_{cev})$ $(SNum\ n2)$ **using** *eval-e-leqI* **by** *simp*
hence *eval-e* $(i\ (\ y \mapsto s\ ))$ *v1'* $(SNum\ n1)$ $\wedge$ *eval-e* $(i\ (\ y \mapsto s\ ))$ *v2'* $(SNum\ n2)$ **using** *subst-v-eval-v eval-e-leqI*
using $*$ **by** *blast*
then show *?case* **using** *RCLogic.eval-e-leqI* $*$ **by** *meson*
next
case (*eval-e-fstI i v1 s1 s2*)
then obtain *v1'* **and** *v2'* **where** $*$:$e = CE\text{-}fst\ v1' \wedge v1 = v1'[y::=v]_{cev}$
using *assms* **by**(*nominal-induct e rule*:*ce.strong-induct*,*simp*+)

hence *eval-e i* $(v1'[y::=v]_{cev})$ (*SPair s1 s2*) **using** *eval-e-fstI* **by** *simp*
hence *eval-e* $(i\ (\ y \mapsto s\ ))$ *v1'* (*SPair s1 s2*) **using** *eval-e-fstI* $*$ **by** *metis*
then show *?case* **using** *RCLogic.eval-e-fstI* $*$ **by** *meson*
next
case (*eval-e-sndI i v1 s1 s2*)
then obtain *v1'* **and** *v2'* **where** $*$:*e* = *CE-snd v1'* $\wedge$ *v1* = *v1'*$[y::=v]_{cev}$
using *assms* **by**(*nominal-induct e rule*:*ce.strong-induct*,*simp*+)
hence *eval-e i* $(v1'[y::=v]_{cev})$ (*SPair s1 s2*) **using** *eval-e-sndI* **by** *simp*
hence *eval-e* $(i\ (\ y \mapsto s\ ))$ *v1'* (*SPair s1 s2*) **using** *subst-v-eval-v eval-e-sndI* $*$ **by** *blast*
then show *?case* **using** *RCLogic.eval-e-sndI* $*$ **by** *meson*
next
case (*eval-e-concatI i v1 bv1 v2 bv2*)
then obtain *v1'* **and** *v2'* **where** $*$:*e* = *CE-concat v1' v2'* $\wedge$ *v1* = *v1'*$[y::=v]_{cev}$ $\wedge$ *v2* = *v2'*$[y::=v]_{cev}$
using *assms* **by**(*nominal-induct e rule*:*ce.strong-induct*,*simp*+)
hence *eval-e i* $(v1'[y::=v]_{cev})$ (*SBitvec bv1*) $\wedge$ *eval-e i* $(v2'[y::=v]_{cev})$ (*SBitvec bv2*) **using** *eval-e-concatI*
by *simp*
moreover hence *eval-e* $(i\ (\ y \mapsto s\ ))$ *v1'* (*SBitvec bv1*) $\wedge$ *eval-e* $(i\ (\ y \mapsto s\ ))$ *v2'* (*SBitvec bv2*)
using *subst-v-eval-v eval-e-concatI* $*$ **by** *blast*
ultimately show *?case* **using** *RCLogic.eval-e-concatI* $*$ *eval-v-uniqueness* **by** (*metis eval-e-concatI.hyps*(*1*))
next
case (*eval-e-lenI i v1 bv*)
then obtain *v1'* **where** $*$:*e* = *CE-len v1'* $\wedge$ *v1* = *v1'*$[y::=v]_{cev}$
using *assms* **by**(*nominal-induct e rule*:*ce.strong-induct*,*simp*+)
hence *eval-e i* $(v1'[y::=v]_{cev})$ (*SBitvec bv*) **using** *eval-e-lenI* **by** *simp*
hence *eval-e* $(i\ (\ y \mapsto s\ ))$ *v1'* (*SBitvec bv*) **using** *subst-v-eval-v eval-e-lenI* $*$ **by** *blast*
then show *?case* **using** *RCLogic.eval-e-lenI* $*$ **by** *meson*
qed

lemma *subst-c-eval-v*[*simp*]:
fixes *v*::*v* **and** *c* :: *c*
assumes *i* $[\![\ v\ ]\!]\sim$ *s* **and** *i* $[\![\ c[x::=v]_{cv}\ ]\!]\sim$ *s1* **and**
$(i\ (\ x \mapsto s))$ $[\![\ c\ ]\!]\sim$ *s2*
shows *s1* = *s2*
using *assms* **proof**(*nominal-induct c arbitrary*: *s1 s2 rule*: *c.strong-induct*)
case *C-true*
hence *s1* = *True* $\wedge$ *s2* = *True* **using** *eval-c-elims subst-cv.simps* **by** *auto*
then show *?case* **by** *auto*
next
case *C-false*
hence *s1* = *False* $\wedge$ *s2* = *False* **using** *eval-c-elims subst-cv.simps* **by** *metis*
then show *?case* **by** *auto*
next
case (*C-conj c1 c2*)
hence $*$:*eval-c i* $(c1[x::=v]_{cv}$ *AND* $c2[x::=v]_{cv})$ *s1* **using** *subst-cv.simps* **by** *auto*
then obtain *s11* **and** *s12* **where** (*s1* = (*s11* $\wedge$ *s12*)) $\wedge$ *eval-c i* $c1[x::=v]_{cv}$ *s11* $\wedge$ *eval-c i* $c2[x::=v]_{cv}$
s12 **using**
eval-c-elims(*3*) **by** *metis*
moreover obtain *s21* **and** *s22* **where** *eval-c* $(i\ (\ x \mapsto s))$ *c1 s21* $\wedge$ *eval-c* $(i\ (\ x \mapsto s))$ *c2 s22* $\wedge$
(*s2* = (*s21* $\wedge$ *s22*)) **using**
eval-c-elims(*3*) *C-conj* **by** *metis*
ultimately show *?case* **using** *C-conj* **by** (*meson eval-c-elims*)
next

case (*C-disj c1 c2*)
hence ∗:*eval-c i* (*c1*[*x*::=*v*]$_{cv}$ *OR c2*[*x*::=*v*]$_{cv}$) *s1* **using** *subst-cv.simps* **by** *auto*
then obtain *s11* **and** *s12* **where** (*s1* = (*s11* ∨ *s12*)) ∧ *eval-c i c1*[*x*::=*v*]$_{cv}$ *s11* ∧ *eval-c i c2*[*x*::=*v*]$_{cv}$ *s12* **using**
eval-c-elims(*4*) **by** *metis*
moreover obtain *s21* **and** *s22* **where** *eval-c* (*i* (*x* ↦ *s*)) *c1 s21* ∧ *eval-c* (*i* (*x* ↦ *s*)) *c2 s22* ∧ (*s2* = (*s21* ∨ *s22*)) **using**
eval-c-elims(*4*) *C-disj* **by** *metis*
ultimately show *?case* **using** *C-disj* **by** (*meson eval-c-elims*)
next
case (*C-not c1*)
then obtain *s11* **where** (*s1* = (¬ *s11*)) ∧ *eval-c i c1*[*x*::=*v*]$_{cv}$ *s11* **using**
eval-c-elims(*6*) **by** (*metis subst-cv.simps*(*7*))
moreover obtain *s21* **where** *eval-c* (*i* (*x* ↦ *s*)) *c1 s21* ∧ (*s2* = (¬*s21*)) **using**
eval-c-elims(*6*) *C-not* **by** *metis*
ultimately show *?case* **using** *C-not* **by** (*meson eval-c-elims*)
next
case (*C-imp c1 c2*)
hence ∗:*eval-c i* (*c1*[*x*::=*v*]$_{cv}$ *IMP c2*[*x*::=*v*]$_{cv}$) *s1* **using** *subst-cv.simps* **by** *auto*
then obtain *s11* **and** *s12* **where** (*s1* = (*s11* ⟶ *s12*)) ∧ *eval-c i c1*[*x*::=*v*]$_{cv}$ *s11* ∧ *eval-c i c2*[*x*::=*v*]$_{cv}$ *s12* **using**
eval-c-elims(*5*) **by** *metis*
moreover obtain *s21* **and** *s22* **where** *eval-c* (*i* (*x* ↦ *s*)) *c1 s21* ∧ *eval-c* (*i* (*x* ↦ *s*)) *c2 s22* ∧ (*s2* = (*s21* ⟶ *s22*)) **using**
eval-c-elims(*5*) *C-imp* **by** *metis*
ultimately show *?case* **using** *C-imp* **by** (*meson eval-c-elims*)
next
case (*C-eq e1 e2*)
hence ∗:*eval-c i* (*e1*[*x*::=*v*]$_{cev}$ == *e2*[*x*::=*v*]$_{cev}$) *s1* **using** *subst-cv.simps* **by** *auto*
then obtain *s11* **and** *s12* **where** (*s1* = (*s11* = *s12*)) ∧ *eval-e i* (*e1*[*x*::=*v*]$_{cev}$) *s11* ∧ *eval-e i* (*e2*[*x*::=*v*]$_{cev}$) *s12* **using**
eval-c-elims(*7*) **by** *metis*
moreover obtain *s21* **and** *s22* **where** *eval-e* (*i* (*x* ↦ *s*)) *e1 s21* ∧ *eval-e* (*i* (*x* ↦ *s*)) *e2 s22* ∧ (*s2* = (*s21* = *s22*)) **using**
eval-c-elims(*7*) *C-eq* **by** *metis*
ultimately show *?case* **using** *C-eq subst-e-eval-v* **by** (*metis eval-e-uniqueness*)
qed

lemma *wfI-upd*:
assumes *wfI* Θ Γ *i* **and** *wfRCV* Θ *s b* **and** *wfG* Θ *B* ((*x*, *b*, *c*) #$_{\Gamma}$ Γ)
shows *wfI* Θ ((*x*, *b*, *c*) #$_{\Gamma}$ Γ) (*i*(*x* ↦ *s*))
proof(*subst wfI-def*,*rule*)
fix *xa*
assume *as*:*xa* ∈ *setG* ((*x*, *b*, *c*) #$_{\Gamma}$ Γ)

then obtain *x1*::*x* **and** *b1*::*b* **and** *c1*::*c* **where** *xa*: *xa* = (*x1*,*b1*,*c1*) **using** *setG.simps*
using *prod-cases3* **by** *blast*

have ∃ *sa*. *Some sa* = (*i*(*x* ↦ *s*)) *x1* ∧ *wfRCV* Θ *sa b1* **proof**(*cases x=x1*)
case *True*
hence *b=b1* **using** *as xa wfG-unique assms* **by** *metis*

hence *Some s = (i(x ↦ s)) x1 ∧ wfRCV Θ s b1* **using** *assms True* **by** *simp*
then show *?thesis* **by** *auto*
next
case *False*
hence *(x1,b1,c1) ∈ setG Γ* **using** *xa as* **by** *auto*
then obtain *sa* **where** *Some sa = i x1 ∧ wfRCV Θ sa b1* **using** *assms wfI-def as xa* **by** *auto*
hence *Some sa = (i(x ↦ s)) x1 ∧ wfRCV Θ sa b1* **using** *False* **by** *auto*
then show *?thesis* **by** *auto*
qed

thus *case xa of (xa, ba, ca) ⇒ ∃ sa. Some sa = (i(x ↦ s)) xa ∧ wfRCV Θ sa ba* **using** *xa* **by** *auto*
qed

lemma *wfI-upd-full*:
fixes *v::v*
assumes *wfI Θ G i* **and** *G = ((Γ'[x::=v]$_{\Gamma v}$)@Γ)* **and** *wfRCV Θ s b* **and** *wfG Θ B (Γ'@((x,b,c)#$_\Gamma$Γ))* **and** *Θ ; B ; Γ ⊢$_{wf}$ v : b*
shows *wfI Θ (Γ'@((x,b,c)#$_\Gamma$Γ)) (i(x ↦ s))*
proof(*subst wfI-def*,*rule*)
fix *xa*
assume *as:xa ∈ setG (Γ'@((x,b,c)#$_\Gamma$Γ))*

then obtain *x1::x* **and** *b1::b* **and** *c1::c* **where** *xa: xa = (x1,b1,c1)* **using** *setG.simps*
using *prod-cases3* **by** *blast*

have *∃ sa. Some sa = (i(x ↦ s)) x1 ∧ wfRCV Θ sa b1*
proof(*cases x=x1*)
case *True*
hence *b=b1* **using** *as xa wfG-unique-full assms* **by** *metis*
hence *Some s = (i(x ↦ s)) x1 ∧ wfRCV Θ s b1* **using** *assms True* **by** *simp*
then show *?thesis* **by** *auto*
next
case *False*
hence *(x1,b1,c1) ∈ setG (Γ'@Γ)* **using** *as xa* **by** *auto*
then obtain *c1'* **where** *(x1,b1,c1') ∈ setG (Γ'[x::=v]$_{\Gamma v}$@Γ)* **using** *xa as wfG-member-subst assms False* **by** *metis*
then obtain *sa* **where** *Some sa = i x1 ∧ wfRCV Θ sa b1* **using** *assms wfI-def as xa* **by** *blast*
hence *Some sa = (i(x ↦ s)) x1 ∧ wfRCV Θ sa b1* **using** *False* **by** *auto*
then show *?thesis* **by** *auto*
qed

thus *case xa of (xa, ba, ca) ⇒ ∃ sa. Some sa = (i(x ↦ s)) xa ∧ wfRCV Θ sa ba* **using** *xa* **by** *auto*
qed

lemma *subst-c-satis*[*simp*]:
fixes *v::v*
assumes *i ⟦ v ⟧ ~ s* **and** *wfC Θ B ((x,b,c')#$_\Gamma$Γ) c* **and** *wfI Θ Γ i* **and** *Θ ; B ; Γ ⊢$_{wf}$ v : b*
shows *i ⊨ (c[x::=v]$_{cv}$) ⟷ (i (x ↦ s)) ⊨ c*
proof −
have *wfI Θ ((x, b, c') #$_\Gamma$ Γ) (i(x ↦ s))* **using** *wfI-upd assms wfC-wf eval-v-base* **by** *blast*
then obtain *s1* **where** *s1:eval-c (i(x ↦ s)) c s1* **using** *eval-c-exist*[*of Θ ((x,b,c')#$_\Gamma$Γ) (i (x ↦ s)) B c*] *assms* **by** *auto*

have Θ ; B ; Γ $\vdash_{wf}$ $c[x::=v]_{cv}$ **using** *wf-subst1(2)[OF assms(2) - assms(4) , of GNil x] subst-gv.simps* **by** *simp*
then obtain *s2* **where** *s2:eval-c i* $c[x::=v]_{cv}$ *s2* **using** *eval-c-exist[of* Θ Γ *i B* $c[x::=v]_{cv}$ *] assms* **by** *auto*

show *?thesis* **using** *s1 s2 subst-c-eval-v[OF assms(1) s2 s1] is-satis.cases*
using *eval-c-uniqueness is-satis.simps* **by** *auto*
qed

Key theorem telling us we can replace a substitution with an update to the valuation

lemma *subst-c-satis-full*:
fixes *v::v* **and** $\Gamma'::\Gamma$
assumes *i* $[\![v]\!]^{\sim}$ *s* **and** *wfC* Θ *B* $(\Gamma'@((x,b,c')\#_\Gamma\Gamma))$ *c* **and** *wfI* Θ $((\Gamma'[x::=v]_{\Gamma v})@\Gamma)$ *i* **and** Θ ; *B* ; $\Gamma \vdash_{wf} v : b$
shows $i \models (c[x::=v]_{cv}) \longleftrightarrow (i\ (\ x \mapsto s)) \models c$
proof −
have *wfI* Θ $(\Gamma'@((x,\ b,\ c'))\ \#_\Gamma\ \Gamma)$ $(i(x \mapsto s))$ **using** *wfI-upd-full assms wfC-wf eval-v-base wfI-suffix wfI-def wfV-wf* **by** *fast*
then obtain *s1* **where** *s1:eval-c* $(i(x \mapsto s))$ *c s1* **using** *eval-c-exist[of* Θ $(\Gamma'@(x,b,c')\#_\Gamma\Gamma)$ $(i\ (\ x \mapsto s))$ *B c] assms* **by** *auto*

have Θ ; *B* ; $((\Gamma'[x::=v]_{\Gamma v})@\Gamma) \vdash_{wf} c[x::=v]_{cv}$ **using** *wbc-subst assms* **by** *auto*

then obtain *s2* **where** *s2:eval-c i* $c[x::=v]_{cv}$ *s2* **using** *eval-c-exist[of* Θ $((\Gamma'[x::=v]_{\Gamma v})@\Gamma)$ *i B* $c[x::=v]_{cv}$ *] assms* **by** *auto*

show *?thesis* **using** *s1 s2 subst-c-eval-v[OF assms(1) s2 s1] is-satis.cases*
using *eval-c-uniqueness is-satis.simps* **by** *auto*
qed

11.5 Validity

lemma *validI[intro]*:
fixes *c::c*
assumes *wfC P B G c* **and** $\forall i.\ P\ ;\ G \vdash i \wedge i \models G \longrightarrow i \models c$
shows $P\ ;\ B\ ;\ G \models c$
using *assms valid.simps* **by** *presburger*

lemma *valid-g-wf*:
fixes *c::c* **and** *G::*Γ
assumes $P\ ;\ B\ ;\ G \models c$
shows $P\ ;\ B \vdash_{wf} G$
using *assms wfC-wf valid.simps* **by** *blast*

lemma *valid-reflI* [*intro*]:
fixes *b::b*
assumes $P\ ;\ B\ ;\ ((x,b,c1)\#_\Gamma G) \vdash_{wf} c1$ **and** $c1 = c2$
shows $P\ ;\ B\ ;\ ((x,b,c1)\#_\Gamma G) \models c2$
using *satis-reflI assms* **by** *simp*

11.5.1 Weakening and Strengthening

Adding to the domain of a valuation doesn't change the result

lemma *eval-v-weakening*:
fixes *c*::*v* **and** *B*::*bv fset*
assumes $i = i' |`` d$ **and** *supp c* $\subseteq$ *atom ‘ d* $\cup$ *supp B* **and** $i \llbracket c \rrbracket \sim s$
shows $i' \llbracket c \rrbracket \sim s$
using *assms* **proof**(*nominal-induct c arbitrary*:*s rule*: *v.strong-induct*)
case (*V-lit x*)
then show *?case* **using** *eval-v-elims eval-v-litI* **by** *metis*
next
case (*V-var x*)
have *atom x* $\in$ *atom ‘ d* **using** *x-not-in-b-set*[*of x B*] *assms v.supp*(*2*) *supp-at-base*
proof −
show *?thesis*
by (*metis UnE V-var.prems*(*2*) ‹*atom x* $\notin$ *supp B*› *singletonI subset-iff supp-at-base v.supp*(*2*))
qed
moreover have *Some s* = *i x* **using** *assms eval-v-elims*(*2*)
using *V-var.prems*(*3*) **by** *blast*
hence *Some s*= i' *x* **using** *assms insert-subset restrict-in*
proof −
show *?thesis*
by (*metis* (*no-types*) ‹$i = i' |`` d$› ‹*Some s* = *i x*› *atom-eq-iff calculation imageE restrict-in*)
qed
thus *?case* **using** *eval-v.eval-v-varI* **by** *simp*

next
case (*V-pair v1 v2*)
then show *?case* **using** *eval-v-elims*(*3*) *eval-v-pairI v.supp*
by (*metis assms le-sup-iff*)
next
case (*V-cons dc v1*)
then show *?case* **using** *eval-v-elims*(*4*) *eval-v-consI v.supp*
by (*metis assms le-sup-iff*)
next
case (*V-consp tyid dc b1 v1*)

then obtain *sv1* **where** ∗:$i \llbracket v1 \rrbracket \sim sv1 \wedge s = SConsp\ tyid\ dc\ b1\ sv1$ **using** *eval-v-elims* **by** *metis*
hence $i' \llbracket v1 \rrbracket \sim sv1$ **using** *V-consp* **by** *auto*
then show *?case* **using** ∗ *eval-v-conspI v.supp eval-v.simps assms le-sup-iff* **by** *metis*
qed

lemma *eval-v-restrict*:
fixes *c*::*v* **and** *B*::*bv fset*
assumes $i = i' |`` d$ **and** *supp c* $\subseteq$ *atom ‘ d* $\cup$ *supp B* **and** $i' \llbracket c \rrbracket \sim s$
shows $i \llbracket c \rrbracket \sim s$
using *assms* **proof**(*nominal-induct c arbitrary*:*s rule*: *v.strong-induct*)
case (*V-lit x*)
then show *?case* **using** *eval-v-elims eval-v-litI* **by** *metis*
next
case (*V-var x*)

```
  have atom x ∈ atom ‘ d using x-not-in-b-set[of x B] assms v.supp(2) supp-at-base
  proof −
    show ?thesis
      by (metis UnE V-var.prems(2) ‹atom x ∉ supp B› singletonI subset-iff supp-at-base v.supp(2))
  qed
  moreover have Some s = i′ x using assms eval-v-elims(2)
    using V-var.prems(3) by blast
  hence Some s= i x using assms insert-subset restrict-in
  proof −
    show ?thesis
      by (metis (no-types) ‹i = i′ |‘ d› ‹Some s = i′ x› atom-eq-iff calculation imageE restrict-in)
  qed
  thus ?case using eval-v.eval-v-varI by simp
next
  case (V-pair v1 v2)
  then show ?case using eval-v-elims(3) eval-v-pairI v.supp
    by (metis assms le-sup-iff)
next
  case (V-cons dc v1)
  then show ?case using eval-v-elims(4) eval-v-consI v.supp
    by (metis assms le-sup-iff)
next
  case (V-consp tyid dc b1 v1)

  then obtain sv1 where ∗:i′ ⟦ v1 ⟧ ~ sv1 ∧ s = SConsp tyid dc b1 sv1 using eval-v-elims by metis
  hence i ⟦ v1 ⟧ ~ sv1 using V-consp by auto
  then show ?case using ∗ eval-v-conspI v.supp eval-v.simps  assms le-sup-iff by metis
qed

lemma eval-e-weakening:
  fixes e::ce and B::bv fset
  assumes  i ⟦ e ⟧ ~ s and i = i′ |‘ d and supp e ⊆ atom ‘ d ∪ supp B
  shows i′ ⟦ e ⟧ ~ s
using assms proof(induct rule: eval-e.induct)
  case (eval-e-valI i v sv)
  then show ?case using ce.supp eval-e.intros
    using eval-v-weakening by auto
next
  case (eval-e-plusI i v1 n1 v2 n2)
  then show ?case using ce.supp eval-e.intros
    using eval-v-weakening by auto
next
  case (eval-e-leqI i v1 n1 v2 n2)
    then show ?case using ce.supp eval-e.intros
    using eval-v-weakening by auto
next
  case (eval-e-fstI i v v1 v2)
  then show ?case using ce.supp eval-e.intros
    using eval-v-weakening by metis
next
  case (eval-e-sndI i v v1 v2)
  then show ?case using ce.supp eval-e.intros
```

using *eval-v-weakening* **by** *metis*
next
case (*eval-e-concatI i v1 bv2 v2 bv1*)
then show *?case* **using** *ce.supp eval-e.intros*
using *eval-v-weakening* **by** *auto*
next
case (*eval-e-lenI i v bv*)
then show *?case* **using** *ce.supp eval-e.intros*
using *eval-v-weakening* **by** *auto*
qed

lemma *eval-e-restrict* :
fixes *e*::*ce* **and** *B*::*bv fset*
assumes $i' [\![e]\!] \sim s$ **and** $i = i' |` d$ **and** $supp\ e \subseteq atom\ `\ d \cup supp\ B$
shows $i [\![e]\!] \sim s$
using *assms* **proof**(*induct rule*: *eval-e.induct*)
case (*eval-e-valI i v sv*)
then show *?case* **using** *ce.supp eval-e.intros*
using *eval-v-restrict* **by** *auto*
next
case (*eval-e-plusI i v1 n1 v2 n2*)
then show *?case* **using** *ce.supp eval-e.intros*
using *eval-v-restrict* **by** *auto*
next
case (*eval-e-leqI i v1 n1 v2 n2*)
then show *?case* **using** *ce.supp eval-e.intros*
using *eval-v-restrict* **by** *auto*
next
case (*eval-e-fstI i v v1 v2*)
then show *?case* **using** *ce.supp eval-e.intros*
using *eval-v-restrict* **by** *metis*
next
case (*eval-e-sndI i v v1 v2*)
then show *?case* **using** *ce.supp eval-e.intros*
using *eval-v-restrict* **by** *metis*
next
case (*eval-e-concatI i v1 bv2 v2 bv1*)
then show *?case* **using** *ce.supp eval-e.intros*
using *eval-v-restrict* **by** *auto*
next
case (*eval-e-lenI i v bv*)
then show *?case* **using** *ce.supp eval-e.intros*
using *eval-v-restrict* **by** *auto*
qed

lemma *eval-c-i-weakening*:
fixes *c*::*c* **and** *B*::*bv fset*
assumes $i [\![c]\!] \sim s$ **and** $i = i' |` d$ **and** $supp\ c \subseteq atom\ `\ d \cup supp\ B$
shows $i' [\![c]\!] \sim s$
using *assms* **proof**(*induct rule*:*eval-c.induct*)
case (*eval-c-eqI i e1 sv1 e2 sv2*)
then show *?case* **using** *eval-c.intros eval-e-weakening* **by** *auto*

qed(*auto simp add*: *eval-c.intros*)+

lemma *eval-c-i-restrict*:
fixes *c*::*c* **and** *B*::*bv fset*
assumes $i' \llbracket c \rrbracket \sim s$ **and** $i = i' \mid` d$ **and** $supp\ c \subseteq atom\ ` d \cup supp\ B$
shows $i \llbracket c \rrbracket \sim s$
using *assms* **proof**(*induct rule*:*eval-c.induct*)
case (*eval-c-eqI i e1 sv1 e2 sv2*)
then show *?case* **using** *eval-c.intros eval-e-restrict* **by** *auto*
qed(*auto simp add*: *eval-c.intros*)+

lemma *is-satis-i-weakening*:
fixes *c*::*c* **and** *B*::*bv fset*
assumes $i = i' \mid` d$ **and** $supp\ c \subseteq atom\ ` d \cup supp\ B$ **and** $i \models c$
shows $i' \models c$
using *is-satis.simps eval-c-i-weakening*[*OF - assms*(*1*) *assms*(*2*)]
using *assms*(*3*) **by** *auto*

lemma *is-satis-i-restrict*:
fixes *c*::*c* **and** *B*::*bv fset*
assumes $i = i' \mid` d$ **and** $supp\ c \subseteq atom\ ` d \cup supp\ B$ **and** $i' \models c$
shows $i \models c$
using *is-satis.simps eval-c-i-restrict*[*OF - assms*(*1*) *assms*(*2*)]
using *assms*(*3*) **by** *auto*

lemma *is-satis-g-restrict1*:
fixes Γ'::Γ **and** Γ::Γ
assumes $setG\ \Gamma \subseteq setG\ \Gamma'$ **and** $i \models \Gamma'$
shows $i \models \Gamma$
using *assms* **proof**(*induct* Γ *rule*: Γ.*induct*)
case *GNil*
then show *?case* **by** *auto*
next
case (*GCons xbc G*)
obtain *x* **and** *b* **and** *c*::*c* **where** *xbc*: *xbc*=(*x*,*b*,*c*)
using *prod-cases3* **by** *blast*
hence $i \models G$ **using** *GCons* **by** *auto*
moreover have $i \models c$ **using** *GCons*
is-satis-iff setG.simps subset-iff
using *xbc* **by** *blast*
ultimately show *?case* **using** *is-satis-g.simps GCons*
by (*simp add*: *xbc*)
qed

lemma *is-satis-g-restrict2*:
fixes Γ'::Γ **and** Γ::Γ
assumes $i \models \Gamma$ **and** $i' = i \mid` d$ **and** *atom-dom* $\Gamma \subseteq atom\ ` d$ **and** $\Theta\ ;\ B \vdash_{wf} \Gamma$
shows $i' \models \Gamma$
using *assms* **proof**(*induct* Γ *rule*: Γ-*induct*)
case *GNil*
then show *?case* **by** *auto*
next

case (*GCons x b c G*)

hence $i' \models G$ **proof** −
have $i \models G$ **using** *GCons* **by** *simp*
moreover have *atom-dom G* ⊆ *atom ‘ d* **using** *GCons* **by** *simp*
ultimately show *?thesis* **using** *GCons wfG-cons2* **by** *blast*
qed

moreover have $i' \models c$ **proof** −
have $i \models c$ **using** *GCons* **by** *auto*
moreover have Θ ; *B* ; (*x*, *b*, *TRUE*) $\#_\Gamma$ *G* $\vdash_{wf}$ *c* **using** *wfG-wfC GCons* **by** *simp*
moreover hence *supp c* ⊆ *atom ‘ d* ∪ *supp B* **using** *wfC-supp GCons atom-dom-eq* **by** *blast*
ultimately show *?thesis* **using** *is-satis-i-restrict*[*of i' i d c*] *GCons* **by** *simp*
qed

ultimately show *?case* **by** *auto*
qed

lemma *is-satis-g-restrict*:
fixes Γ′::Γ **and** Γ::Γ
assumes *setG* Γ ⊆ *setG* Γ′ **and** $i' \models \Gamma'$ **and** $i = i' \mid `(fst ` setG\ \Gamma)$ **and** Θ ; *B* $\vdash_{wf}$ Γ
shows $i \models \Gamma$
using *assms is-satis-g-restrict1*[*OF assms*(*1*) *assms*(*2*)] *is-satis-g-restrict2*[*OF - assms*(*3*)] **by** *simp*

11.5.2 Updating valuation

lemma *is-satis-c-i-upd*:
fixes *c*::*c*
assumes *atom x* ♯ *c* **and** $i \models c$
shows $((i\ (\ x \mapsto s))) \models c$
using *assms eval-c-i-upd is-satis.simps* **by** *simp*

lemma *is-satis-g-i-upd*:
fixes *G*::Γ
assumes *atom x* ♯ *G* **and** $i \models G$
shows $((i\ (\ x \mapsto s))) \models G$
using *assms* **proof**(*induct G rule*: Γ-*induct*)
case *GNil*
then show *?case* **by** *auto*
next
case (*GCons x′ b′ c′ G′*)

hence ∗:*atom x* ♯ *G′* ∧ *atom x* ♯ *c′*
using *fresh-def fresh-GCons GCons* **by** *force*
moreover hence *is-satis* $((i\ (\ x \mapsto s)))\ c'$
using *is-satis-c-i-upd GCons is-satis-g.simps* **by** *auto*
moreover have *is-satis-g* $(i(x \mapsto s))\ G'$ **using** *GCons* ∗ **by** *fastforce*
ultimately show *?case*
using *GCons is-satis-g.simps*(*2*) **by** *metis*
qed

lemma *valid-weakening*:
assumes Θ ; *B* ; Γ $\models$ *c* **and** *setG* Γ ⊆ *setG* Γ′ **and** *wfG* Θ *B* Γ′

```
  shows  Θ ; B ; Γ' ⊨ c
proof −
  have wfC Θ B  Γ c using assms valid.simps by auto
  hence sp: supp c ⊆ atom ‘(fst ‘setG Γ) ∪ supp B using wfX-wfY wfG-elims
    using atom-dom.simps wf-supp(2) by metis
  have wfg: wfG Θ B  Γ using assms valid.simps wfC-wf by auto

  moreover have a1: (∀ i. wfI Θ Γ' i ∧ i ⊨ Γ' ⟶ i ⊨ c) proof(rule allI, rule impI)
    fix i
    assume as: wfI Θ Γ' i ∧ i ⊨ Γ'
    hence as1: fst ‘ setG Γ ⊆ dom i using assms wfI-domi by blast
    obtain i' where idash: i' =  restrict-map i (fst ‘setG Γ) by blast
    hence as2: dom i' = (fst ‘setG Γ)  using dom-restrict as1 by auto

    have id2: Θ ; Γ ⊢ i' ∧ i' ⊨ Γ proof −
      have wfI Θ  Γ i' using as2 wfI-restrict-weakening[of Θ Γ' i i' Γ] as  assms
        using idash by blast
      moreover have i' ⊨ Γ using is-satis-g-restrict[OF assms(2)] wfg as idash by auto
      ultimately show ?thesis using idash by auto
    qed
    hence i' ⊨ c using assms valid.simps by auto
    thus  i ⊨ c using assms valid.simps is-satis-i-weakening  idash sp by blast
  qed
  moreover have wfC Θ B Γ' c using wf-weakening assms valid.simps
    by (meson  wfg)
  ultimately show  ?thesis using assms valid.simps by auto
qed

lemma is-satis-g-suffix:
  fixes G::Γ
  assumes i ⊨ (G'@G)
  shows i ⊨ G
  using assms proof(induct G' rule:Γ.induct)
  case GNil
  then show ?case by auto
next
  case (GCons xbc x2)
  obtain x and b and c::c where xbc: xbc=(x,b,c)
     using prod-cases3 by blast
  hence  i ⊨ (xbc #Γ (x2 @ G)) using append-g.simps GCons by fastforce
  then show ?case using is-satis-g.simps GCons xbc by blast
qed

lemma wfG-inside-valid2:
  fixes x::x and Γ::Γ and c0::c and c0'::c
  assumes wfG Θ B  (Γ'@((x,b0,c0')#ΓΓ)) and
       Θ ; B ; Γ'@(x,b0,c0)#ΓΓ ⊨ c0'
  shows wfG Θ B  (Γ'@((x,b0,c0)#ΓΓ))
proof −
  have wfG Θ  B  (Γ'@(x,b0,c0)#ΓΓ) using valid.simps wfC-wf assms by auto
  thus ?thesis using wfG-replace-inside-full assms by auto
qed
```

lemma *valid-trans*:
assumes Θ ; $\mathcal{B}$; $\Gamma \models c0[z::=v]_v$ **and** Θ ; $\mathcal{B}$; $(z,b,c0)\#_\Gamma\Gamma \models c1$ **and** *atom* $z \sharp \Gamma$ **and** *wfV* Θ $\mathcal{B}$ Γ *v b*
shows Θ ; $\mathcal{B}$; $\Gamma \models c1[z::=v]_v$
proof −
have ∗:*wfC* Θ $\mathcal{B}$ $((z,b,c0)\#_\Gamma\Gamma)$ *c1* **using** *valid.simps assms* **by** *auto*
hence *wfC* Θ $\mathcal{B}$ Γ $(c1[z::=v]_v)$ **using** *wf-subst1*(*2*)[*OF* ∗ , *of GNil*] *assms subst-gv.simps subst-v-c-def* **by** *force*

moreover have $\forall i.$ *wfI* Θ Γ $\quad i \wedge$ *is-satis-g i* $\Gamma \longrightarrow$ *is-satis i* $(c1[z::=v]_v)$
proof(*rule*,*rule*)
fix *i*
assume *as*: *wfI* Θ Γ $\quad i \wedge$ *is-satis-g i* Γ
then obtain *sv* **where** *sv*: *eval-v i v sv* $\wedge$ *wfRCV* Θ *sv b* **using** *eval-v-exist assms* **by** *metis*
hence *is-satis i* $(c0[z::=v]_v)$ **using** *assms valid.simps as* **by** *metis*
hence *is-satis* $(i(z \mapsto sv))$ *c0* **using** *subst-c-satis sv as assms valid.simps wfC-wf wfG-elim2 subst-v-c-def* **by** *metis*
moreover have *is-satis-g* $(i(z \mapsto sv))$ Γ
using *is-satis-g-i-upd assms* **by** (*simp add*: *as*)
ultimately have *is-satis-g* $(i(z \mapsto sv))$ $((z,b,c0)\#_\Gamma\Gamma)$
using *is-satis-g.simps* **by** *simp*
moreover have *wfI* Θ $((z,b,c0)\#_\Gamma\Gamma)$ $(i(z \mapsto sv))$ **using** *as wfI-upd sv assms valid.simps wfC-wf* **by** *metis*
ultimately have *is-satis* $(i(z \mapsto sv))$ *c1* **using** *assms valid.simps* **by** *auto*
thus *is-satis i* $(c1[z::=v]_v)$ **using** *subst-c-satis sv as assms valid.simps wfC-wf wfG-elim2 subst-v-c-def* **by** *metis*
qed

ultimately show *?thesis* **using** *valid.simps* **by** *auto*
qed

lemma *valid-trans-2*:
assumes Θ ; $\mathcal{B}$; $((x,\ b,\ c1[y::=V\text{-}var\ x]_v)\ \#_\Gamma\ \Gamma) \models c2[y::=V\text{-}var\ x]_v$ **and**
Θ ; $\mathcal{B}$; $((x,\ b,\ c2[y::=V\text{-}var\ x]_v)\ \#_\Gamma\ \Gamma) \models c3[y::=V\text{-}var\ x]_v$
shows Θ ; $\mathcal{B}$; $((x,\ b,\ c1[y::=V\text{-}var\ x]_v)\ \#_\Gamma\ \Gamma) \models c3[y::=V\text{-}var\ x]_v$
unfolding *valid.simps* **proof**
show Θ ; $\mathcal{B}$; $(x,\ b,\ c1[y::=V\text{-}var\ x]_v)\ \#_\Gamma\ \Gamma \quad \vdash_{wf} c3[y::=V\text{-}var\ x]_v$ **using** *wf-trans valid.simps assms* **by** *metis*

show $\forall i.$ (*wfI* Θ $((x,\ b,\ c1[y::=V\text{-}var\ x]_v)\ \#_\Gamma\ \Gamma)$ $i \wedge$ (*is-satis-g i* $((x,\ b,\ c1[y::=V\text{-}var\ x]_v)\ \#_\Gamma\ \Gamma)) \longrightarrow$ (*is-satis i* $(c3[y::=V\text{-}var\ x]_v)$))
proof(*rule*,*rule*)
fix *i*
assume *as*: Θ ; $(x,\ b,\ c1[y::=V\text{-}var\ x]_v)\ \#_\Gamma\ \Gamma \vdash i \wedge\ \ i \models (x,\ b,\ c1[y::=V\text{-}var\ x]_v)\ \#_\Gamma\ \Gamma$
have $i \models c2[y::=V\text{-}var\ x]_v$ **using** *is-satis-g.simps as assms* **by** *simp*
moreover have $i \models \Gamma$ **using** *is-satis-g.simps as* **by** *simp*
ultimately show $i \models c3[y::=V\text{-}var\ x]_v$ **using** *assms is-satis-g.simps valid.simps*
by (*metis append-g.simps*(*1*) *as wfI-replace-inside*)
qed
qed

lemma *eval-v-weakening-x*:
fixes *c*::*v*
assumes $i' \llbracket c \rrbracket \sim s$ **and** *atom* $x \sharp c$ **and** $i = i'(x \mapsto s')$
shows $i \llbracket c \rrbracket \sim s$
using *assms* **proof**(*induct rule*: *eval-v.induct*)
case (*eval-v-litI i l*)
then show *?case* **using** *eval-v.intros* **by** *auto*
next
case (*eval-v-varI sv i1 x1*)
hence $x \neq x1$ **using** *v.fresh fresh-at-base* **by** *auto*
hence *i x1* = *Some sv* **using** *eval-v-varI* **by** *simp*
then show *?case* **using** *eval-v.intros* **by** *auto*
next
case (*eval-v-pairI i v1 s1 v2 s2*)
then show *?case* **using** *eval-v.intros* **by** *auto*
next
case (*eval-v-consI i v s tyid dc*)
then show *?case* **using** *eval-v.intros* **by** *auto*
next
case (*eval-v-conspI i v s tyid dc b*)
then show *?case* **using** *eval-v.intros* **by** *auto*
qed

lemma *eval-e-weakening-x*:
fixes *c*::*ce*
assumes $i' \llbracket c \rrbracket \sim s$ **and** *atom* $x \sharp c$ **and** $i = i'(x \mapsto s')$
shows $i \llbracket c \rrbracket \sim s$
using *assms* **proof**(*induct rule*: *eval-e.induct*)
case (*eval-e-valI i v sv*)
then show *?case* **using** *eval-v-weakening-x eval-e.intros ce.fresh* **by** *metis*
next
case (*eval-e-plusI i v1 n1 v2 n2*)
then show *?case* **using** *eval-v-weakening-x eval-e.intros ce.fresh* **by** *metis*
next
case (*eval-e-leqI i v1 n1 v2 n2*)
then show *?case* **using** *eval-v-weakening-x eval-e.intros ce.fresh* **by** *metis*
next
case (*eval-e-fstI i v v1 v2*)
then show *?case* **using** *eval-v-weakening-x eval-e.intros ce.fresh* **by** *metis*
next
case (*eval-e-sndI i v v1 v2*)
then show *?case* **using** *eval-v-weakening-x eval-e.intros ce.fresh* **by** *metis*
next
case (*eval-e-concatI i v1 bv1 v2 bv2*)
then show *?case* **using** *eval-v-weakening-x eval-e.intros ce.fresh* **by** *metis*
next
case (*eval-e-lenI i v bv*)
then show *?case* **using** *eval-v-weakening-x eval-e.intros ce.fresh* **by** *metis*
qed

lemma *eval-c-weakening-x*:

fixes c::c
assumes $i' \llbracket c \rrbracket \sim s$ **and** *atom* $x \sharp c$ **and** $i = i'(x \mapsto s')$
shows $i \llbracket c \rrbracket \sim s$
using *assms* **proof**(*induct rule*: *eval-c.induct*)
case (*eval-c-trueI i*)
then show *?case* **using** *eval-c.intros* **by** *auto*
next
case (*eval-c-falseI i*)
then show *?case* **using** *eval-c.intros* **by** *auto*
next
case (*eval-c-conjI i c1 b1 c2 b2*)
then show *?case* **using** *eval-c.intros* **by** *auto*
next
case (*eval-c-disjI i c1 b1 c2 b2*)
then show *?case* **using** *eval-c.intros* **by** *auto*
next
case (*eval-c-impI i c1 b1 c2 b2*)
then show *?case* **using** *eval-c.intros* **by** *auto*
next
case (*eval-c-notI i c b*)
then show *?case* **using** *eval-c.intros* **by** *auto*
next
case (*eval-c-eqI i e1 sv1 e2 sv2*)
then show *?case* **using** *eval-e-weakening-x c.fresh eval-c.intros* **by** *metis*
qed

lemma *is-satis-weakening-x*:
fixes c::c
assumes $i' \models c$ **and** *atom* $x \sharp c$ **and** $i = i'(x \mapsto s)$
shows $i \models c$
using *eval-c-weakening-x assms is-satis.simps* **by** *simp*

lemma *is-satis-g-weakening-x*:
fixes G::Γ
assumes $i' \models G$ **and** *atom* $x \sharp G$ **and** $i = i'(x \mapsto s)$
shows $i \models G$
using *assms* **proof**(*induct G rule*: Γ-*induct*)
case *GNil*
then show *?case* **by** *auto*
next
case (*GCons x' b' c'* Γ')
hence *atom* $x \sharp c'$ **using** *fresh-GCons fresh-prodN* **by** *simp*
moreover hence $i \models c'$ **using** *is-satis-weakening-x is-satis-g.simps*(*2*) *GCons* **by** *metis*
then show *?case* **using** *is-satis-g.simps*(*2*)[*of i x' b' c'* Γ'] *GCons fresh-GCons* **by** *simp*
qed

11.6 Base Type Substitution

The idea of boxing is to take an smt val and its base type and at nodes in the smt val that correspond to type variables we wrap them in an SUt smt val node. Another way of looking at

it is that s' where the node for the base type variable is an 'any node'. It is needed to prove subst_b_valid - the base-type variable substitution lemma for validity.

The first *rcl-val* is the expanded form (has type with base-variables replaced with base-type terms) ; the second is its corresponding form

We only have one variable so we need to ensure that in all of the *bs-boxed-BVarI* cases, the s has the same base type.

For example is an SMT value is (SPair (SInt 1) (SBool true)) and it has sort (BPair (BVar x) BBool)[x::=BInt] then the boxed version is SPair (SUt (SInt 1)) (SBool true) and is has sort (BPair (BVar x) BBool). We need to do this so that we can obtain from a valuation i, that gives values like the first smt value, to a valuation i' that gives values like the second.

inductive *boxed-b* :: $\Theta \Rightarrow$ *rcl-val* $\Rightarrow$ *b* $\Rightarrow$ *bv* $\Rightarrow$ *b* $\Rightarrow$ *rcl-val* $\Rightarrow$ *bool* (- $\vdash$ - $\sim$ - [- ::= -] \ - [50,50] 50) **where**

boxed-b-BVar1I: $[\![bv = bv' ; \ wfRCV\ P\ s\ b]\!] \Longrightarrow$ *boxed-b P s* (*B-var bv'*) *bv b* (*SUt s*)

| *boxed-b-BVar2I*: $[\![bv \neq bv'; wfRCV\ P\ s\ (B\text{-}var\ bv')]\!] \Longrightarrow$ *boxed-b P s* (*B-var bv'*) *bv b s*

| *boxed-b-BIntI*:*wfRCV P s B-int* $\Longrightarrow$ *boxed-b P s B-int - - s*

| *boxed-b-BBoolI*:*wfRCV P s B-bool* $\Longrightarrow$ *boxed-b P s B-bool - - s*

| *boxed-b-BUnitI*:*wfRCV P s B-unit* $\Longrightarrow$ *boxed-b P s B-unit - - s*

| *boxed-b-BPairI*:$[\![$ *boxed-b P s1 b1 bv b s1'* ; *boxed-b P s2 b2 bv b s2'* $]\!] \Longrightarrow$ *boxed-b P* (*SPair s1 s2*) (*B-pair b1 b2*) *bv b* (*SPair s1' s2'*)

| *boxed-b-BConsI*:$[\![$
AF-typedef tyid dclist $\in$ *set P*;
(*dc*, $\{\!|$ *x* : *b* | *c* $|\!\}$) $\in$ *set dclist* ;
boxed-b P s1 b bv b' s1'
$]\!] \Longrightarrow$
boxed-b P (*SCons tyid dc s1*) (*B-id tyid*) *bv b'* (*SCons tyid dc s1'*)

| *boxed-b-BConspI*:$[\![$ *AF-typedef-poly tyid bva dclist* $\in$ *set P*;
atom bva $\sharp$ (*b1*,*bv*,*b'*,*s1*,*s1'*);
(*dc*, $\{\!|$ *x* : *b* | *c* $|\!\}$) $\in$ *set dclist* ;
boxed-b P s1 (*b*$[$*bva*::=*b1*$]_{bb}$) *bv b' s1'*
$]\!] \Longrightarrow$
boxed-b P (*SConsp tyid dc b1*$[$*bv*::=*b'*$]_{bb}$ *s1*) (*B-app tyid b1*) *bv b'* (*SConsp tyid dc b1 s1'*)

| *boxed-b-Bbitvec*: *wfRCV P s B-bitvec* $\Longrightarrow$ *boxed-b P s B-bitvec bv b s*

equivariance *boxed-b*
nominal-inductive *boxed-b* **.**

inductive-cases *boxed-b-elims*:
boxed-b P s (*B-var bv*) *bv' b s'*
boxed-b P s B-int bv b s'
boxed-b P s B-bool bv b s'
boxed-b P s B-unit bv b s'
boxed-b P s (*B-pair b1 b2*) *bv b s'*
boxed-b P s (*B-id dc*) *bv b s'*
boxed-b P s B-bitvec bv b s'
boxed-b P s (*B-app dc b'*) *bv b s'*

lemma *boxed-b-wfRCV*:
assumes *boxed-b P s b bv b′ s′* **and** $\vdash_{wf} P$
shows *wfRCV P s b*$[bv::=b']_{bb} \wedge$ *wfRCV P s′ b*
using *assms* **proof**(*induct rule*: *boxed-b.inducts*)
case (*boxed-b-BVar1I bv bv′ P s b*)
then show *?case* **using** *wfRCV.intros* **by** *auto*
next
case (*boxed-b-BVar2I bv bv′ P s*)
then show *?case* **using** *wfRCV.intros* **by** *auto*
next
case (*boxed-b-BPairI P s1 b1 bv b s1′ s2 b2 s2′*)
then show *?case* **using** *wfRCV.intros rcl-val.supp* **by** *simp*
next
case (*boxed-b-BConsI tyid dclist P dc x b c s1 bv b′ s1′*)
hence *supp b* = {} **using** *wfTh-supp-b* **by** *metis*
hence *b* [*bv* ::= *b′*]$_{bb}$ = *b* **using** *fresh-def subst-b-b-def forget-subst*[*of bv b b′*] **by** *auto*
hence *P* ⊢ *SCons tyid dc s1* : (*B-id tyid*) **using** *wfRCV.intros rcl-val.supp subst-bb.simps boxed-b-BConsI*
by *metis*
moreover have *P* ⊢ *SCons tyid dc s1′* : *B-id tyid* **using** *boxed-b-BConsI*
using *wfRCV.intros rcl-val.supp subst-bb.simps boxed-b-BConsI* **by** *metis*
ultimately show *?case* **using** *subst-bb.simps* **by** *metis*
next
case (*boxed-b-BConspI tyid bva dclist P b1 bv b′ s1 s1′ dc x b c*)

obtain *bva2* **and** *dclist2* **where** ∗:*AF-typedef-poly tyid bva dclist* = *AF-typedef-poly tyid bva2 dclist2*
∧
atom bva2 ♯ (*bv*,(*P*, *SConsp tyid dc b1*$[bv::=b']_{bb}$ *s1*, *B-app tyid b1*$[bv::=b']_{bb}$))
using *obtain-fresh-bv* **by** *metis*

then obtain *x2* **and** *b2* **and** *c2* **where** ∗∗:‹(*dc*, {| *x2* : *b2* | *c2* |}) ∈ *set dclist2*›
using *boxed-b-BConspI td-lookup-eq-iff type-def.eq-iff* **by** *metis*

have *P* ⊢ *SConsp tyid dc b1*$[bv::=b']_{bb}$ *s1* : (*B-app tyid b1*$[bv::=b']_{bb}$) **proof**
show *1*: ‹*AF-typedef-poly tyid bva2 dclist2* ∈ *set P*› **using** *boxed-b-BConspI* ∗ **by** *auto*
show *2*: ‹(*dc*, {| *x2* : *b2* | *c2* |}) ∈ *set dclist2*› **using** *boxed-b-BConspI* **using** ∗∗ **by** *simp*

hence *atom bv* ♯ *b2* **proof** −
have *supp b2* ⊆ { *atom bva2* } **using** *wfTh-poly-supp-b 1 2 boxed-b-BConspI* **by** *auto*
moreover have *bv* ≠ *bva2* **using** ∗ *fresh-Pair fresh-at-base* **by** *metis*
ultimately show *?thesis* **using** *fresh-def* **by** *force*
qed
moreover have *b*[*bva*::=*b1*]$_{bb}$ = *b2*[*bva2*::=*b1*]$_{bb}$ **using** *wfTh-typedef-poly-b-eq-iff* ∗ *2 boxed-b-BConspI*
by *metis*
ultimately show ‹ *P* ⊢ *s1* : *b2*[*bva2*::=*b1*[*bv*::=*b′*]$_{bb}$]$_{bb}$› **using** *boxed-b-BConspI subst-b-b-def*
subst-bb-commute **by** *auto*
show *atom bva2* ♯ (*P*, *SConsp tyid dc b1*$[bv::=b']_{bb}$ *s1*, *B-app tyid b1*$[bv::=b']_{bb}$) **using** ∗ *fresh-Pair*
by *metis*
qed

moreover have *P* ⊢ *SConsp tyid dc b1 s1′* : *B-app tyid b1* **proof**
show ‹*AF-typedef-poly tyid bva dclist* ∈ *set P*› **using** *boxed-b-BConspI* **by** *auto*
show ‹(*dc*, {| *x* : *b* | *c* |}) ∈ *set dclist*› **using** *boxed-b-BConspI* **by** *auto*

show ‹ $P \vdash s1' : b[bva::=b1]_{bb}$ › **using** *boxed-b-BConspI* **by** *auto*
have *atom bva* ♯ P **using** *boxed-b-BConspI wfTh-fresh* **by** *metis*
thus *atom bva* ♯ $(P, SConsp\ tyid\ dc\ b1\ s1', B\text{-}app\ tyid\ b1)$ **using** *boxed-b-BConspI rcl-val.fresh b.fresh pure-fresh fresh-prodN* **by** *metis*
qed

ultimately show *?case* **using** *subst-bb.simps* **by** *simp*
qed(*auto*)+

lemma *subst-b-var*:
assumes $B\text{-}var\ bv2 = b[bv::=b']_{bb}$
shows $(b = B\text{-}var\ bv \wedge b' = B\text{-}var\ bv2) \vee (b = B\text{-}var\ bv2 \wedge bv \neq bv2)$
using *assms* **by**(*nominal-induct b rule*: *b.strong-induct*,*auto*+)

Here the valuation i' is the conv wrap version of i. For every x in G, i' x is the conv wrap version of i x

inductive *boxed-i* :: $\Theta \Rightarrow \Gamma \Rightarrow b \Rightarrow bv \Rightarrow valuation \Rightarrow valuation \Rightarrow bool$ (- ; - ; - , - ⊢ - ≈ - [50,50] 50) **where**
boxed-i-GNilI: $\Theta\ ;\ GNil\ ;\ b\ ,\ bv\ \vdash i \approx i$
| *boxed-i-GConsI*: $[\![\ Some\ s = i\ x;\ \ boxed\text{-}b\ \Theta\ s\ b\ bv\ b'\ s'\ ;\ \ \Theta\ ;\ \Gamma\ ;\ b'\ ,\ bv \vdash i \approx i'\ ]\!] \Longrightarrow \Theta\ ;\ ((x,b,c)\#_\Gamma\Gamma)\ ;\ b'\ ,\ bv \vdash i \approx (i'(x \mapsto s'))$
equivariance *boxed-i*
nominal-inductive *boxed-i* .

inductive-cases *boxed-i-elims*:
$\Theta\ ; GNil\ ;\ b\ ,\ bv \vdash i \approx i'$
$\Theta\ ;\ ((x,b,c)\#_\Gamma\Gamma)\ ;\ b'\ ,\ bv \vdash i \approx i'$

lemma *wfRCV-poly-elims*:
fixes $tm::'a::fs$ **and** $b::b$
assumes $T \vdash SConsp\ typid\ dc\ bdc\ s : b$
obtains *bva dclist x1 b1 c1* **where** $b = B\text{-}app\ typid\ bdc \wedge$
$AF\text{-}typedef\text{-}poly\ typid\ bva\ dclist \in set\ T \wedge (dc, \{\!|\ x1 : b1\ |\ c1\ |\!\}) \in set\ dclist \wedge\ T \vdash s : b1[bva::=bdc]_{bb} \wedge atom\ bva \sharp tm$
using *assms* **proof**(*nominal-induct SConsp typid dc bdc s b avoiding*: *tm rule*:*wfRCV.strong-induct*)
case (*wfRCV-BConsPI bv dclist* Θ *x b c*)
then show *?case* **by** *simp*
qed

lemma *boxed-b-ex*:
assumes *wfRCV T s* $b[bv::=b']_{bb}$ **and** *wfTh T*
shows $\exists s'.\ boxed\text{-}b\ T\ s\ b\ bv\ b'\ s'$
using *assms* **proof**(*nominal-induct s arbitrary*: *b rule*: *rcl-val.strong-induct*)
case (*SBitvec x*)
have $*:b[bv::=b']_{bb} = B\text{-}bitvec$ **using** *wfRCV-elims*(*9*)[*OF SBitvec*(*1*)] **by** *metis*
show *?case* **proof** (*cases b* = *B-var bv*)
case *True*

moreover have $T \vdash$ *SBitvec x* : *B-bitvec* **using** *wfRCV.intros* **by** *simp*
moreover hence $b' =$ *B-bitvec* **using** *True SBitvec subst-bb.simps* $*$ **by** *simp*
ultimately show *?thesis* **using** *boxed-b.intros wfRCV.intros* **by** *metis*
next
case *False*
hence $b =$ *B-bitvec* **using** *subst-bb-inject* $*$ **by** *metis*
then show *?thesis* **using** $*$ *SBitvec boxed-b.intros* **by** *metis*
qed
next
case (*SNum x*)
have $*{:}b[bv::=b']_{bb} =$ *B-int* **using** *wfRCV-elims*(*10*)[*OF SNum*(*1*)] **by** *metis*
show *?case* **proof** (*cases b* = *B-var bv*)
case *True*
moreover have $T \vdash$ *SNum x* : *B-int* **using** *wfRCV.intros* **by** *simp*
moreover hence $b' =$ *B-int* **using** *True SNum subst-bb.simps*(*1*) $*$ **by** *simp*
ultimately show *?thesis* **using** *boxed-b-BVar1I wfRCV.intros* **by** *metis*
next
case *False*
hence $b =$ *B-int* **using** *subst-bb-inject*(*1*) $*$ **by** *metis*
then show *?thesis* **using** $*$ *SNum boxed-b-BIntI* **by** *metis*
qed
next
case (*SBool x*)
have $*{:}b[bv::=b']_{bb} =$ *B-bool* **using** *wfRCV-elims*(*11*)[*OF SBool*(*1*)] **by** *metis*
show *?case* **proof** (*cases b* = *B-var bv*)
case *True*
moreover have $T \vdash$ *SBool x* : *B-bool* **using** *wfRCV.intros* **by** *simp*
moreover hence $b' =$ *B-bool* **using** *True SBool subst-bb.simps* $*$ **by** *simp*
ultimately show *?thesis* **using** *boxed-b.intros wfRCV.intros* **by** *metis*
next
case *False*
hence $b =$ *B-bool* **using** *subst-bb-inject* $*$ **by** *metis*
then show *?thesis* **using** $*$ *SBool boxed-b.intros* **by** *metis*
qed
next
case (*SPair s1 s2*)
then obtain *b1* **and** *b2* **where** $*{:}b[bv::=b']_{bb} =$ *B-pair b1 b2* $\wedge$ *wfRCV T s1 b1* $\wedge$ *wfRCV T s2 b2*
using *wfRCV-elims*(*12*) **by** *metis*
show *?case* **proof** (*cases b* = *B-var bv*)
case *True*
moreover have $T \vdash$ *SPair s1 s2* : *B-pair b1 b2* **using** *wfRCV.intros* $*$ **by** *simp*
moreover hence $b' =$ *B-pair b1 b2* **using** *True SPair subst-bb.simps*(*1*) $*$ **by** *simp*
ultimately show *?thesis* **using** *boxed-b-BVar1I* **by** *metis*
next
case *False*
then obtain *b1'* **and** *b2'* **where** $b =$ *B-pair b1' b2'* $\wedge$ *b1*=*b1'*$[bv::=b']_{bb}$ $\wedge$ *b2*=*b2'*$[bv::=b']_{bb}$ **using**
subst-bb-inject(*5*)[*OF - False*] $*$ **by** *metis*
then show *?thesis* **using** $*$ *SPair boxed-b-BPairI* **by** *blast*
qed
next
case (*SCons tyid dc s1*)
have $*{:}b[bv::=b']_{bb} =$ *B-id tyid* **using** *wfRCV-elims*(*13*)[*OF SCons*(*2*)] **by** *metis*

show *?case* **proof** (*cases b = B-var bv*)
case *True*
moreover have $T \vdash$ *SCons tyid dc s1* : *B-id tyid* **using** *wfRCV.intros*
using *local.∗ SCons.prems* **by** *auto*
moreover hence $b' =$ *B-id tyid* **using** *True SCons subst-bb.simps(1) ∗* **by** *simp*
ultimately show *?thesis* **using** *boxed-b-BVar1I wfRCV.intros* **by** *metis*
next
case *False*
then obtain *b1'* **where** *beq*: *b = B-id tyid* **using** *subst-bb-inject ∗* **by** *metis*
then obtain *b2 dclist x c* **where** ∗∗:*AF-typedef tyid dclist* ∈ *set T* ∧ (*dc*, {| *x* : *b2* | *c* |}) ∈ *set dclist* ∧ *wfRCV T s1 b2* **using** *wfRCV-elims(13) ∗ SCons* **by** *metis*
then have *atom bv* ♯ *b2* **using** ‹*wfTh T*› *wfTh-lookup-supp-empty*[*of tyid dclist T dc* {| *x* : *b2* | *c* |}] *τ.fresh fresh-def* **by** *auto*
then have *b2* = *b2*[*bv* ::= *b'*]$_{bb}$ **using** *forget-subst subst-b-b-def* **by** *metis*
then obtain *s1'* **where** *s1*:*T* ⊢ *s1* ~ *b2* [*bv* ::= *b'*] \ *s1'* **using** *SCons* ∗∗ **by** *metis*

have *T* ⊢ *SCons tyid dc s1* ~ (*B-id tyid*) [*bv* ::= *b'*] \ *SCons tyid dc s1'* **proof**(*rule boxed-b-BConsI*)
show *AF-typedef tyid dclist* ∈ *set T* **using** ∗∗ **by** *auto*
show (*dc*, {| *x* : *b2* | *c* |}) ∈ *set dclist* **using** ∗∗ **by** *auto*
show *T* ⊢ *s1* ~ *b2* [*bv* ::= *b'*] \ *s1'* **using** *s1* ∗∗ **by** *auto*

qed
thus *?thesis* **using** *beq* **by** *metis*
qed
next
case (*SConsp typid dc bdc s*)

obtain *bva dclist x1 b1 c1* **where** ∗∗:*b*[*bv*::=*b'*]$_{bb}$ = *B-app typid bdc* ∧
AF-typedef-poly typid bva dclist ∈ *set T* ∧ (*dc*, {| *x1* : *b1* | *c1* |}) ∈ *set dclist* ∧ *T* ⊢ *s* : *b1*[*bva*::=*bdc*]$_{bb}$ ∧ *atom bva* ♯ *bv*
using *wfRCV-poly-elims*[*OF SConsp(2)*] **by** *metis*

then have ∗:*B-app typid bdc* = *b*[*bv*::=*b'*]$_{bb}$ **using** *wfRCV-elims(14)*[*OF SConsp(2)*] **by** *metis*
show *?case* **proof** (*cases b = B-var bv*)
case *True*
moreover have *T* ⊢ *SConsp typid dc bdc s* : *B-app typid bdc* **using** *wfRCV.intros*
using *local.∗ SConsp.prems(1)* **by** *auto*
moreover hence *b'* = *B-app typid bdc* **using** *True SConsp subst-bb.simps ∗* **by** *simp*
ultimately show *?thesis* **using** *boxed-b.intros wfRCV.intros* **by** *metis*
next
case *False*
then obtain *bdc'* **where** *bdc*: *b = B-app typid bdc'* ∧ *bdc* = *bdc'*[*bv*::=*b'*]$_{bb}$ **using** ∗ *subst-bb-inject(8)*[*OF* ∗] **by** *metis*

have *atom bv* ♯ *b1* **proof** −
have *supp b1* ⊆ { *atom bva* } **using** *wfTh-poly-supp-b* ∗∗ *SConsp* **by** *metis*
moreover have *bv* ≠ *bva* **using** ∗∗ **by** *auto*
ultimately show *?thesis* **using** *fresh-def* **by** *force*
qed
have *T* ⊢ *s* : *b1*[*bva*::=*bdc*]$_{bb}$ **using** ∗∗ **by** *auto*
moreover have *b1*[*bva*::=*bdc'*]$_{bb}$[*bv*::=*b'*]$_{bb}$ = *b1*[*bva*::=*bdc*]$_{bb}$ **using** *bdc subst-bb-commute* ‹*atom bv* ♯ *b1*› **by** *auto*

ultimately obtain s' **where** $s'{:}T \vdash s \sim b1[bva::=bdc']_{bb}\ [\ bv ::= b'\ ] \setminus s'$
using *SConsp(1)[of b1*$[bva::=bdc']_{bb}$*] bdc SConsp* **by** *metis*
have $T \vdash$ *SConsp typid dc bdc'*$[bv::=b']_{bb}$ $s \sim$ (*B-app typid bdc'*) $[\ bv ::= b'\ ] \setminus$ *SConsp typid dc bdc' s'*
proof −

obtain *bva3* **and** *dclist3* **where** *3:AF-typedef-poly typid bva3 dclist3 = AF-typedef-poly typid bva dclist* ∧
atom bva3 ♯ (*bdc', bv, b', s, s'*) **using** *obtain-fresh-bv* **by** *metis*
then obtain *x3 b3 c3* **where** *4:*(*dc*, ⦃ *x3* : *b3* | *c3* ⦄) ∈ *set dclist3*
using *boxed-b-BConspI td-lookup-eq-iff type-def.eq-iff*
by (*metis* ∗∗)

show *?thesis* **proof**
show ‹*AF-typedef-poly typid bva3 dclist3* ∈ *set T*› **using** *3* ∗∗ **by** *metis*
show ‹*atom bva3* ♯ (*bdc', bv, b', s, s'*)› **using** *3* **by** *metis*
show *4:*‹(*dc*, ⦃ *x3* : *b3* | *c3* ⦄) ∈ *set dclist3*› **using** *4* **by** *auto*
have *b3*$[bva3::=bdc']_{bb}$ = *b1*$[bva::=bdc']_{bb}$ **proof**(*rule wfTh-typedef-poly-b-eq-iff*)
show ‹*AF-typedef-poly typid bva3 dclist3* ∈ *set T*› **using** *3* ∗∗ **by** *metis*
show ‹(*dc*, ⦃ *x3* : *b3* | *c3* ⦄) ∈ *set dclist3*› **using** *4* **by** *auto*
show ‹*AF-typedef-poly typid bva dclist* ∈ *set T*› **using** ∗∗ **by** *auto*
show ‹(*dc*, ⦃ *x1* : *b1* | *c1* ⦄) ∈ *set dclist*› **using** ∗∗ **by** *auto*
qed(*simp add*: ∗∗ *SConsp*)
thus ‹ $T \vdash s \sim b3[bva3::=bdc']_{bb}\ [\ bv ::= b'\ ] \setminus s'$ › **using** *s'* **by** *auto*
qed
qed
then show *?thesis* **using** *bdc* **by** *auto*

qed
next
case *SUnit*
have ∗:*b*$[bv::=b']_{bb}$ = *B-unit* **using** *wfRCV-elims SUnit* **by** *metis*
show *?case* **proof** (*cases b* = *B-var bv*)
case *True*
moreover have $T \vdash$ *SUnit* : *B-unit* **using** *wfRCV.intros* **by** *simp*
moreover hence *b'* = *B-unit* **using** *True SUnit subst-bb.simps* ∗ **by** *simp*
ultimately show *?thesis* **using** *boxed-b.intros wfRCV.intros* **by** *metis*
next
case *False*
hence *b* = *B-unit* **using** *subst-bb-inject* ∗ **by** *metis*
then show *?thesis* **using** ∗ *SUnit boxed-b.intros* **by** *metis*
qed
next
case (*SUt x*)
then obtain *bv'* **where** ∗:*b*$[bv::=b']_{bb}$= *B-var bv'* **using** *wfRCV-elims* **by** *metis*
show *?case* **proof** (*cases b* = *B-var bv*)
case *True*
then show *?thesis* **using** *boxed-b-BVar1I*
using *local.*∗ *wfRCV-BVarI* **by** *fastforce*
next
case *False*
then show *?thesis* **using** *boxed-b-BVar1I boxed-b-BVar2I*

```
    using local.* wfRCV-BVarI    by (metis subst-b-var)
  qed
qed

lemma boxed-i-ex:
  assumes wfI T Γ[bv::=b]_Γb i and wfTh T
  shows ∃i'. T ; Γ ; b , bv ⊢ i ≈ i'
using assms proof(induct Γ arbitrary: i rule:Γ-induct)
  case GNil
  then show ?case using boxed-i-GNilI by metis
next
  case (GCons x' b' c' Γ')
  then obtain s where 1:Some s = i x' ∧ wfRCV T s b'[bv::=b]_bb using wfI-def subst-gb.simps by
auto
  then obtain s' where 2: boxed-b T s b' bv b s' using boxed-b-ex GCons by metis
  then obtain i' where 3: boxed-i T Γ' b bv i  i' using GCons wfI-def subst-gb.simps by force
  have boxed-i T ((x', b', c') #_Γ Γ') b bv i  (i'(x' ↦ s')) proof
    show Some s = i x' using 1 by auto
    show boxed-b T s b' bv b s' using 2 by auto
    show T  ; Γ' ; b , bv ⊢ i ≈ i'  using  3 by auto
  qed
  thus ?case by auto
qed


lemma  boxed-b-eq:
  assumes boxed-b Θ s1 b bv b' s1' and ⊢_wf Θ
  shows wfTh Θ ⟹ boxed-b Θ s2 b bv b' s2' ⟹ ( s1 = s2 ) = ( s1' = s2' )
using assms proof(induct arbitrary: s2 s2'  rule: boxed-b.inducts )
  case (boxed-b-BVar1I bv bv'  P s b )
  then show ?case
    using boxed-b-elims(1) rcl-val.eq-iff by metis
next
  case (boxed-b-BVar2I bv bv' P s b)
  then show ?case using boxed-b-elims(1) by metis
next
  case (boxed-b-BIntI P s uu uv)
  hence s2 = s2' using boxed-b-elims by metis
  then show ?case by auto
next
  case (boxed-b-BBoolI P s uw ux)
  hence s2 = s2' using boxed-b-elims by metis
  then show ?case by auto
next
  case (boxed-b-BUnitI P s uy uz)
  hence s2 = s2' using boxed-b-elims by metis
  then show ?case by auto
next
  case (boxed-b-BPairI P s1 b1 bv b s1' s2a b2 s2a')
  then show ?case
    by (metis boxed-b-elims(5) rcl-val.eq-iff(4))
next
```

case (*boxed-b-BConsI tyid dclist P dc x b c s1 bv b′ s1′*)
obtain *s22* **and** *s22′ dclist2 dc2 x2 b2 c2* **where** ∗:*s2 = SCons tyid dc2 s22* ∧ *s2′ = SCons tyid dc2 s22′* ∧ *boxed-b P s22 b2 bv b′ s22′*
∧ *AF-typedef tyid dclist2* ∈ *set P* ∧ (*dc2*, {| *x2* : *b2* | *c2* |}) ∈ *set dclist2* **using** *boxed-b-elims*(*6*)[*OF boxed-b-BConsI*(*6*)] **by** *metis*
show *?case* **proof**(*cases dc = dc2*)
case *True*
hence *b = b2* **using** *wfTh-ctor-unique τ.eq-iff wfTh-dclist-unique wf boxed-b-BConsI* ∗ **by** *metis*
then show *?thesis* **using** *boxed-b-BConsI True* ∗ **by** *auto*
next
case *False*
then show *?thesis* **using** ∗ *boxed-b-BConsI* **by** *simp*
qed
next
case (*boxed-b-Bbitvec P s bv b*)
hence *s2 = s2′* **using** *boxed-b-elims* **by** *metis*
then show *?case* **by** *auto*
next
case (*boxed-b-BConspI tyid bva dclist P b1 bv b′ s1 s1′ dc x b c*)
thm *boxed-b-elims*(*8*)[*OF boxed-b-BConspI*(*7*)]
obtain *bva2 s22 s22′ dclist2 dc2 x2 b2 c2* **where** ∗:
s2 = SConsp tyid dc2 b1[*bv*::=*b′*]$_{bb}$ *s22* ∧
s2′ = SConsp tyid dc2 b1 s22′ ∧
boxed-b P s22 b2[*bva2*::=*b1*]$_{bb}$ *bv b′ s22′* ∧
AF-typedef-poly tyid bva2 dclist2 ∈ *set P* ∧ (*dc2*, {| *x2* : *b2* | *c2* |}) ∈ *set dclist2* **using** *boxed-b-elims*(*8*)[*OF boxed-b-BConspI*(*7*)] **by** *metis*
show *?case* **proof**(*cases dc = dc2*)
case *True*
hence *AF-typedef-poly tyid bva2 dclist2* ∈ *set P* ∧ (*dc*, {| *x2* : *b2* | *c2* |}) ∈ *set dclist2* **using** ∗ **by** *auto*
hence *b*[*bva*::=*b1*]$_{bb}$ = *b2*[*bva2*::=*b1*]$_{bb}$ **using** *wfTh-typedef-poly-b-eq-iff*[*OF boxed-b-BConspI*(*1*) *boxed-b-BConspI*(*3*)] ∗ *boxed-b-BConspI* **by** *metis*
then show *?thesis* **using** *boxed-b-BConspI True* ∗ **by** *auto*
next
case *False*
then show *?thesis* **using** ∗ *boxed-b-BConspI* **by** *simp*
qed
qed

lemma *bs-boxed-var*:
assumes *boxed-i* Θ Γ *b′ bv i i′*
shows *Some* (*b*,*c*) = *lookup* Γ *x* ⟹ *Some s = i x* ⟹ *Some s′ = i′ x* ⟹ *boxed-b* Θ *s b bv b′ s′*
using *assms* **proof**(*induct rule*: *boxed-i.inducts*)
case (*boxed-i-GNilI T i*)
then show *?case* **using** *lookup.simps* **by** *auto*
next
case (*boxed-i-GConsI s i x1* Θ *b1 bv b′ s′* Γ *i′ c*)
show *?case* **proof** (*cases x=x1*)
case *True*
then show *?thesis* **using** *boxed-i-GConsI*
fun-upd-same lookup.simps(*2*) *option.inject prod.inject* **by** *metis*

next
case *False*
then show *?thesis* **using** *boxed-i-GConsI*
fun-upd-same lookup.simps option.inject prod.inject **by** *auto*
qed
qed

lemma *eval-l-boxed-b*:
assumes $[\![l]\!] = s$
shows *boxed-b* Θ *s* (*base-for-lit l*) *bv b' s*
using *assms* **proof**(*nominal-induct l arbitrary*: *s rule*:*l.strong-induct*)
qed(*auto simp add*: *boxed-b.intros wfRCV.intros*)+

lemma *boxed-i-eval-v-boxed-b*:
fixes *v*::*v*
assumes *boxed-i* Θ Γ *b' bv i i'* **and** $i\ [\![v[bv::=b']_{vb}]\!] \sim s$ **and** $i'\ [\![v]\!] \sim s'$ **and** *wfV* Θ *B* Γ *v b* **and** *wfI* Θ Γ *i'*
shows *boxed-b* Θ *s b bv b' s'*
using *assms* **proof**(*nominal-induct v arbitrary*: *s s' b rule*:*v.strong-induct*)
case (*V-lit l*)
hence $[\![l]\!] = s \wedge [\![l]\!] = s'$ **using** *eval-v-elims* **by** *auto*
moreover have *b* = *base-for-lit l* **using** *wfV-elims*(*2*) *V-lit* **by** *metis*
ultimately show *?case* **using** *V-lit* **using** *eval-l-boxed-b subst-b-base-for-lit* **by** *metis*
next
case (*V-var x*)
hence *Some s* = *i x* $\wedge$ *Some s'* = *i' x* **using** *eval-v-elims subst-vb.simps* **by** *metis*
moreover obtain *c1* **where** *bc*:*Some* (*b*,*c1*) = *lookup* Γ *x* **using** *wfV-elims V-var* **by** *metis*
ultimately show *?case* **using** *bs-boxed-var V-var* **by** *metis*

next
case (*V-pair v1 v2*)
then obtain *b1* **and** *b2* **where** *b*:*b*=*B-pair b1 b2* **using** *wfV-elims subst-vb.simps* **by** *metis*
obtain *s1* **and** *s2* **where** *s*: *eval-v i* (*v1*[*bv*::=*b'*]$_{vb}$) *s1* $\wedge$ *eval-v i* (*v2*[*bv*::=*b'*]$_{vb}$) *s2* $\wedge$ *s* = *SPair s1 s2* **using** *eval-v-elims V-pair subst-vb.simps* **by** *metis*
obtain *s1'* **and** *s2'* **where** *s'*: *eval-v i' v1 s1'* $\wedge$ *eval-v i' v2 s2'* $\wedge$ *s'* = *SPair s1' s2'* **using** *eval-v-elims V-pair* **by** *metis*
thm *boxed-b-BPairI*
have *boxed-b* Θ (*SPair s1 s2*) (*B-pair b1 b2*) *bv b'* (*SPair s1' s2'*) **proof**(*rule boxed-b-BPairI*)
show *boxed-b* Θ *s1 b1 bv b' s1'* **using** *V-pair eval-v-elims wfV-elims b s s' b.eq-iff* **by** *metis*
show *boxed-b* Θ *s2 b2 bv b' s2'* **using** *V-pair eval-v-elims wfV-elims b s s' b.eq-iff* **by** *metis*
qed
then show *?case* **using** *s s' b* **by** *auto*
next
case (*V-cons tyid dc v1*)

obtain *dclist x b1 c* **where** $*$: *b* = *B-id tyid* $\wedge$ *AF-typedef tyid dclist* $\in$ *set* Θ $\wedge$ (*dc*, $\{\!|$ *x* : *b1* | *c* $|\!\}$) $\in$ *set dclist* $\wedge$ Θ ; *B* ; $\Gamma \vdash_{wf}$ *v1* : *b1*
using *wfV-elims*(*4*)[*OF V-cons*(*5*)] *V-cons* **by** *metis*
obtain *s2* **where** *s2*: *s* = *SCons tyid dc s2* $\wedge$ *i* $[\![$ (*v1*[*bv*::=*b'*]$_{vb}$) $]\!] \sim$ *s2* **using** *eval-v-elims V-cons subst-vb.simps* **by** *metis*
obtain *s2'* **where** *s2'*: *s'* = *SCons tyid dc s2'* $\wedge$ *i'* $[\![$ *v1* $]\!] \sim$ *s2'* **using** *eval-v-elims V-cons* **by** *metis*

have *sp*: *supp* {| *x* : *b1* | *c* |} = {} **using** *wfTh-lookup-supp-empty* ∗ *wfX-wfY* **by** *metis*

have *boxed-b* Θ (*SCons tyid dc s2*) (*B-id tyid*) *bv b′* (*SCons tyid dc s2′*)
proof(*rule boxed-b-BConsI*)
show *1*:*AF-typedef tyid dclist* ∈ *set* Θ **using** ∗ **by** *auto*
show *2*:(*dc*, {| *x* : *b1* | *c* |}) ∈ *set dclist* **using** ∗ **by** *auto*
have *bvf*:*atom bv* ♯ *b1* **using** *sp* *τ.fresh fresh-def* **by** *auto*
show Θ ⊢ *s2* ~ *b1* [*bv* ::= *b′*] \ *s2′* **using** *V-cons s2 s2′* ∗ **by** *metis*
qed
then show *?case* **using** ∗ *s2 s2′* **by** *simp*
next
case (*V-consp tyid dc b1 v1*)

obtain *bv2 dclist x2 b2 c2* **where** ∗: *b* = *B-app tyid b1* ∧ *AF-typedef-poly tyid bv2 dclist* ∈ *set* Θ ∧
(*dc*, {| *x2* : *b2* | *c2* |}) ∈ *set dclist* ∧ Θ ; *B* ; Γ ⊢$_{wf}$ *v1* : *b2*[*bv2*::=*b1*]$_{bb}$
using *wf-strong-elim*(*1*)[*OF V-consp* (*5*)] **by** *metis*

obtain *s2* **where** *s2*: *s* = *SConsp tyid dc b1*[*bv*::=*b′*]$_{bb}$ *s2* ∧ *i* ⟦ (*v1*[*bv*::=*b′*]$_{vb}$) ⟧ ~ *s2*
using *eval-v-elims V-consp subst-vb.simps* **by** *metis*

obtain *s2′* **where** *s2′*: *s′* = *SConsp tyid dc b1 s2′* ∧ *i′* ⟦ *v1* ⟧ ~ *s2′*
using *eval-v-elims V-consp* **by** *metis*

thm *obtain-fresh-bv-dclist-b-iff*

have ⊢$_{wf}$ Θ **using** *V-consp wfX-wfY* **by** *metis*
then obtain *bv3*::*bv* **and** *dclist3 x3 b3 c3* **where** ∗∗: *AF-typedef-poly tyid bv2 dclist* = *AF-typedef-poly tyid bv3 dclist3* ∧
(*dc*, {| *x3* : *b3* | *c3* |}) ∈ *set dclist3* ∧ *atom bv3* ♯ (*b1*, *bv*, *b′*, *s2*, *s2′*) ∧ *b2*[*bv2*::=*b1*]$_{bb}$ = *b3*[*bv3*::=*b1*]$_{bb}$
using ∗ *obtain-fresh-bv-dclist-b-iff*[**where** *tm*=(*b1*, *bv*, *b′*, *s2*, *s2′*)] **by** *metis*

have *boxed-b* Θ (*SConsp tyid dc b1*[*bv*::=*b′*]$_{bb}$ *s2*) (*B-app tyid b1*) *bv b′* (*SConsp tyid dc b1 s2′*)
proof(*rule boxed-b-BConspI*[*of tyid bv3 dclist3* Θ, **where** *x*=*x3* **and** *b*=*b3* **and** *c*=*c3*])
show *1*:*AF-typedef-poly tyid bv3 dclist3* ∈ *set* Θ **using** ∗ ∗∗ **by** *auto*
show *2*:(*dc*, {| *x3* : *b3* | *c3* |}) ∈ *set dclist3* **using** ∗∗ **by** *auto*
show *atom bv3* ♯ (*b1*, *bv*, *b′*, *s2*, *s2′*) **using** ∗∗ **by** *auto*
show Θ ⊢ *s2* ~ *b3*[*bv3*::=*b1*]$_{bb}$ [*bv* ::= *b′*] \ *s2′* **using** *V-consp s2 s2′* ∗ ∗∗ **by** *metis*
qed
then show *?case* **using** ∗ *s2 s2′* **by** *simp*

qed

lemma *boxed-i-eval-ce-boxed-b*:
fixes *e*::*ce*
assumes *i′* ⟦ *e* ⟧ ~ *s′* **and** *i* ⟦ *e*[*bv*::=*b′*]$_{ceb}$ ⟧ ~ *s* **and** *wfCE* Θ *B* Γ *e b* **and** *boxed-i* Θ Γ *b′ bv i i′* **and** *wfI* Θ Γ *i′*
shows *boxed-b* Θ *s b bv b′ s′*
using *assms* **proof**(*nominal-induct e arbitrary*: *s s′ b b′ rule*: *ce.strong-induct*)

case (*CE-val x*)
then show *?case* **using** *boxed-i-eval-v-boxed-b eval-e-elims wfCE-elims subst-ceb.simps* **by** *metis*
next
case (*CE-op opp v1 v2*)

have *1*:*wfCE* Θ *B* Γ *v1* (*B-int*) **using** *wfCE-elims CE-op* **by** *metis*
have *2*:*wfCE* Θ *B* Γ *v2* (*B-int*) **using** *wfCE-elims CE-op* **by** *metis*

consider (*Plus*) *opp* = *Plus* | (*LEq*) *opp* = *LEq* **using** *opp.exhaust* **by** *auto*
then show *?case* **proof**(*cases*)
case *Plus*
have ∗:*b* = *B-int* **using** *CE-op wfCE-elims Plus* **by** *metis*

obtain *n1* **and** *n2* **where** *n*:*s* = *SNum* (*n1* + *n2*) ∧ *i* ⟦ *v1*[*bv*::=*b'*]$_{ceb}$ ⟧ ~ *SNum n1* ∧ *i* ⟦ *v2*[*bv*::=*b'*]$_{ceb}$ ⟧ ~ *SNum n2* **using** *eval-e-elims CE-op subst-ceb.simps Plus* **by** *metis*
obtain *n1'* **and** *n2'* **where** *n'*:*s'* = *SNum* (*n1'* + *n2'*) ∧ *i'* ⟦ *v1* ⟧ ~ *SNum n1'* ∧ *i'* ⟦ *v2* ⟧ ~ *SNum n2'* **using** *eval-e-elims Plus CE-op* **by** *metis*

have *boxed-b* Θ (*SNum n1*) *B-int bv b'* (*SNum n1'*) **using** *boxed-i-eval-v-boxed-b 1 2 n n' CE-op* **by** *metis*
moreover have *boxed-b* Θ (*SNum n2*) *B-int bv b'* (*SNum n2'*) **using** *boxed-i-eval-v-boxed-b 1 2 n n' CE-op* **by** *metis*
ultimately have *s*=*s'* **using** *n' n boxed-b-elims*(*2*)
by (*metis rcl-val.eq-iff*(*2*))
thus *?thesis* **using** ∗ *n n' boxed-b-BIntI CE-op wfRCV.intros Plus* **by** *simp*
next
case *LEq*
hence ∗:*b* = *B-bool* **using** *CE-op wfCE-elims* **by** *metis*
obtain *n1* **and** *n2* **where** *n*:*s* = *SBool* (*n1* ≤ *n2*) ∧ *i* ⟦ *v1*[*bv*::=*b'*]$_{ceb}$ ⟧ ~ *SNum n1* ∧ *i* ⟦ *v2*[*bv*::=*b'*]$_{ceb}$ ⟧ ~ *SNum n2* **using** *eval-e-elims subst-ceb.simps CE-op LEq* **by** *metis*
obtain *n1'* **and** *n2'* **where** *n'*:*s'* = *SBool* (*n1'* ≤ *n2'*) ∧ *i'* ⟦ *v1* ⟧ ~ *SNum n1'* ∧ *i'* ⟦ *v2* ⟧ ~ *SNum n2'* **using** *eval-e-elims CE-op LEq* **by** *metis*

have *boxed-b* Θ (*SNum n1*) *B-int bv b'* (*SNum n1'*) **using** *boxed-i-eval-v-boxed-b 1 2 n n' CE-op* **by** *metis*
moreover have *boxed-b* Θ (*SNum n2*) *B-int bv b'* (*SNum n2'*) **using** *boxed-i-eval-v-boxed-b 1 2 n n' CE-op* **by** *metis*
ultimately have *s*=*s'* **using** *n' n boxed-b-elims*(*2*)
by (*metis rcl-val.eq-iff*(*2*))
thus *?thesis* **using** ∗ *n n' boxed-b-BBoolI CE-op wfRCV.intros LEq* **by** *simp*
qed

next
case (*CE-concat v1 v2*)

obtain *bv1* **and** *bv2* **where** *s* : *s* = *SBitvec* (*bv1* @ *bv2*) ∧ (*i* ⟦ *v1*[*bv*::=*b'*]$_{ceb}$ ⟧ ~ *SBitvec bv1*) ∧ *i* ⟦ *v2*[*bv*::=*b'*]$_{ceb}$ ⟧ ~ *SBitvec bv2*
using *eval-e-elims*(*6*) *subst-ceb.simps CE-concat.prems*(*2*) *eval-e-elims*(*6*) *subst-ceb.simps*(*6*) **by** *metis*
obtain *bv1'* **and** *bv2'* **where** *s'* : *s'* = *SBitvec* (*bv1'* @ *bv2'*) ∧ *i'* ⟦ *v1* ⟧ ~ *SBitvec bv1'* ∧ *i'* ⟦ *v2* ⟧ ~ *SBitvec bv2'*
using *eval-e-elims*(*6*) *CE-concat* **by** *metis*

then show *?case* **using** *boxed-i-eval-v-boxed-b wfCE-elims s s' CE-concat*
by (*metis CE-concat.prems*(*3*) *assms assms*(*5*) *wfRCV-BBitvecI boxed-b-Bbitvec boxed-b-elims*(*7*) *eval-e-concatI eval-e-uniqueness*)
next
case (*CE-fst ce*)
obtain *s2* **where** *1*:*i* ⟦ *ce*[*bv*::=*b'*]$_{ceb}$ ⟧ ~ *SPair s s2* **using** *CE-fst eval-e-elims subst-ceb.simps* **by** *metis*
obtain *s2'* **where** *2*:*i'* ⟦ *ce* ⟧ ~ *SPair s' s2'* **using** *CE-fst eval-e-elims* **by** *metis*
obtain *b2* **where** *3*:*wfCE* Θ *B* Γ *ce* (*B-pair b b2*) **using** *wfCE-elims*(*4*) *CE-fst* **by** *metis*

have *boxed-b* Θ (*SPair s s2*) (*B-pair b b2*) *bv b'* (*SPair s' s2'*)
using *1 2 3 CE-fst boxed-i-eval-v-boxed-b boxed-b-BPairI* **by** *auto*
thus *?case* **using** *boxed-b-elims*(*5*) **by** *force*
next
case (*CE-snd v*)
obtain *s1* **where** *1*:*i* ⟦ *v*[*bv*::=*b'*]$_{ceb}$ ⟧ ~ *SPair s1 s* **using** *CE-snd eval-e-elims subst-ceb.simps* **by** *metis*
obtain *s1'* **where** *2*:*i'* ⟦ *v* ⟧ ~ *SPair s1' s'* **using** *CE-snd eval-e-elims* **by** *metis*
obtain *b1* **where** *3*:*wfCE* Θ *B* Γ *v* (*B-pair b1 b*) **using** *wfCE-elims*(*5*) *CE-snd* **by** *metis*

have *boxed-b* Θ (*SPair s1 s*) (*B-pair b1 b*) *bv b'* (*SPair s1' s'*) **using** *1 2 3 CE-snd boxed-i-eval-v-boxed-b* **by** *simp*
thus *?case* **using** *boxed-b-elims*(*5*) **by** *force*
next
case (*CE-len v*)
obtain *s1* **where** *s*: *i* ⟦ *v*[*bv*::=*b'*]$_{ceb}$ ⟧ ~ *SBitvec s1* **using** *CE-len eval-e-elims subst-ceb.simps* **by** *metis*
obtain *s1'* **where** *s'*: *i'* ⟦ *v* ⟧ ~ *SBitvec s1'* **using** *CE-len eval-e-elims* **by** *metis*

have Θ ; *B* ; Γ ⊢$_{wf}$ *v* : *B-bitvec* ∧ *b* = *B-int* **using** *wfCE-elims CE-len* **by** *metis*
then show *?case* **using** *boxed-i-eval-v-boxed-b s s' CE-len*
by (*metis boxed-b-BIntI boxed-b-elims*(*7*) *eval-e-lenI eval-e-uniqueness subst-ceb.simps*(*5*) *wfI-wfCE-eval-e*)
qed

lemma *eval-c-eq-bs-boxed*:
fixes *c*::*c*
assumes *i* ⟦ *c*[*bv*::=*b*]$_{cb}$ ⟧ ~ *s* **and** *i'* ⟦ *c* ⟧ ~ *s'* **and** *wfC* Θ *B* Γ *c* **and** *wfI* Θ Γ *i'* **and** Θ ; Γ[*bv*::=*b*]$_{\Gamma b}$ ⊢ *i*
and *boxed-i* Θ Γ *b bv i i'*
shows *s* = *s'*
using *assms* **proof**(*nominal-induct c arbitrary*: *s s' rule*:*c.strong-induct*)
case *C-true*
then show *?case* **using** *eval-c-elims subst-cb.simps* **by** *metis*
next
case *C-false*
then show *?case* **using** *eval-c-elims subst-cb.simps* **by** *metis*
next
case (*C-conj c1 c2*)
obtain *s1* **and** *s2* **where** *1*: *eval-c i* (*c1*[*bv*::=*b*]$_{cb}$) *s1* ∧ *eval-c i* (*c2*[*bv*::=*b*]$_{cb}$) *s2* ∧ *s* = (*s1*∧*s2*)
using *C-conj eval-c-elims*(*3*) *subst-cb.simps*(*3*) **by** *metis*

obtain *s1'* **and** *s2'* **where** *2:eval-c i' c1 s1'* ∧ *eval-c i' c2 s2'* ∧ *s'* = (*s1'*∧*s2'*) **using** *C-conj eval-c-elims(3)* **by** *metis*
then show *?case* **using** *1 2 wfC-elims C-conj* **by** *metis*
next
case (*C-disj c1 c2*)

obtain *s1* **and** *s2* **where** *1*: *eval-c i* (*c1*[*bv*::=*b*]$_{cb}$) *s1* ∧ *eval-c i* (*c2*[*bv*::=*b*]$_{cb}$) *s2* ∧ *s* = (*s1*∨*s2*) **using** *C-disj eval-c-elims(4) subst-cb.simps(4)* **by** *metis*
obtain *s1'* **and** *s2'* **where** *2:eval-c i' c1 s1'* ∧ *eval-c i' c2 s2'* ∧ *s'* = (*s1'*∨*s2'*) **using** *C-disj eval-c-elims(4)* **by** *metis*
then show *?case* **using** *1 2 wfC-elims C-disj* **by** *metis*
next
case (*C-not c*)
obtain *s1*::*bool* **where** *1*: (*i* ⟦ *c*[*bv*::=*b*]$_{cb}$ ⟧ ~ *s1*) ∧ (*s* = (¬ *s1*)) **using** *C-not eval-c-elims(6) subst-cb.simps(7)* **by** *metis*
obtain *s1'*::*bool* **where** *2*: (*i'* ⟦ *c* ⟧ ~ *s1'*) ∧ (*s'* = (¬ *s1'*)) **using** *C-not eval-c-elims(6)* **by** *metis*
then show *?case* **using** *1 2 wfC-elims C-not* **by** *metis*
next
case (*C-imp c1 c2*)
obtain *s1* **and** *s2* **where** *1*: *eval-c i* (*c1*[*bv*::=*b*]$_{cb}$) *s1* ∧ *eval-c i* (*c2*[*bv*::=*b*]$_{cb}$) *s2* ∧ *s* = (*s1* ⟶ *s2*) **using** *C-imp eval-c-elims(5) subst-cb.simps(5)* **by** *metis*
obtain *s1'* **and** *s2'* **where** *2:eval-c i' c1 s1'* ∧ *eval-c i' c2 s2'* ∧ *s'* = (*s1'* ⟶ *s2'*) **using** *C-imp eval-c-elims(5)* **by** *metis*
then show *?case* **using** *1 2 wfC-elims C-imp* **by** *metis*
next
case (*C-eq e1 e2*)
obtain *be* **where** *be*: *wfCE* Θ *B* Γ *e1 be* ∧ *wfCE* Θ *B* Γ *e2 be* **using** *C-eq wfC-elims* **by** *metis*
obtain *s1* **and** *s2* **where** *1*: *eval-e i* (*e1*[*bv*::=*b*]$_{ceb}$) *s1* ∧ *eval-e i* (*e2*[*bv*::=*b*]$_{ceb}$) *s2* ∧ *s* = (*s1* = *s2*) **using** *C-eq eval-c-elims(7) subst-cb.simps(6)* **by** *metis*
obtain *s1'* **and** *s2'* **where** *2:eval-e i' e1 s1'* ∧ *eval-e i' e2 s2'* ∧ *s'* = (*s1'* = *s2'*) **using** *C-eq eval-c-elims(7)* **by** *metis*
have ⊢$_{wf}$ Θ **using** *C-eq wfX-wfY* **by** *metis*
moreover have Θ ; Γ[*bv*::=*b*]$_{\Gamma b}$ ⊢ *i* **using** *C-eq* **by** *auto*
ultimately show *?case* **using** *boxed-b-eq*[*of* Θ *s1 be bv b s1' s2 s2'*] *1 2 boxed-i-eval-ce-boxed-b C-eq wfC-elims subst-cb.simps 1 2 be* **by** *auto*
qed

lemma *is-satis-bs-boxed*:
fixes *c*::*c*
assumes *boxed-i* Θ Γ *b bv i i'* **and** *wfC* Θ *B* Γ *c* **and** *wfI* Θ Γ[*bv*::=*b*]$_{\Gamma b}$ *i* **and** Θ ; Γ ⊢ *i'*
and (*i* ⊨ *c*[*bv*::=*b*]$_{cb}$)
shows (*i'* ⊨ *c*)
proof −
have *eval-c i* (*c*[*bv*::=*b*]$_{cb}$) *True* **using** *is-satis.simps assms* **by** *auto*
moreover obtain *s* **where** *i'* ⟦ *c* ⟧ ~ *s* **using** *eval-c-exist assms* **by** *metis*
ultimately show *?thesis* **using** *eval-c-eq-bs-boxed assms is-satis.simps* **by** *metis*
qed

lemma *is-satis-bs-boxed-rev*:
fixes *c*::*c*
assumes *boxed-i* Θ Γ *b bv i i'* **and** *wfC* Θ *B* Γ *c* **and** *wfI* Θ Γ[*bv*::=*b*]$_{\Gamma b}$ *i* **and** Θ ; Γ ⊢ *i'* **and** *wfC*

Θ {||} $\Gamma[bv::=b]_{\Gamma b}$ $(c[bv::=b]_{cb})$
and $(i' \models c)$
shows $(i \models c[bv::=b]_{cb})$
proof −
have *eval-c i' c True* **using** *is-satis.simps assms* **by** *auto*
moreover obtain *s* **where** $i \llbracket c[bv::=b]_{cb} \rrbracket \sim s$ **using** *eval-c-exist assms* **by** *metis*
ultimately show *?thesis* **using** *eval-c-eq-bs-boxed assms is-satis.simps* **by** *metis*
qed

lemma *bs-boxed-wfi-aux*:
fixes *b*::*b* **and** *bv*::*bv* **and** Θ::Θ **and** B::$\mathcal{B}$
assumes *boxed-i* Θ Γ *b bv i i'* **and** *wfI* Θ $\Gamma[bv::=b]_{\Gamma b}$ *i* **and** $\vdash_{wf}$ Θ **and** *wfG* Θ *B* Γ
shows Θ ; $\Gamma \vdash i'$
using *assms* **proof**(*induct rule*: *boxed-i.inducts*)
case (*boxed-i-GNilI T i*)
then show *?case* **using** *wfI-def* **by** *auto*
next
case (*boxed-i-GConsI s i x1 T b1 bv b s' G i' c1*)
{
fix *x2 b2 c2*
assume *as* : (*x2*,*b2*,*c2*) ∈ *setG* ((*x1*, *b1*, *c1*) #$_\Gamma$ *G*)

then consider (*hd*) (*x2*,*b2*,*c2*) = (*x1*, *b1*, *c1*) | (*tail*) (*x2*,*b2*,*c2*) ∈ *setG G* **using** *setG.simps* **by** *auto*
hence ∃ *s*. *Some s* = (*i'*(*x1* ↦ *s'*)) *x2* ∧ *wfRCV T s b2* **proof**(*cases*)
case *hd*
hence *b1*=*b2* **by** *auto*
moreover have (*x2*,*b2*[*bv*::=*b*]$_{bb}$,*c2*[*bv*::=*b*]$_{cb}$) ∈ *setG* ((*x1*, *b1*, *c1*) #$_\Gamma$ *G*)[*bv*::=*b*]$_{\Gamma b}$ **using** *hd subst-gb.simps* **by** *simp*
moreover hence *wfRCV T s b2*[*bv*::=*b*]$_{bb}$ **using** *wfI-def boxed-i-GConsI hd*
proof −
obtain *ss* :: *b* ⇒ *x* ⇒ (*x* ⇒ *rcl-val option*) ⇒ *type-def list* ⇒ *rcl-val* **where**
∀ *x1a x2a x3 x4*. (∃ *v5*. *Some v5* = *x3 x2a* ∧ *wfRCV x4 v5 x1a*) = (*Some* (*ss x1a x2a x3 x4*) = *x3 x2a* ∧ *wfRCV x4* (*ss x1a x2a x3 x4*) *x1a*)
by *moura*
then have *f1*: *Some* (*ss b2*[*bv*::=*b*]$_{bb}$ *x1 i T*) = *i x1* ∧ *wfRCV T* (*ss b2*[*bv*::=*b*]$_{bb}$ *x1 i T*) *b2*[*bv*::=*b*]$_{bb}$
using *boxed-i-GConsI.prems*(*1*) *hd wfI-def* **by** *auto*
then have *ss b2*[*bv*::=*b*]$_{bb}$ *x1 i T* = *s*
by (*metis* (*no-types*) *boxed-i-GConsI.hyps*(*1*) *option.inject*)
then show *?thesis*
using *f1* **by** *blast*
qed
ultimately have *wfRCV T s' b2* **using** *boxed-i-GConsI boxed-b-wfRCV* **by** *metis*

then show *?thesis* **using** *hd* **by** *simp*
next
case *tail*
hence *wfI T G i'* **using** *boxed-i-GConsI wfI-suffix wfG-suffix subst-gb.simps*
by (*metis* (*no-types*, *lifting*) *Un-iff setG.simps*(*2*) *wfG-cons2 wfI-def*)
then show *?thesis* **using** *wfI-def*[*of T G i'*] *tail*

using *boxed-i-GConsI.prems(3) split-G wfG-cons-fresh2* **by** *fastforce*
qed
}
thus *?case* **using** *wfI-def* **by** *fast*

qed

lemma *is-satis-g-bs-boxed-aux*:
fixes $G::\Gamma$
assumes *boxed-i* Θ *G1 b bv i i'* **and** *wfI* Θ $G1[bv::=b]_{\Gamma b}$ *i* **and** *wfI* Θ *G1 i'* **and** $G1 = (G2@G)$
and *wfG* Θ *B G1*
and $(i \models G[bv::=b]_{\Gamma b})$
shows $(i' \models G)$
using *assms* **proof**(*induct G arbitrary*: *G2 rule*: Γ-*induct*)
case *GNil*
then show *?case* **by** *auto*
next
case (*GCons x' b' c'* Γ' *G2*)
show *?case* **proof**(*subst is-satis-g.simps*,*rule*)
have $*$:*wfC* Θ *B G1 c'* **using** *GCons wfG-wfC-inside* **by** *force*
show $i' \models c'$ **using** *is-satis-bs-boxed*[*OF assms*(*1*) $*$] *GCons* **by** *auto*
obtain *G3* **where** *G1* = *G3* @ Γ' **using** *GCons append-g.simps*
by (*metis append-g-assoc*)
then show $i' \models \Gamma'$ **using** *GCons append-g.simps* **by** *simp*
qed
qed

lemma *is-satis-g-bs-boxed*:
fixes $G::\Gamma$
assumes *boxed-i* Θ *G b bv i i'* **and** *wfI* Θ $G[bv::=b]_{\Gamma b}$ *i* **and** *wfI* Θ *G i'* **and** *wfG* Θ *B G*
and $(i \models G[bv::=b]_{\Gamma b})$
shows $(i' \models G)$
using *is-satis-g-bs-boxed-aux assms*
by (*metis* (*full-types*) *append-g.simps*(*1*))

lemma *subst-b-valid*:
fixes *s*::*s* **and** *b*::*b*
assumes Θ ; {||} $\vdash_{wf}$ *b* **and** $B = \{|bv|\}$ **and** Θ ; $\{|bv|\}$;Γ $\models c$
shows Θ ; {||} ; $\Gamma[bv::=b]_{\Gamma b}$ $\models c[bv::=b]_{cb}$
proof(*rule validI*)

show $**$:Θ ; {||} ; $\Gamma[bv::=b]_{\Gamma b}$ $\vdash_{wf}$ $c[bv::=b]_{cb}$ **using** *assms valid.simps wf-b-subst subst-gb.simps*
by *metis*
show $\forall i.$ (*wfI* Θ $\Gamma[bv::=b]_{\Gamma b}$ $i \wedge i \models \Gamma[bv::=b]_{\Gamma b}) \longrightarrow i \models c[bv::=b]_{cb}$
proof(*rule*,*rule*)
fix *i*
assume $*$:*wfI* Θ $\Gamma[bv::=b]_{\Gamma b}$ $i \wedge i \models \Gamma[bv::=b]_{\Gamma b}$

obtain *i'* **where** *idash*: *boxed-i* Θ Γ *b bv i i'* **using** *boxed-i-ex wfX-wfY assms* $*$ **by** *fastforce*

have *wfc*: Θ ; $\{|bv|\}$; Γ $\vdash_{wf}$ *c* **using** *valid.simps assms* **by** *simp*
have *wfg*: Θ ; $\{|bv|\}$ $\vdash_{wf}$ Γ **using** *valid.simps wfX-wfY assms* **by** *metis*
hence *wfi*: *wfI* Θ Γ *i'* **using** *idash* $*$ *bs-boxed-wfi-aux subst-gb.simps wfX-wfY* **by** *metis*
moreover have *i'* $\models$ Γ **proof** (*rule is-satis-g-bs-boxed*[*OF idash*] *wfX-wfY*(*2*)[*OF wfc*])
show *wfI* Θ Γ[*bv*::=*b*]$_{\Gamma b}$ *i* **using** *subst-gb.simps* $*$ **by** *simp*
show *wfI* Θ Γ *i'* **using** *wfi* **by** *auto*
show Θ ; *B* $\vdash_{wf}$ Γ **using** *wfg assms* **by** *auto*
show *i* $\models$ Γ[*bv*::=*b*]$_{\Gamma b}$ **using** *subst-gb.simps* $*$ **by** *simp*
qed
ultimately have *ic*:*i'* $\models$ *c* **using** *assms valid-def* **using** *valid.simps* **by** *blast*

show *i* $\models$ *c*[*bv*::=*b*]$_{cb}$ **proof**(*rule is-satis-bs-boxed-rev*)
show Θ ; Γ ; *b* , *bv* $\vdash$ *i* $\approx$ *i'* **using** *idash* **by** *auto*
show Θ ; *B* ; Γ $\vdash_{wf}$ *c* **using** *wfc assms* **by** *auto*
show Θ ; Γ[*bv*::=*b*]$_{\Gamma b}$ $\vdash$ *i* **using** *subst-gb.simps* $*$ **by** *simp*
show Θ ; Γ $\vdash$ *i'* **using** *wfi* **by** *auto*
show Θ ; $\{||\}$; Γ[*bv*::=*b*]$_{\Gamma b}$ $\vdash_{wf}$ *c*[*bv*::=*b*]$_{cb}$ **using** $**$ **by** *auto*
show *i'* $\models$ *c* **using** *ic* **by** *auto*
qed

qed
qed

11.7 Expression Operator Lemmas

lemma *is-satis-len-imp*:
assumes *i* $\models$ (*CE-val* (*V-var x*) == *CE-val* (*V-lit* (*L-num* (*int* (*length v*))))) (**is** *is-satis i ?c1*)
shows *i* $\models$ (*CE-val* (*V-var x*) == *CE-len* [*V-lit* (*L-bitvec v*)]^{ce})
proof −
have $*$:*eval-c i ?c1 True* **using** *assms is-satis.simps* **by** *blast*
then have *eval-e i* (*CE-val* (*V-lit* (*L-num* (*int* (*length v*))))) (*SNum* (*int* (*length v*)))
using *eval-e-elims*(*1*) *eval-v-elims eval-l.simps* **by** (*metis eval-e.intros*(*1*) *eval-v-litI*)
hence *eval-e i* (*CE-val* (*V-var x*)) (*SNum* (*int* (*length v*))) **using** *eval-c-elims*(*7*)[*OF* $*$]
by (*metis eval-e-elims*(*1*) *eval-v-elims*(*1*))
moreover have *eval-e i* (*CE-len* [*V-lit* (*L-bitvec v*)]^{ce}) (*SNum* (*int* (*length v*)))
using *eval-e-elims*(*7*) *eval-v-elims eval-l.simps* **by** (*metis eval-e.intros eval-v-litI*)
ultimately show *?thesis* **using** *eval-c.intros is-satis.simps* **by** *fastforce*
qed

lemma *is-satis-plus-imp*:
assumes *i* $\models$ (*CE-val* (*V-var x*) == *CE-val* (*V-lit* (*L-num* (*n1*+*n2*)))) (**is** *is-satis i ?c1*)
shows *i* $\models$ (*CE-val* (*V-var x*) == *CE-op Plus* ([*V-lit* (*L-num n1*)]^{ce}) ([*V-lit* (*L-num n2*)]^{ce}))
proof −
have $*$:*eval-c i ?c1 True* **using** *assms is-satis.simps* **by** *blast*
then have *eval-e i* (*CE-val* (*V-lit* (*L-num* (*n1*+*n2*)))) (*SNum* (*n1*+*n2*))
using *eval-e-elims*(*1*) *eval-v-elims eval-l.simps* **by** (*metis eval-e.intros*(*1*) *eval-v-litI*)
hence *eval-e i* (*CE-val* (*V-var x*)) (*SNum* (*n1*+*n2*)) **using** *eval-c-elims*(*7*)[*OF* $*$]
by (*metis eval-e-elims*(*1*) *eval-v-elims*(*1*))
moreover have *eval-e i* (*CE-op Plus* ([*V-lit* (*L-num n1*)]^{ce}) ([*V-lit* (*L-num n2*)]^{ce})) (*SNum* (*n1*+*n2*))
using *eval-e-elims*(*7*) *eval-v-elims eval-l.simps* **by** (*metis eval-e.intros eval-v-litI*)
ultimately show *?thesis* **using** *eval-c.intros is-satis.simps* **by** *fastforce*

qed

lemma *is-satis-leq-imp*:
assumes $i \models (\textit{CE-val}\ (\textit{V-var}\ x) == \textit{CE-val}\ (\textit{V-lit}\ (\textit{if}\ (n1 \leq n2)\ \textit{then L-true else L-false})))$ (**is** *is-satis i ?c1*)
shows $i \models (\textit{CE-val}\ (\textit{V-var}\ x) == \textit{CE-op LEq}\ [(\textit{V-lit}\ (\textit{L-num}\ n1))]^{ce}\ [(\textit{V-lit}\ (\textit{L-num}\ n2))]^{ce})$
proof −
have ∗:*eval-c i ?c1 True* **using** *assms is-satis.simps* **by** *blast*
then have *eval-e i* $(\textit{CE-val}\ (\textit{V-lit}\ ((\textit{if}\ (n1 \leq n2)\ \textit{then L-true else L-false}))))\ (\textit{SBool}\ (n1 \leq n2))$
using *eval-e-elims*(*1*) *eval-v-elims eval-l.simps*
by (*metis* (*full-types*) *eval-e.intros*(*1*) *eval-v-litI*)
hence *eval-e i* $(\textit{CE-val}\ (\textit{V-var}\ x))\ (\textit{SBool}\ (n1 \leq n2))$ **using** *eval-c-elims*(*7*)[*OF* ∗]
by (*metis eval-e-elims*(*1*) *eval-v-elims*(*1*))
moreover have *eval-e i* $(\textit{CE-op LEq}\ [(\textit{V-lit}\ (\textit{L-num}\ n1))]^{ce}\ [(\textit{V-lit}\ (\textit{L-num}\ n2))]^{ce})\ (\textit{SBool}\ (n1 \leq n2))$
using *eval-e-elims*(*3*) *eval-v-elims eval-l.simps* **by** (*metis eval-e.intros eval-v-litI*)
ultimately show *?thesis* **using** *eval-c.intros is-satis.simps* **by** *fastforce*
qed

lemma *valid-eq-e*:
assumes $\forall\, i\ s1\ s2.\ \textit{wfG}\ P\ \mathcal{B}\ \textit{GNil} \wedge \textit{wfI}\ P\ \textit{GNil}\ i \wedge \textit{eval-e}\ i\ e1\ s1 \wedge \textit{eval-e}\ i\ e2\ s2 \longrightarrow s1 = s2$
and *wfCE P* $\mathcal{B}$ *GNil e1 b* **and** *wfCE P* $\mathcal{B}$ *GNil e2 b*
shows $P\ ;\ \mathcal{B}\ ;\ (x,\ b\ ,\ \textit{CE-val}\ (\textit{V-var}\ x)\ ==\ e1\ )\#_\Gamma\ \textit{GNil} \models \textit{CE-val}\ (\textit{V-var}\ x)\ ==\ e2$
unfolding *valid.simps*
proof(*intro conjI*)
show $\langle\ P\ ;\ \mathcal{B}\ ;\ (x,\ b,\ [\ [\ x\ ]^v\ ]^{ce}\ ==\ e1\ )\ \#_\Gamma\ \textit{GNil}\ \vdash_{wf}\ [\ [\ x\ ]^v\ ]^{ce}\ ==\ e2\ \rangle$
using *assms wf-intros wfX-wfY b.eq-iff fresh-GNil wfC-e-eq2 wfV-elims* **by** *meson*
show $\langle\forall\, i.\ ((P\ ;\ (x,\ b,\ [\ [\ x\ ]^v\ ]^{ce}\ ==\ e1\ )\ \#_\Gamma\ \textit{GNil} \vdash i) \wedge\ (i \models (x,\ b,\ [\ [\ x\ ]^v\ ]^{ce}\ ==\ e1\ )\ \#_\Gamma\ \textit{GNil})\ \longrightarrow$
$(i \models [\ [\ x\ ]^v\ ]^{ce}\ ==\ e2))\ \rangle$ **proof**(*rule+*)
fix *i*
assume *as*:$P\ ;\ (x,\ b,\ [\ [\ x\ ]^v\ ]^{ce}\ ==\ e1\ )\ \#_\Gamma\ \textit{GNil} \vdash i \wedge\ i \models (x,\ b,\ [\ [\ x\ ]^v\ ]^{ce}\ ==\ e1\ )\ \#_\Gamma\ \textit{GNil}$

have ∗: $P\ ;\ \textit{GNil} \vdash i$ **using** *wfI-def* **by** *auto*

then obtain *s1* **where** *s1*:*eval-e i e1 s1* **using** *assms eval-e-exist* **by** *metis*
obtain *s2* **where** *s2*:*eval-e i e2 s2* **using** *assms eval-e-exist* ∗ **by** *metis*
moreover have $i\ x = \textit{Some}\ s1$ **proof** −
have $i \models [\ [\ x\ ]^v\ ]^{ce}\ ==\ e1$ **using** *as is-satis-g.simps* **by** *auto*
thus *?thesis* **using** *s1*
by (*metis eval-c-elims*(*7*) *eval-e-elims*(*1*) *eval-e-uniqueness eval-v-elims*(*2*) *is-satis.cases*)
qed
moreover have *s1 = s2* **using** *s1 s2* ∗ *assms wfG-nilI wfX-wfY* **by** *metis*

ultimately show $i \llbracket\ [\ [\ x\ ]^v\ ]^{ce}\ ==\ e2\ \rrbracket \sim \textit{True}$
using *eval-c.intros eval-e.intros eval-v.intros*
proof −
have $i \llbracket\ e2\ \rrbracket \sim s1$
by (*metis* ⟨*s1 = s2*⟩ *s2*)
then show *?thesis*
by (*metis* (*full-types*) ⟨$i\ x = \textit{Some}\ s1$⟩ *eval-c-eqI eval-e-valI eval-v-varI*)

qed
qed
qed

lemma *valid-len*:
assumes $\vdash_{wf} \Theta$
shows Θ ; $\mathcal{B}$; (*x*, *B-int*, $[[x]^v]^{ce}$ == $[[$ *L-num* (*int* (*length v*)) $]^v]^{ce}$) $\#_\Gamma$ *GNil* $\models$ $[[x]^v]^{ce}$ == *CE-len* $[[$ *L-bitvec v* $]^v]^{ce}$ (**is** Θ ; $\mathcal{B}$; *?G* $\models$ *?c*)
proof −
have $*$:Θ $\vdash_{wf}$ ($[]$::Φ) $\wedge$ Θ ; $\mathcal{B}$; *GNil* $\vdash_{wf}$ $[]_\Delta$ **using** *assms wfG-nilI wfD-emptyI wfPhi-emptyI* **by** *auto*

moreover hence Θ ; $\mathcal{B}$; *GNil* $\vdash_{wf}$ *CE-val* (*V-lit* (*L-num* (*int* (*length v*)))) : *B-int*
using *wfCE-valI* $*$ *wfV-litI base-for-lit.simps*
by (*metis wfE-valI wfX-wfY*)

moreover have Θ ; $\mathcal{B}$; *GNil* $\vdash_{wf}$ *CE-len* $[($*V-lit* (*L-bitvec v*))$]^{ce}$: *B-int*
using *wfE-valI* $*$ *wfV-litI base-for-lit.simps wfE-valI wfX-wfY wfCE-valI*
by (*metis wfCE-lenI*)
moreover have *atom x* $\sharp$ *GNil* **by** *auto*
ultimately have Θ ; $\mathcal{B}$; *?G* $\vdash_{wf}$ *?c* **using** *wfC-e-eq2 assms* **by** *simp*
moreover have ($\forall$ *i*. *wfI* Θ *?G i* $\wedge$ *is-satis-g i ?G* $\longrightarrow$ *is-satis i ?c*) **using** *is-satis-len-imp* **by** *auto*
ultimately show *?thesis* **using** *valid.simps* **by** *auto*
qed

lemma *valid-bop*:
assumes *wfG* Θ $\mathcal{B}$ Γ **and** *opp* = *Plus* $\wedge$ *ll* = (*L-num* (*n1*+*n2*)) $\vee$ (*opp* = *LEq* $\wedge$ *ll* = (*if n1*$\leq$*n2 then L-true else L-false*))
and (*opp* = *Plus* $\longrightarrow$ *b* = *B-int*) $\wedge$ (*opp* = *LEq* $\longrightarrow$ *b* = *B-bool*) **and**
atom x $\sharp$ Γ
shows Θ; $\mathcal{B}$; (*x*, *b*, (*CE-val* (*V-var x*) == *CE-val* (*V-lit* (*ll*)))) $\#_\Gamma$ Γ
$\models$ (*CE-val* (*V-var x*) == *CE-op opp* ($[$*V-lit* (*L-num n1*)$]^{ce}$) ($[$*V-lit* (*L-num n2*)$]^{ce}$)) (**is** Θ ; $\mathcal{B}$; *?G* $\models$ *?c*)
proof −
have *wfC* Θ $\mathcal{B}$ *?G ?c* **proof**(*rule wfC-e-eq2*)
show Θ ; $\mathcal{B}$; Γ $\vdash_{wf}$ *CE-val* (*V-lit ll*) : *b* **using** *wfCE-valI wfV-litI assms base-for-lit.simps* **by** *metis*
show Θ ; $\mathcal{B}$; Γ $\vdash_{wf}$ *CE-op opp* ($[$*V-lit* (*L-num n1*)$]^{ce}$) ($[$*V-lit* (*L-num n2*)$]^{ce}$) : *b*
using *wfCE-plusI wfCE-leqI wfV-litI wfCE-valI base-for-lit.simps assms* **by** *metis*
show $\vdash_{wf}$ Θ **using** *assms wfX-wfY* **by** *auto*
show *atom x* $\sharp$ Γ **using** *assms* **by** *auto*
qed

moreover have $\forall$ *i*. *wfI* Θ *?G i* $\wedge$ *is-satis-g i ?G* $\longrightarrow$ *is-satis i ?c* **proof**(*rule allI* , *rule impI*)
fix *i*
assume *wfI* Θ *?G i* $\wedge$ *is-satis-g i ?G*

hence *is-satis i* ((*CE-val* (*V-var x*) == *CE-val* (*V-lit* (*ll*)))) **by** *auto*
thus *is-satis i* ((*CE-val* (*V-var x*) == *CE-op opp* ($[$*V-lit* (*L-num n1*)$]^{ce}$) ($[$*V-lit* (*L-num n2*)$]^{ce}$)))

using *is-satis-plus-imp assms opp.exhaust is-satis-leq-imp* **by** *auto*
qed
ultimately show *?thesis* **using** *valid.simps* **by** *metis*
qed

lemma *valid-fst*:
fixes x::x **and** v_1::v **and** v_2::v
assumes *wfTh* Θ **and** *wfV* Θ $\mathcal{B}$ *GNil* (*V-pair* v_1 v_2) (*B-pair* b_1 b_2)
shows Θ ; $\mathcal{B}$; (x, b_1, $[[x]^v]^{ce}$ == $[v_1]^{ce}$) $\#_\Gamma$ *GNil* $\models$ $[[x]^v]^{ce}$ == $[\#1[[v_1,v_2]^v]^{ce}]^{ce}$
proof(*rule valid-eq-e*)
show ‹$\forall$ *i s1 s2*. (Θ ; $\mathcal{B}$ $\vdash_{wf}$ *GNil*) $\wedge$ (Θ ; *GNil* $\vdash$ *i*) $\wedge$ (*i* ⟦ [v_1]^{ce} ⟧ ~ *s1*) $\wedge$ (*i* ⟦ [*#1*[[v_1 , v_2]^v]^{ce}]^{ce} ⟧ ~ *s2*) ⟶ *s1* = *s2*›
proof(*rule+*)
fix *i s1 s2*
assume *as*:Θ ; $\mathcal{B}$ $\vdash_{wf}$ *GNil* $\wedge$ Θ ; *GNil* $\vdash$ *i* $\wedge$ (*i* ⟦ [v_1]^{ce} ⟧ ~ *s1*) $\wedge$ (*i* ⟦ [*#1*[[v_1 , v_2]^v]^{ce}]^{ce} ⟧ ~ *s2*)
then obtain *s2'* **where** ∗:*i* ⟦ [v_1 , v_2]^v ⟧ ~ *SPair s2 s2'*
using *eval-e-elims*(*4*)[*of i* [[v_1 , v_2]^v]^{ce} *s2*] *eval-e-elims*
by *meson*
then have *i* ⟦ v_1 ⟧ ~ *s2* **using** *eval-v-elims*(*3*)[*OF* ∗] **by** *auto*
then show *s1* = *s2* **using** *eval-v-uniqueness as*
using *eval-e-uniqueness eval-e-valI* **by** *blast*
qed

show ‹ Θ ; $\mathcal{B}$; *GNil* $\vdash_{wf}$ [v_1]^{ce} : b_1 › **using** *assms*
by (*metis b.eq-iff*(*4*) *wfV-elims*(*3*) *wfV-wfCE*)
show ‹ Θ ; $\mathcal{B}$; *GNil* $\vdash_{wf}$ [*#1*[[v_1 , v_2]^v]^{ce}]^{ce} : b_1 › **using** *assms* **using** *wfCE-fstI*
using *wfCE-valI* **by** *blast*
qed

lemma *valid-snd*:
fixes x::x **and** v_1::v **and** v_2::v
assumes *wfTh* Θ **and** *wfV* Θ $\mathcal{B}$ *GNil* (*V-pair* v_1 v_2) (*B-pair* b_1 b_2)
shows Θ ; $\mathcal{B}$; (x, b_2, $[[x]^v]^{ce}$ == $[v_2]^{ce}$) $\#_\Gamma$ *GNil* $\models$ $[[x]^v]^{ce}$ == $[\#2[[v_1,v_2]^v]^{ce}]^{ce}$
proof(*rule valid-eq-e*)
show ‹$\forall$ *i s1 s2*. (Θ ; $\mathcal{B}$ $\vdash_{wf}$ *GNil*) $\wedge$ (Θ ; *GNil* $\vdash$ *i*) $\wedge$ (*i* ⟦ [v_2]^{ce} ⟧ ~ *s1*) $\wedge$ (*i* ⟦ [*#2*[[v_1 , v_2]^v]^{ce}]^{ce} ⟧ ~ *s2*) ⟶ *s1* = *s2*›
proof(*rule+*)
fix *i s1 s2*
assume *as*:Θ ; $\mathcal{B}$ $\vdash_{wf}$ *GNil* $\wedge$ Θ ; *GNil* $\vdash$ *i* $\wedge$ (*i* ⟦ [v_2]^{ce} ⟧ ~ *s1*) $\wedge$ (*i* ⟦ [*#2*[[v_1 , v_2]^v]^{ce}]^{ce} ⟧ ~ *s2*)
then obtain *s2'* **where** ∗:*i* ⟦ [v_1 , v_2]^v ⟧ ~ *SPair s2' s2*
using *eval-e-elims*(*4*)[*of i* [[v_1 , v_2]^v]^{ce} *s2*] *eval-e-elims*
by *meson*
then have *i* ⟦ v_2 ⟧ ~ *s2* **using** *eval-v-elims*(*3*)[*OF* ∗] **by** *auto*
then show *s1* = *s2* **using** *eval-v-uniqueness as*
using *eval-e-uniqueness eval-e-valI* **by** *blast*
qed

show ‹ Θ ; $\mathcal{B}$; *GNil* $\vdash_{wf}$ [v_2]^{ce} : b_2 › **using** *assms*

by (*metis b.eq-iff wfV-elims wfV-wfCE*)
show ‹ Θ ; $\mathcal{B}$; *GNil* $\vdash_{wf}$ [#*2*[[v_1 , v_2]^{v}]^{ce}]^{ce} : b_2 › **using** *assms* **using** *wfCE-sndI wfCE-valI* **by** *blast*
qed

lemma *valid-concat*:
fixes *v1*::*bit list* **and** *v2*::*bit list*
assumes $\vdash_{wf}$ Π
shows Π ; $\mathcal{B}$; (*x*, *B-bitvec*, (*CE-val* (*V-var x*) == *CE-val* (*V-lit* (*L-bitvec* (*v1*@ *v2*))))) $\#_\Gamma$ *GNil* $\models$ (*CE-val* (*V-var x*) == *CE-concat* ([*V-lit* (*L-bitvec v1*)]^{ce}) ([*V-lit* (*L-bitvec v2*)]^{ce}))
proof(*rule valid-eq-e*)
show ‹$\forall$ *i s1 s2*. ((Π ; $\mathcal{B}$ $\vdash_{wf}$ *GNil*) $\wedge$ (Π ; *GNil* $\vdash$ *i*) $\wedge$ (*i* ⟦ [[*L-bitvec* (*v1* @ *v2*)]^{v}]^{ce} ⟧ ~ *s1*) $\wedge$ (*i* ⟦ [[[*L-bitvec v1*]^{v}]^{ce} @@ [[*L-bitvec v2*]^{v}]^{ce}]^{ce} ⟧ ~ *s2*) $\longrightarrow$ *s1* = *s2*)›
proof(*rule+*)
fix *i s1 s2*
assume *as*: (Π ; $\mathcal{B}$ $\vdash_{wf}$ *GNil*) $\wedge$ (Π ; *GNil* $\vdash$ *i*) $\wedge$ (*i* ⟦ [[*L-bitvec* (*v1* @ *v2*)]^{v}]^{ce} ⟧ ~ *s1*) $\wedge$ (*i* ⟦ [[[*L-bitvec v1*]^{v}]^{ce} @@ [[*L-bitvec v2*]^{v}]^{ce}]^{ce} ⟧ ~ *s2*)

hence $*$: *i* ⟦ [[[*L-bitvec v1*]^{v}]^{ce} @@ [[*L-bitvec v2*]^{v}]^{ce}]^{ce} ⟧ ~ *s2* **by** *auto*
obtain *bv1 bv2* **where** *s2*:*s2* = *SBitvec* (*bv1* @ *bv2*) $\wedge$ *i* ⟦ [*L-bitvec v1*]^{v} ⟧ ~ *SBitvec bv1* $\wedge$ (*i* ⟦ [*L-bitvec v2*]^{v} ⟧ ~ *SBitvec bv2*)
using *eval-e-elims*(*6*)[*OF* $*$] *eval-e-elims*(*1*) **by** *metis*
hence *v1* = *bv1* $\wedge$ *v2* = *bv2* **using** *eval-v-elims*(*1*) *eval-l.simps*(*5*) **by** *force*
moreover then have *s1* = *SBitvec* (*bv1* @ *bv2*) **using** *s2* **using** *eval-v-elims*(*1*) *eval-l.simps*(*5*)
by (*metis as eval-e-elims*(*1*))

then show *s1* = *s2* **using** *s2* **by** *auto*
qed

show ‹ Π ; $\mathcal{B}$; *GNil* $\vdash_{wf}$ [[*L-bitvec* (*v1* @ *v2*)]^{v}]^{ce} : *B-bitvec* ›
by (*metis assms base-for-lit.simps*(*5*) *wfG-nilI wfV-litI wfV-wfCE*)
show ‹ Π ; $\mathcal{B}$; *GNil* $\vdash_{wf}$ [[[*L-bitvec v1*]^{v}]^{ce} @@ [[*L-bitvec v2*]^{v}]^{ce}]^{ce} : *B-bitvec* ›
by (*metis assms base-for-lit.simps*(*5*) *wfCE-concatI wfG-nilI wfV-litI wfCE-valI*)
qed

lemma *valid-ce-eq*:
fixes *ce*::*ce*
assumes Θ ; $\mathcal{B}$; Γ $\vdash_{wf}$ *ce* : *b*
shows ‹Θ ; $\mathcal{B}$; Γ $\models$ *ce* == *ce* ›
unfolding *valid.simps* **proof**
show ‹ Θ ; $\mathcal{B}$; Γ $\vdash_{wf}$ *ce* == *ce* › **using** *assms wfC-eqI* **by** *auto*
show ‹$\forall$ *i*. Θ ; Γ $\vdash$ *i* $\wedge$ *i* $\models$ Γ $\longrightarrow$ *i* $\models$ *ce* == *ce* › **proof**(*rule+*)
fix *i*
assume Θ ; Γ $\vdash$ *i* $\wedge$ *i* $\models$ Γ
then obtain *s* **where** *i*⟦ *ce* ⟧ ~ *s* **using** *assms eval-e-exist* **by** *metis*
then show *i* ⟦ *ce* == *ce* ⟧ ~ *True* **using** *eval-c-eqI* **by** *metis*
qed
qed

lemma *valid-eq-imp*:
fixes *c1*::*c* **and** *c2*::*c*
assumes Θ ; $\mathcal{B}$; (*x*, *b*, *c2*) $\#_\Gamma$ Γ $\vdash_{wf}$ *c1 IMP c2*
shows Θ ; $\mathcal{B}$; (*x*, *b*, *c2*) $\#_\Gamma$ Γ $\models$ *c1 IMP c2*
proof −
have $\forall\, i.$ (Θ ; (*x*, *b*, *c2*) $\#_\Gamma$ $\Gamma \vdash i \wedge$ *i* $\models$ (*x*, *b*, *c2*) $\#_\Gamma$ Γ) $\longrightarrow$ *i* $\models$ (*c1 IMP c2*)
proof(*rule*,*rule*)
fix *i*
assume *as*:Θ ; (*x*, *b*, *c2*) $\#_\Gamma$ $\Gamma \vdash i \wedge$ *i* $\models$ (*x*, *b*, *c2*) $\#_\Gamma$ Γ

have Θ ; $\mathcal{B}$; (*x*, *b*, *c2*) $\#_\Gamma$ Γ $\vdash_{wf}$ *c1* **using** *wfC-elims assms* **by** *metis*

then obtain *sc* **where** *i* ⟦ *c1* ⟧ ~ *sc* **using** *eval-c-exist assms as* **by** *metis*
moreover have *i* ⟦ *c2* ⟧ ~ *True* **using** *as is-satis-g.simps is-satis.simps* **by** *auto*

ultimately have *i* ⟦ *c1 IMP c2* ⟧ ~ *True* **using** *eval-c-impI* **by** *metis*

thus *i* $\models$ *c1 IMP c2* **using** *is-satis.simps* **by** *auto*
qed
thus *?thesis* **using** *assms* **by** *auto*
qed

lemma *valid-range*:
assumes $0 \leq n \wedge n \leq m$ **and** $\vdash_{wf}$ Θ
shows Θ ; {||} ; (*x*, *B-int* , (*C-eq* (*CE-val* (*V-var x*)) (*CE-val* (*V-lit* (*L-num n*))))) $\#_\Gamma$ *GNil* $\models$
(*C-eq* (*CE-op LEq* (*CE-val* (*V-var x*)) (*CE-val* (*V-lit* (*L-num m*)))) [[*L-true*]^v]^{ce}) *AND*
(*C-eq* (*CE-op LEq* (*CE-val* (*V-lit* (*L-num 0*))) (*CE-val* (*V-var x*))) [[*L-true*]^v]^{ce})
(**is** Θ ; {||} ; *?G* $\models$ *?c1 AND ?c2*)
proof(*rule validI*)
have *wfg*: Θ ; {||} $\vdash_{wf}$ (*x*, *B-int*, [[*x*]^v]^{ce} == [[*L-num n*]^v]^{ce}) $\#_\Gamma$ *GNil*
using *assms base-for-lit.simps wfG-nilI wfV-litI fresh-GNil wfB-intI wfC-v-eq wfG-cons1I wfG-cons2I* **by** *metis*

show Θ ; {||} ; *?G* $\vdash_{wf}$ *?c1 AND ?c2*
using *wfC-conjI wfC-eqI wfCE-leqI wfCE-valI wfV-varI wfg lookup.simps base-for-lit.simps wfV-litI wfB-intI wfB-boolI*
by *metis*

show $\forall\, i.$ Θ ; *?G* $\vdash$ *i* $\wedge$ *i* $\models$ *?G* $\longrightarrow$ *i* $\models$ *?c1 AND ?c2* **proof**(*rule*,*rule*)
fix *i*
assume *a*:Θ ; *?G* $\vdash$ *i* $\wedge$ *i* $\models$ *?G*
hence ∗:*i* ⟦ *V-var x* ⟧ ~ *SNum n*
proof −
obtain *sv* **where** *sv*: *i x* = *Some sv* $\wedge$ $\Theta \vdash$ *sv* : *B-int* **using** *a wfI-def* **by** *force*
have *i* ⟦ (*C-eq* (*CE-val* (*V-var x*)) (*CE-val* (*V-lit* (*L-num n*)))) ⟧ ~ *True*
using *a is-satis-g.simps*
using *is-satis.cases* **by** *blast*
hence *i x* = *Some*(*SNum n*) **using** *sv*
by (*metis eval-c-elims*(*7*) *eval-e-elims*(*1*) *eval-l.simps*(*3*) *eval-v-elims*(*1*) *eval-v-elims*(*2*))

```
    thus ?thesis using eval-v-varI by auto
  qed

  show i ⊨ ?c1 AND ?c2
  proof −
    have i ⟦ ?c1 ⟧ ~ True
    proof −
      have i ⟦ [ leq [ [ x ]^v ]^ce [ [ L-num m ]^v ]^ce ]^ce⟧ ~ SBool True
        using eval-e-leqI assms eval-v-litI eval-l.simps ∗
        by (metis (full-types) eval-e-valI)
      moreover have i ⟦ [ [ L-true ]^v ]^ce ⟧ ~ SBool True
        using eval-v-litI eval-e-valI eval-l.simps by metis
      ultimately show ?thesis using eval-c-eqI by metis
    qed

    moreover have i ⟦ ?c2 ⟧ ~ True
    proof −
      have i ⟦ [ leq [ [ L-num 0 ]^v ]^ce [ [ x ]^v ]^ce ]^ce ⟧ ~ SBool True
      using eval-e-leqI assms eval-v-litI eval-l.simps ∗
        by (metis (full-types) eval-e-valI)
      moreover have i ⟦ [ [ L-true ]^v ]^ce ⟧ ~ SBool True
        using eval-v-litI eval-e-valI eval-l.simps by metis
      ultimately show ?thesis using eval-c-eqI by metis
    qed
    ultimately show ?thesis using eval-c-conjI is-satis.simps by metis
  qed
qed
qed


lemma valid-range-length:
  fixes Γ::Γ
  assumes 0 ≤ n ∧ n ≤ int (length v) and Θ ; {||} ⊢_wf Γ and atom x ♯ Γ
  shows Θ ; {||} ; (x, B-int , (C-eq (CE-val (V-var x)) (CE-val (V-lit (L-num n))))) #_Γ Γ ⊨
                (C-eq (CE-op LEq (CE-val (V-lit (L-num 0))) (CE-val (V-var x))) [[ L-true ]^v ]^ce)
AND
                (C-eq (CE-op LEq (CE-val (V-var x)) ([| [ [ L-bitvec v ]^v ]^ce |]^ce )) [[ L-true ]^v ]^ce)

      (is Θ ; {||} ; ?G ⊨ ?c1 AND ?c2)
proof(rule validI)
  have wfg: Θ ; {||} ⊢_wf (x, B-int, [ [ x ]^v ]^ce == [ [ L-num n ]^v ]^ce ) #_Γ Γ apply(rule wfG-cons1I)
      apply simp
    using assms apply simp+
     using assms base-for-lit.simps wfG-nilI wfV-litI wfB-intI wfC-v-eq wfB-intI wfX-wfY assms by
metis+

  show Θ ; {||} ; ?G ⊢_wf ?c1 AND ?c2
    using wfC-conjI wfC-eqI wfCE-leqI wfCE-valI wfV-varI wfg lookup.simps base-for-lit.simps wfV-litI
wfB-intI wfB-boolI
    by (metis (full-types) wfCE-lenI)

  show ∀ i. Θ ; ?G ⊢ i ∧ i ⊨ ?G ⟶ i ⊨ ?c1 AND ?c2 proof(rule,rule)
```

```
    fix i
    assume a:Θ ; ?G ⊢ i ∧  i ⊨ ?G
    hence ∗:i ⟦ V-var x ⟧ ~ SNum n
    proof −
      obtain sv where sv: i x = Some sv ∧ Θ ⊢ sv : B-int using a wfI-def by force
      have i ⟦ (C-eq (CE-val (V-var x)) (CE-val (V-lit (L-num n)))) ⟧ ~ True
        using a is-satis-g.simps
        using is-satis.cases by blast
      hence i x = Some(SNum n) using sv
        by (metis eval-c-elims(7) eval-e-elims(1) eval-l.simps(3) eval-v-elims(1) eval-v-elims(2))
      thus ?thesis using eval-v-varI by auto
    qed

    show i ⊨ ?c1 AND ?c2
    proof −
      have i ⟦ ?c2 ⟧ ~ True
      proof −
        have i ⟦ [ leq [ [ x ]^v ]^ce [| [ [ L-bitvec v ]^v ]^ce |]^ce ]^ce⟧ ~ SBool True
          using eval-e-leqI assms eval-v-litI eval-l.simps ∗
          by (metis (full-types) eval-e-lenI eval-e-valI)
        moreover have i ⟦ [ [ L-true ]^v ]^ce  ⟧ ~ SBool True
          using eval-v-litI eval-e-valI eval-l.simps by metis
        ultimately show ?thesis using  eval-c-eqI by metis
      qed

      moreover have i ⟦ ?c1 ⟧ ~ True
      proof −
        have i ⟦ [ leq [ [ L-num 0 ]^v ]^ce [ [ x ]^v ]^ce ]^ce ⟧ ~ SBool True
        using eval-e-leqI assms eval-v-litI eval-l.simps ∗
          by (metis (full-types) eval-e-valI)
        moreover have i ⟦ [ [ L-true ]^v ]^ce  ⟧ ~ SBool True
          using eval-v-litI eval-e-valI eval-l.simps by metis
        ultimately show ?thesis using  eval-c-eqI by metis
      qed
      ultimately show ?thesis using eval-c-conjI is-satis.simps by metis
  qed
qed
qed

thm valid-weakening

lemma valid-range-length-inv-gnil:
  fixes Γ::Γ
  assumes ⊢_wf Θ
  and  Θ ; {||} ; (x, B-int  , (C-eq (CE-val (V-var x)) (CE-val (V-lit (L-num n))))) #_Γ  GNil ⊨
                  (C-eq (CE-op LEq (CE-val (V-lit (L-num 0))) (CE-val (V-var x)))  [[ L-true ]^v ]^ce)
AND
                  (C-eq (CE-op LEq (CE-val (V-var x)) ([| [ [ L-bitvec v ]^v ]^ce |]^ce ))  [[ L-true ]^v ]^ce)

      (is Θ ; {||} ; ?G ⊨ ?c1 AND ?c2)
    shows 0 ≤ n ∧ n ≤ int (length v)
proof −
```

have $*$:$\forall\, i.\ \ \Theta\ ;\ ?G \vdash i \wedge\ \ i \models ?G\ \longrightarrow i \models ?c1\ AND\ ?c2$ **using** *assms valid.simps* **by** *simp*

obtain i **where** i: $i\ x = Some\ (SNum\ n)$ **by** *auto*
have $\Theta\ ;\ ?G \vdash i \wedge\ \ i \models ?G$ **proof**
show $\Theta\ ;\ ?G \vdash i$ **unfolding** *wfI-def* **using** *wfRCV-BIntI* i $*$ **by** *auto*
have $i \llbracket ([\ [\ x\ ]^v\ ]^{ce}\ ==\ [\ [\ L\text{-}num\ n\ ]^v\ ]^{ce}\ ) \rrbracket \sim True$
using $*$ *eval-c.intros*(7) *eval-e.intros eval-v.intros eval-l.simps*
by (*metis* (*full-types*) i)
thus $i \models ?G$ **unfolding** *is-satis-g.simps is-satis.simps* **by** *auto*
qed
hence $**$:$i \models ?c1\ AND\ ?c2$ **using** $*$ **by** *auto*

hence *1*: $i \llbracket\ ?c1\ \rrbracket \sim True$ **using** *eval-c-elims*(3) *is-satis.simps*
by *fastforce*
then obtain *sv1* **and** *sv2* **where** $(sv1 = sv2) = True \wedge i \llbracket [\ leq\ [\ [\ L\text{-}num\ 0\ ]^v\ ]^{ce}\ [\ [\ x\ ]^v\ ]^{ce}\ ]^{ce} \rrbracket \sim sv1 \wedge i \llbracket [\ [\ L\text{-}true\ ]^v\ ]^{ce}\ \rrbracket \sim sv2$
using *eval-c-elims*(7) **by** *metis*
hence $sv1 = SBool\ True$ **using** *eval-e-elims eval-v-elims eval-l.simps* i **by** *metis*
obtain *n1* **and** *n2* **where** $SBool\ True = SBool\ (n1 \leq n2) \wedge (i \llbracket [\ [\ L\text{-}num\ 0\ ]^v\ ]^{ce} \rrbracket \sim SNum\ n1) \wedge (i \llbracket [\ [\ x\ ]^v\ ]^{ce} \rrbracket \sim SNum\ n2)$
using *eval-e-elims*(3)[*of* i $[\ [\ L\text{-}num\ 0\ ]^v\ ]^{ce}\ [\ [\ x\ ]^v\ ]^{ce}$ $SBool\ True$]
using $\langle(sv1 = sv2) = True \wedge i \llbracket [\ leq\ [\ [\ L\text{-}num\ 0\ ]^v\ ]^{ce}\ [\ [\ x\ ]^v\ ]^{ce}\ ]^{ce} \rrbracket \sim sv1 \wedge i \llbracket [\ [\ [\ L\text{-}true\ ]^v\ ]^{ce} \rrbracket \sim sv2\rangle\ \langle sv1 = SBool\ True\rangle$ **by** *fastforce*
moreover hence $n1 = 0$ **and** $n2 = n$ **using** *eval-e-elims eval-v-elims* i
apply (*metis eval-l.simps*(3) *rcl-val.eq-iff*(2))
using *eval-e-elims eval-v-elims* i
by (*metis calculation option.inject rcl-val.eq-iff*(2))
ultimately have *le1*: $0 \leq n$ **by** *simp*

hence *2*: $i \llbracket\ ?c2\ \rrbracket \sim True$ **using** $**$ *eval-c-elims*(3) *is-satis.simps*
by *fastforce*
then obtain *sv1* **and** *sv2* **where** *sv*: $(sv1 = sv2) = True \wedge i \llbracket [\ [\ leq\ [\ [\ x\ ]^v\ ]^{ce}\ \ [|\ [\ [\ L\text{-}bitvec\ v\ ]^v\ ]^{ce}\ |]^{ce}\ ]^{ce}\ \rrbracket \sim sv1 \wedge i \llbracket [\ [\ L\text{-}true\ ]^v\ ]^{ce}\ \rrbracket \sim sv2$
using *eval-c-elims*(7) **by** *metis*
hence $sv1 = SBool\ True$ **using** *eval-e-elims eval-v-elims eval-l.simps* i **by** *metis*
obtain *n1* **and** *n2* **where** $***$:$SBool\ True = SBool\ (n1 \leq n2) \wedge\ (i \llbracket [\ [\ x\ ]^v\ ]^{ce} \rrbracket \sim SNum\ n1) \wedge (i \llbracket [|\ [\ [\ L\text{-}bitvec\ v\ ]^v\ ]^{ce}\ |]^{ce}\ \rrbracket \sim SNum\ n2)$
using *eval-e-elims*(3)
using *sv* $\langle sv1 = SBool\ True\rangle$ **by** *metis*
moreover hence $n1 = n$ **using** *eval-e-elims*(1)[*of* i] *eval-v-elims*(2)[*of* i x $SNum\ n1$] i **by** *auto*
moreover have $n2 = int\ (length\ v)$ **using** *eval-e-elims*(7) *eval-v-elims*(1) *eval-l.simps* i
by (*metis* $***$ *eval-e-elims*(1) *rcl-val.eq-iff*(1) *rcl-val.eq-iff*(2))
ultimately have *le2*: $n \leq int\ (length\ v)$ **by** *simp*

show *?thesis* **using** *le1 le2* **by** *auto*
qed

thm *wfI-def*

lemma *wfI-cons*:
fixes i::*valuation* **and** Γ::Γ
assumes $i' \models \Gamma$ **and** $\Theta\ ;\ \Gamma \vdash i'$ **and** $i = i'\ (\ x \mapsto s)$ **and** $\Theta\ \vdash s : b$ **and** $atom\ x \sharp \Gamma$

```
  shows Θ ; (x,b,c) #Γ Γ ⊢ i
unfolding wfI-def proof −
  {
    fix x′ b′ c′
    assume (x′,b′,c′) ∈ setG ((x, b, c) #Γ Γ)
    then consider (x′,b′,c′) = (x,b,c) | (x′,b′,c′) ∈ setG Γ using setG.simps by auto
    then have ∃ s. Some s = i x′ ∧  Θ  ⊢ s : b′ proof(cases)
      case 1
      then show ?thesis using assms by auto
    next
      case 2
      then obtain s where s:Some s = i′ x′ ∧ Θ  ⊢ s : b′ using assms wfI-def by auto
      moreover have x′ ≠ x using assms 2 fresh-dom-free by auto
      ultimately have Some s = i x′ using assms by auto
      then show ?thesis using s  wfI-def by auto
    qed
  }
  thus ∀(x, b, c)∈setG ((x, b, c) #Γ Γ). ∃ s. Some s = i x ∧  Θ  ⊢ s : b by auto
qed


lemma valid-range-length-inv:
  fixes Γ::Γ
  assumes  Θ ; {||} ⊢wf Γ  and atom x ♯ Γ and ∃ i. i ⊨ Γ ∧ Θ ; Γ ⊢ i
  and  Θ ; {||} ; (x, B-int  , (C-eq (CE-val (V-var x)) (CE-val (V-lit (L-num n))))) #Γ  Γ ⊨
                  (C-eq (CE-op LEq (CE-val (V-lit (L-num 0))) (CE-val (V-var x)))  [[ L-true ]v ]ce)
AND
                  (C-eq (CE-op LEq (CE-val (V-var x)) ([|| [ [ L-bitvec v ]v ]ce ||]ce ))  [[ L-true ]v ]ce)

      (is Θ ; {||} ; ?G ⊨ ?c1 AND ?c2)
    shows 0 ≤ n ∧ n ≤ int (length v)
proof −
  have ∗:∀ i.  Θ ; ?G ⊢ i ∧  i ⊨ ?G  ⟶ i ⊨ ?c1 AND ?c2 using assms valid.simps by simp

  obtain i′ where idash: is-satis-g i′ Γ ∧ Θ ; Γ ⊢ i′ using assms by auto
  obtain i where i: i = i′ ( x ↦ SNum n) by auto
  hence ix: i x = Some (SNum n) by auto
  have Θ ; ?G ⊢ i ∧  i ⊨ ?G proof
    show  Θ ; ?G ⊢ i using wfI-cons i idash ix wfRCV-BIntI assms by simp

    have ∗∗:i ⟦ ([ [ x ]v ]ce  ==  [ [ L-num n ]v ]ce ) ⟧ ~ True
     using ∗ eval-c.intros(7) eval-e.intros eval-v.intros  eval-l.simps i
     by (metis (full-types) ix)

   show  i ⊨ ?G unfolding  is-satis-g.simps proof
     show ‹ i ⊨ [ [ x ]v ]ce  ==  [ [ L-num n ]v ]ce › using ∗∗ is-satis.simps by auto
     show ‹ i ⊨ Γ › using idash i assms is-satis-g-i-upd by metis
qed
  qed
  hence ∗∗:i ⊨ ?c1 AND ?c2 using ∗ by auto

  hence  1: i ⟦ ?c1 ⟧ ~ True using eval-c-elims(3) is-satis.simps
```

by *fastforce*
then obtain *sv1* **and** *sv2* **where** $(sv1 = sv2) = True \wedge i \llbracket [\, leq\, [\, [\, L\text{-}num\ 0\,]^v\,]^{ce}\, [\, [\, x\,]^v\,]^{ce}\,]^{ce} \rrbracket \sim sv1 \wedge i \llbracket [\, [\, L\text{-}true\,]^v\,]^{ce} \rrbracket \sim sv2$
using *eval-c-elims*(7) **by** *metis*
hence $sv1 = SBool\ True$ **using** *eval-e-elims eval-v-elims eval-l.simps i* **by** *metis*
obtain *n1* **and** *n2* **where** $SBool\ True = SBool\ (n1 \leq n2) \wedge (i \llbracket [\, [\, L\text{-}num\ 0\,]^v\,]^{ce} \rrbracket \sim SNum\ n1) \wedge (i \llbracket [\, [\, x\,]^v\,]^{ce} \rrbracket \sim SNum\ n2)$
using *eval-e-elims*(3)[*of i* $[\, [\, L\text{-}num\ 0\,]^v\,]^{ce}\, [\, [\, x\,]^v\,]^{ce}\ \ SBool\ True$]
using ‹$(sv1 = sv2) = True \wedge i \llbracket [\, leq\, [\, [\, L\text{-}num\ 0\,]^v\,]^{ce}\, [\, [\, x\,]^v\,]^{ce}\,]^{ce} \rrbracket \sim sv1 \wedge i \llbracket [\, [\, L\text{-}true\,]^v\,]^{ce} \rrbracket \sim sv2$› ‹$sv1 = SBool\ True$› **by** *fastforce*
moreover hence $n1 = 0$ **and** $n2 = n$ **using** *eval-e-elims eval-v-elims i*
apply (*metis eval-l.simps*(3) *rcl-val.eq-iff*(2))
using *eval-e-elims eval-v-elims i*
calculation option.inject rcl-val.eq-iff(2)
by (*metis ix*)
ultimately have *le1*: $0 \leq n$ **by** *simp*

hence *2*: $i \llbracket ?c2 \rrbracket \sim True$ **using** ** *eval-c-elims*(3) *is-satis.simps*
by *fastforce*
then obtain *sv1* **and** *sv2* **where** *sv*: $(sv1 = sv2) = True \wedge i \llbracket [\, leq\, [\, [\, x\,]^v\,]^{ce}\ \ [|\, [\, [\, L\text{-}bitvec\ v\,]^v\,]^{ce}\, |]^{ce}\,]^{ce}\ \ \rrbracket \sim sv1 \wedge i \llbracket [\, [\, L\text{-}true\,]^v\,]^{ce}\ \rrbracket\ \sim sv2$
using *eval-c-elims*(7) **by** *metis*
hence $sv1 = SBool\ True$ **using** *eval-e-elims eval-v-elims eval-l.simps i* **by** *metis*
obtain *n1* **and** *n2* **where** ***:$SBool\ True = SBool\ (n1 \leq n2) \wedge\ (i \llbracket [\, [\, x\,]^v\,]^{ce} \rrbracket \sim SNum\ n1) \wedge (i \llbracket [|\, [\, [\, L\text{-}bitvec\ v\,]^v\,]^{ce}\, |]^{ce}\ \rrbracket \sim SNum\ n2)$
using *eval-e-elims*(3)
using *sv* ‹$sv1 = SBool\ True$› **by** *metis*
moreover hence $n1 = n$ **using** *eval-e-elims*(1)[*of i*] *eval-v-elims*(2)[*of i x SNum n1*] *i* **by** *auto*
moreover have $n2 = int\ (length\ v)$ **using** *eval-e-elims*(7) *eval-v-elims*(1) *eval-l.simps i*
by (*metis* *** *eval-e-elims*(1) *rcl-val.eq-iff*(1) *rcl-val.eq-iff*(2))
ultimately have *le2*: $n \leq int\ (length\ v)$ **by** *simp*

show *?thesis* **using** *le1 le2* **by** *auto*
qed

lemma *eval-c-conj2I*[*intro*]:
assumes $i \llbracket c1 \rrbracket \sim True$ **and** $i \llbracket c2 \rrbracket \sim True$
shows $i \llbracket (C\text{-}conj\ c1\ c2) \rrbracket \sim\ True$
using *assms eval-c-conjI* **by** *metis*

lemma *valid-split*:
assumes *split n v* (*v1*,*v2*) **and** $\vdash_{wf} \Theta$
shows $\Theta\ ;\ \{||\}\ ;\ (z\ ,\ [B\text{-}bitvec\ ,\ B\text{-}bitvec\ ]^b\ ,\ [\, [\, z\,]^v\,]^{ce}\ ==\ [\, [\, [\, L\text{-}bitvec\ \ v1\,]^v\ ,\ [\, L\text{-}bitvec\ \ v2\,]^v\,]^v\,]^{ce})\ \#_\Gamma\ GNil$
$\models$ $([\, [\, L\text{-}bitvec\ v\,]^v\,]^{ce}\ ==\ [\, [\#1[\, [\, z\,]^v\,]^{ce}]^{ce}$ @@ $[\#2[\, [\, z\,]^v\,]^{ce}]^{ce}\,]^{ce})\ \ AND\ \ ([|\, [\#1[\, [\, z\,]^v\,]^{ce}]^{ce}\ |]^{ce}\ ==\ [\, [\, L\text{-}num\ n\,]^v\,]^{ce})$
(**is** $\Theta\ ;\ \{||\}\ ;\ ?G \models ?c1\ AND\ ?c2$)
unfolding *valid.simps* **proof**

have *wfg*: $\Theta\ ;\ \{||\}\ \vdash_{wf} (z, [B\text{-}bitvec\ ,\ B\text{-}bitvec\ ]^b\ ,\ [\, [\, z\,]^v\,]^{ce}\ ==\ [\, [\, [\, L\text{-}bitvec\ \ v1\,]^v\ ,\ [\, L\text{-}bitvec$

$v2\ ]^v\ ]^v\ ]^{ce})\ \#_\Gamma\ GNil$
using *wf-intros assms base-for-lit.simps fresh-GNil wfC-v-eq wfG-intros2* **by** *metis*

show $\Theta\ ;\ \{||\}\ ;\ ?G\ \vdash_{wf}\ ?c1\ AND\ ?c2$

apply(*rule wfC-conjI*)
apply(*rule wfC-eqI*)
apply(*rule wfCE-valI*)
apply(*rule wfV-litI*)
using *wf-intros wfg lookup.simps base-for-lit.simps wfC-v-eq*
apply (*metis*)+
done

have *len*:*int* (*length v1*) = *n* **using** *assms split-length* **by** *auto*

show $\forall\, i.\ \Theta\ ;\ ?G \vdash i \wedge i \models ?G \longrightarrow i \models (?c1\ AND\ ?c2)$
proof(*rule*,*rule*)
fix *i*
assume *a*:$\Theta\ ;\ ?G \vdash i \wedge i \models ?G$
hence $i\ [\![\ [\ [\ z\ ]^v\ ]^{ce}\ ==\ [\ [\ [\ L\text{-}bitvec\ v1\ ]^v\ ,\ [\ L\text{-}bitvec\ v2\ ]^v\ ]^v\ ]^{ce}\ ]\!] \sim True$
using *is-satis-g.simps is-satis.simps* **by** *simp*
then obtain *sv* **where** $i\ [\![\ [\ [\ z\ ]^v\ ]^{ce}\ ]\!] \sim sv \wedge i\ [\![\ [\ [\ [\ L\text{-}bitvec\ v1\ ]^v\ ,\ [\ L\text{-}bitvec\ v2\ ]^v\ ]^v\ ]^{ce}\ ]\!] \sim sv$
using *eval-c-elims* **by** *metis*
hence $i\ [\![\ [\ [\ z\ ]^v\ ]^{ce}\ ]\!] \sim (SPair\ (SBitvec\ v1)\ (SBitvec\ v2))$ **using** *eval-c-eqI eval-v.intros eval-l.simps*

by (*metis eval-e-elims*(*1*) *eval-v-uniqueness*)
hence *b*:$i\ z = Some\ (SPair\ (SBitvec\ v1)\ (SBitvec\ v2))$ **using** *a eval-e-elims eval-v-elims* **by** *metis*

have *v1*: $i\ [\![\ [\#1[\ [\ z\ ]^v\ ]^{ce}]^{ce}\ ]\!] \sim SBitvec\ v1$
using *eval-e-fstI eval-e-valI eval-v-varI b* **by** *metis*
have *v2*: $i\ [\![\ [\#2[\ [\ z\ ]^v\ ]^{ce}]^{ce}\ ]\!] \sim SBitvec\ v2$
using *eval-e-sndI eval-e-valI eval-v-varI b* **by** *metis*

have $i\ [\![\ [\ [\ L\text{-}bitvec\ v\ ]^v\ ]^{ce}\ ]\!] \sim SBitvec\ v$ **using** *eval-e.intros eval-v.intros eval-l.simps* **by** *metis*
moreover have $i\ [\![\ [\ [\#1[\ [\ z\ ]^v\ ]^{ce}]^{ce}\ @@\ [\#2[\ [\ z\ ]^v\ ]^{ce}]^{ce}\ ]^{ce}\ ]\!] \sim SBitvec\ v$
using *assms split-concat v1 v2 eval-e-concatI* **by** *metis*
moreover have $i\ [\![\ [|\ [\#1[\ [\ z\ ]^v\ ]^{ce}]^{ce}\ |]^{ce}\ ]\!] \sim SNum\ (int\ (length\ v1))$
using *v1 eval-e-lenI* **by** *auto*
moreover have $i\ [\![\ [\ [\ L\text{-}num\ n\ ]^v\ ]^{ce}\ ]\!] \sim SNum\ n$ **using** *eval-e.intros eval-v.intros eval-l.simps*
by *metis*
ultimately show $i \models ?c1\ AND\ ?c2$ **using** *is-satis.intros eval-c-conj2I eval-c-eqI len* **by** *metis*
qed
qed

end

lemma *wfT-restrict2*:
fixes τ::τ

assumes *wfT* Θ $\mathcal{B}$ $((x,\ b,\ c)\ \#_\Gamma\ \Gamma)$ τ **and** *atom* x $\sharp$ τ
shows Θ ; $\mathcal{B}$; $\Gamma \vdash_{wf} \tau$
using *wf-restrict1*(4)[*of* Θ $\mathcal{B}$ $((x,\ b,\ c)\ \#_\Gamma\ \Gamma)$ τ *GNil x b c* Γ] *assms fresh-GNil append-g.simps* **by** *auto*

Chapter 12

Typing Lemmas

12.1 Subtyping

lemma *subtype-reflI2*:
fixes τ::τ
assumes Θ ; $\mathcal{B}$; $\Gamma \vdash_{wf} \tau$
shows Θ ; $\mathcal{B}$; $\Gamma \vdash \tau \lesssim \tau$
proof −
obtain *z b c* **where** $*$:τ = $\{\!|$ *z* : *b* | *c* $|\!\}$ $\wedge$ *atom z* $\sharp$ (Θ,$\mathcal{B}$,Γ) $\wedge$ Θ ; $\mathcal{B}$; (*z*, *b*, *TRUE*) $\#_\Gamma$ Γ $\vdash_{wf}$ *c*
using *wfT-elims(1)*[*OF assms*] **by** *metis*
obtain *x*::*x* **where** $**$: *atom x* $\sharp$ (Θ, $\mathcal{B}$, Γ, *c*, *z* ,*c*, *z* , *c*) **using** *obtain-fresh* **by** *metis*
have Θ ; $\mathcal{B}$; $\Gamma \vdash$ $\{\!|$ *z* : *b* | *c* $|\!\}$ $\lesssim$ $\{\!|$ *z* : *b* | *c* $|\!\}$ **proof**
show Θ ; $\mathcal{B}$; Γ $\vdash_{wf}$ $\{\!|$ *z* : *b* | *c* $|\!\}$ **using** $*$ *assms* **by** *auto*
show Θ ; $\mathcal{B}$; Γ $\vdash_{wf}$ $\{\!|$ *z* : *b* | *c* $|\!\}$ **using** $*$ *assms* **by** *auto*
show *atom x* $\sharp$ (Θ, $\mathcal{B}$, Γ, *z* , *c* , *z* , *c*) **using** *fresh-prod6 fresh-prod5* $**$ **by** *metis*
thus Θ ; $\mathcal{B}$; (*x*, *b*, *c*[*z*::=*V-var x*]$_v$) $\#_\Gamma$ Γ $\models$ *c*[*z*::=*V-var x*]$_v$ **using** *wfT-wfC-cons assms* $*$ $**$ *subst-v-c-def* **by** *simp*
qed
thus *?thesis* **using** $*$ **by** *auto*
qed

lemma *subtype-reflI*:
assumes $\{\!|$ *z1* : *b* | *c1* $|\!\}$ = $\{\!|$ *z2* : *b* | *c2* $|\!\}$ **and** *wf1*: Θ ; $\mathcal{B}$;$\Gamma \vdash_{wf}$ ($\{\!|$ *z1* : *b* | *c1* $|\!\}$)
shows Θ ; $\mathcal{B}$; $\Gamma \vdash$ ($\{\!|$ *z1* : *b* | *c1* $|\!\}$) $\lesssim$ ($\{\!|$ *z2* : *b* | *c2* $|\!\}$)
using *assms subtype-reflI2* **by** *metis*

nominal-function *base-eq* :: $\Gamma \Rightarrow \tau \Rightarrow \tau \Rightarrow$ *bool* **where**
base-eq - $\{\!|$ *z1* : *b1*| *c1* $|\!\}$ $\{\!|$ *z2* : *b2* | *c2* $|\!\}$ = (*b1* = *b2*)
apply(*auto*,*simp add*: *eqvt-def base-eq-graph-aux-def*)
by (*meson* τ.*exhaust*)
nominal-termination (*eqvt*) **by** *lexicographic-order*

lemma *subtype-wfT*:
fixes *t1*::τ **and** *t2*::τ
assumes Θ ; $\mathcal{B}$; $\Gamma \vdash$ *t1* $\lesssim$ *t2*
shows Θ ; $\mathcal{B}$; $\Gamma \vdash_{wf}$ *t1* $\wedge$ Θ ; $\mathcal{B}$; $\Gamma \vdash_{wf}$ *t2*

using *assms subtype-elims* **by** *metis*

lemma *subtype-eq-base*:
assumes Θ ; $\mathcal{B}$; $\Gamma \vdash$ ($\{\!|$ *z1* : *b1*| *c1* $|\!\}$) $\lesssim$ ($\{\!|$ *z2* : *b2* | *c2* $|\!\}$)
shows *b1*=*b2*
using *subtype.simps assms* **by** *auto*

lemma *subtype-eq-base2*:
assumes Θ ; $\mathcal{B}$; $\Gamma \vdash$ *t1* $\lesssim$ *t2*
shows *b-of t1* = *b-of t2*
using *assms* **proof**(*rule subtype.induct*[*of* Θ $\mathcal{B}$ Γ *t1 t2*],*goal-cases*)
case (*1* Θ Γ *z1 b c1 z2 c2 x*)
then show *?case* **using** *subtype-eq-base* **by** *auto*
qed

lemma *subtype-wf*:
fixes $\tau 1::\tau$ **and** $\tau 2::\tau$
assumes Θ ; $\mathcal{B}$; $\Gamma \vdash$ $\tau 1$ $\lesssim$ $\tau 2$
shows Θ ; $\mathcal{B}$; $\Gamma \vdash_{wf} \tau 1$ $\wedge\Theta$; $\mathcal{B}$; $\Gamma \vdash_{wf} \tau 2$
using *assms*
proof(*rule subtype.induct*[*of* Θ $\mathcal{B}$ Γ $\tau 1$ $\tau 2$],*goal-cases*)
case (*1* Θ ΓG *z1 b c1 z2 c2 x*)
then show *?case* **by** *blast*
qed

lemma *subtype-g-wf*:
fixes $\tau 1::\tau$ **and** $\tau 2::\tau$ **and** $\Gamma::\Gamma$
assumes Θ ; $\mathcal{B}$; $\Gamma \vdash$ $\tau 1$ $\lesssim$ $\tau 2$
shows Θ ; $\mathcal{B}\vdash_{wf} \Gamma$
using *assms*
proof(*rule subtype.induct*[*of* Θ $\mathcal{B}$ Γ $\tau 1$ $\tau 2$],*goal-cases*)
case (*1* Θ $\mathcal{B}$ Γ *z1 b c1 z2 c2 x*)
then show *?case* **using** *wfX-wfY* **by** *auto*
qed

For when we have a particular y that satisfies the freshness conditions that we want the validity check to use

lemma *valid-flip-simple*:
assumes Θ ; $\mathcal{B}$; (z, b, c) $\#_\Gamma$ Γ $\models c'$ **and** *atom z* $\sharp$ Γ **and** *atom x* $\sharp$ (z, c, z, c', Γ)
shows Θ ; $\mathcal{B}$; (x, b, ($z \leftrightarrow x$) $\cdot$ c) $\#_\Gamma$ Γ $\models$ ($z \leftrightarrow x$) $\cdot$ c'
proof −
have ($z \leftrightarrow x$) $\cdot$ Θ ; $\mathcal{B}$; ($z \leftrightarrow x$) $\cdot$ ((z, b, c) $\#_\Gamma$ Γ) $\models$ ($z \leftrightarrow x$) $\cdot$ c'
using *True-eqvt valid.eqvt assms beta-flip-eq wfX-wfY* **by** *metis*
moreover have $\vdash_{wf} \Theta$ **using** *valid.simps wfC-wf wfG-wf assms* **by** *metis*
ultimately show *?thesis*
using *theta-flip-eq G-cons-flip-fresh3*[*of x* Γ *z b c*] *assms fresh-Pair flip-commute* **by** *metis*
qed

lemma *valid-wf-all*:

assumes Θ ; $\mathcal{B}$; $(z0,b,c0)\#_\Gamma G \models c$
shows *wfG* Θ $\mathcal{B}$ G **and** *wfC* Θ $\mathcal{B}$ $((z0,b,c0)\#_\Gamma G)$ c **and** *atom z0* $\sharp$ G
using *valid.simps wfC-wf wfG-cons assms* **by** *metis+*

lemma *valid-wfT*:
fixes $z::x$
assumes Θ ; $\mathcal{B}$; $(z0,b,c0[z::=V\text{-}var\ z0]_v)\#_\Gamma G \models c[z::=V\text{-}var\ z0]_v$ **and** *atom z0* $\sharp$ $(\Theta, \mathcal{B}, G,c,c0)$
shows Θ ; $\mathcal{B}$; $G \vdash_{wf} \{\!|\ z : b \mid c0\ |\!\}$ **and** Θ ; $\mathcal{B}$; $G \vdash_{wf} \{\!|\ z : b \mid c\ |\!\}$
proof −
have *atom z0* $\sharp$ *c0* **using** *assms fresh-Pair* **by** *auto*
moreover have $*$: Θ ; $\mathcal{B} \vdash_{wf} (z0,b,c0[z::=V\text{-}var\ z0]_{cv})\#_\Gamma G$ **using** *valid-wf-all wfX-wfY assms subst-v-c-def* **by** *metis*
ultimately show *wft*: Θ ; $\mathcal{B}$; $G \vdash_{wf} \{\!|\ z : b \mid c0\ |\!\}$ **using** *wfG-wfT*[*OF* $*$] **by** *auto*

have *atom z0* $\sharp$ *c* **using** *assms fresh-Pair* **by** *auto*
moreover have *wfc*: Θ ; $\mathcal{B}$; $(z0,b,c0[z::=V\text{-}var\ z0]_v)\#_\Gamma G \vdash_{wf} c[z::=V\text{-}var\ z0]_v$ **using** *valid-wf-all assms* **by** *metis*
have Θ ; $\mathcal{B}$; $G \vdash_{wf} \{\!|\ z0 : b \mid c[z::=V\text{-}var\ z0]_v\ |\!\}$ **proof**
show ‹*atom z0* $\sharp$ $(\Theta, \mathcal{B}, G)$› **using** *assms fresh-prodN* **by** *simp*
show ‹ Θ ; $\mathcal{B} \vdash_{wf} b$ › **using** *wft wfT-wfB* **by** *force*
show ‹ Θ ; $\mathcal{B}$; $(z0,\ b,\ TRUE)\ \#_\Gamma\ G \vdash_{wf} c[z::=[\ z0\ ]^v]_v$ › **using** *wfc wfC-replace-inside*[*OF wfc, of GNil z0 b c0*[*z::=*[*z0*]^v]$_v$ *G C-true*] *wfC-trueI*
append-g.simps
by (*metis local.* wfG-elim2 wf-trans*(*2*))
qed
moreover have $\{\!|\ z0 : b \mid c[z::=V\text{-}var\ z0]_v\ |\!\} = \{\!|\ z : b \mid c\ |\!\}$ **using** ‹*atom z0* $\sharp$ *c0*› *τ.eq-iff Abs1-eq-iff*(*3*)
using *calculation*(*1*) *subst-v-c-def* **by** *auto*
ultimately show Θ ; $\mathcal{B}$; $G \vdash_{wf} \{\!|\ z : b \mid c\ |\!\}$ **by** *auto*
qed

lemma *valid-flip*:
fixes *c*::*c* **and** *z*::*x* **and** *z0*::*x* **and** *xx2*::*x*
assumes Θ ; $\mathcal{B}$; $(xx2,\ b,\ c0[z0::=V\text{-}var\ xx2]_v)\ \#_\Gamma\ \Gamma \models c[z::=V\text{-}var\ xx2]_v$ **and**
atom xx2 $\sharp$ $(c0,\Gamma,c,z)$ **and** *atom z0* $\sharp$ (Γ,c,z)
shows Θ ; $\mathcal{B}$; $(z0,\ b,\ c0)\ \#_\Gamma\ \Gamma \models c[z::=V\text{-}var\ z0]_v$
proof −
have $\vdash_{wf} \Theta$ **using** *assms valid-wf-all wfX-wfY* **by** *metis*
hence Θ ; $\mathcal{B}$; $(xx2 \leftrightarrow z0) \cdot ((xx2,\ b,\ c0[z0::=V\text{-}var\ xx2]_v)\ \#_\Gamma\ \Gamma) \models ((xx2 \leftrightarrow z0) \cdot c[z::=V\text{-}var\ xx2]_v)$
using *valid.eqvt True-eqvt assms beta-flip-eq theta-flip-eq* **by** *metis*
hence Θ ; $\mathcal{B}$; $(((xx2 \leftrightarrow z0) \cdot xx2,\ b,\ (xx2 \leftrightarrow z0) \cdot c0[z0::=V\text{-}var\ xx2]_v)\ \#_\Gamma\ (xx2 \leftrightarrow z0) \cdot \Gamma) \models ((xx2 \leftrightarrow z0) \cdot (c[z::=V\text{-}var\ xx2]_v))$
using *G-cons-flip*[*of xx2 z0 xx2 b c0*[*z0::=V-var xx2*]$_v$ Γ] **by** *auto*
moreover have $(xx2 \leftrightarrow z0) \cdot xx2 = z0$ **by** *simp*
moreover have $(xx2 \leftrightarrow z0) \cdot c0[z0::=V\text{-}var\ xx2]_v = c0$
using *assms subst-cv-v-flip*[*of xx2 c0 z0 V-var z0*] *assms fresh-prod4* **by** *auto*
moreover have $(xx2 \leftrightarrow z0) \cdot \Gamma = \Gamma$ **proof** −
have *atom xx2* $\sharp$ Γ **using** *assms* **by** *auto*
moreover have *atom z0* $\sharp$ Γ **using** *assms* **by** *auto*
ultimately show *?thesis* **using** *flip-fresh-fresh* **by** *auto*

qed
moreover have $(xx2 \leftrightarrow z0\ ) \cdot (c[z::=V\text{-}var\ xx2]_v) = c[z::=V\text{-}var\ z0]_v$ **using** *subst-cv-v-flip2*[*of xx2 z c z0*] *assms* **by** *force*
ultimately show *?thesis* **by** *auto*
qed

lemma *subtype-valid*:
assumes $\Theta\ ;\ \mathcal{B}\ ;\ \Gamma \vdash t1 \lesssim t2$ **and** *atom* $y \sharp \Gamma$ **and** $t1 = \{\!|\ z1 : b\ |\ c1\ |\!\}$ **and** $t2 = \{\!|\ z2 : b\ |\ c2\ |\!\}$
shows $\Theta\ ;\ \mathcal{B}\ ;\ ((y,\ b,\ c1[z1::=V\text{-}var\ y]_v)\ \#_\Gamma\ \Gamma) \models c2[z2::=V\text{-}var\ y]_v$
using *assms* **proof**(*nominal-induct t2 avoiding*: *y rule*: *subtype.strong-induct*)
case (*subtype-baseI x* Θ $\mathcal{B}$ Γ *z c z' c' ba*)

hence $(x \leftrightarrow y) \cdot \Theta\ ;\ (x \leftrightarrow y) \cdot \mathcal{B}\ ;\ (x \leftrightarrow y) \cdot ((x,\ ba,\ c[z::=[\ x\ ]^v]_v)\ \#_\Gamma\ \Gamma) \models (x \leftrightarrow y) \cdot c'[z'::=[\ x\ ]^v]_v$ **using** *valid.eqvt*
using *permute-boolI* **by** *blast*
moreover have $\vdash_{wf} \Theta$ **using** *valid.simps wfC-wf wfG-wf subtype-baseI* **by** *metis*
ultimately have $\Theta\ ;\ \mathcal{B}\ ;\ ((y,\ ba,\ (x \leftrightarrow y) \cdot c[z::=[\ x\ ]^v]_v)\ \#_\Gamma\ \Gamma) \models (x \leftrightarrow y) \cdot c'[z'::=[\ x\ ]^v]_v$
using *subtype-baseI theta-flip-eq beta-flip-eq* τ*.eq-iff G-cons-flip-fresh3*[*of y* Γ *x ba*] **by** (*metis flip-commute*)
moreover have $(x \leftrightarrow y) \cdot c[z::=[\ x\ ]^v]_v = c1[z1::=[\ y\ ]^v]_v$
by (*metis subtype-baseI permute-flip-cancel subst-cv-id subst-cv-v-flip3 subst-cv-var-flip type-eq-subst-eq wfT-fresh-c subst-v-c-def*)
moreover have $(x \leftrightarrow y) \cdot c'[z'::=[\ x\ ]^v]_v = c2[z2::=[\ y\ ]^v]_v$
by (*metis subtype-baseI permute-flip-cancel subst-cv-id subst-cv-v-flip3 subst-cv-var-flip type-eq-subst-eq wfT-fresh-c subst-v-c-def*)

ultimately show *?case* **using** *subtype-baseI* τ*.eq-iff* **by** *metis*
qed

lemma *subtype-valid-simple*:
assumes $\Theta\ ;\ \mathcal{B}\ ;\ \Gamma \vdash t1 \lesssim t2$ **and** *atom* $z \sharp \Gamma$ **and** $t1 = \{\!|\ z : b\ |\ c1\ |\!\}$ **and** $t2 = \{\!|\ z : b\ |\ c2\ |\!\}$
shows $\Theta\ ;\ \mathcal{B}\ ;\ ((z,\ b,\ c1)\ \#_\Gamma\ \Gamma) \models c2$
using *subst-v-c-def subst-v-id assms subtype-valid*[*OF assms*] **by** *simp*

lemma *obtain-for-t-with-fresh*:
assumes *atom* $x \sharp t$
shows $\exists\, b\ c.\ t = \{\!|\ x : b\ |\ c\ |\!\}$
proof −
obtain *z1 b1 c1* **where** $*$: $t = \{\!|\ z1 : b1\ |\ c1\ |\!\} \wedge$ *atom z1* $\sharp$ *t* **using** *obtain-fresh-z* **by** *metis*
then have $t = (x \leftrightarrow z1) \cdot t$ **using** *flip-fresh-fresh assms* **by** *metis*
also have ... $= \{\!|\ (x \leftrightarrow z1) \cdot z1 : (x \leftrightarrow z1) \cdot b1\ |\ (x \leftrightarrow z1) \cdot c1\ |\!\}$ **using** $*$ *assms* **by** *simp*
also have ... $= \{\!|\ x : b1\ |\ (x \leftrightarrow z1) \cdot c1\ |\!\}$ **using** $*$ *assms* **by** *auto*
finally show *?thesis* **by** *auto*
qed

lemma *subtype-trans*:
assumes $\Theta\ ;\ \mathcal{B}\ ;\ \Gamma \vdash \tau1 \lesssim \tau2$ **and** $\Theta\ ;\ \mathcal{B}\ ;\ \Gamma \vdash \tau2 \lesssim \tau3$
shows $\Theta\ ;\ \mathcal{B}\ ;\ \Gamma \vdash \tau1 \lesssim \tau3$
proof −

obtain $y::x$ **where** yf:*atom* $y \sharp (\Gamma,\ \tau 1\ ,\ \tau 2,\ \tau 3\ )$ **using** *obtain-fresh* **by** *metis*

obtain *c1* **and** *b1* **where** $t1{:}\tau 1 = \{\!|\ y : b1 \mid c1\ |\!\}$ **using** *obtain-for-t-with-fresh yf* **by** *force*
obtain *c2* **and** *b2* **where** $t2{:}\tau 2 = \{\!|\ y : b2 \mid c2\ |\!\}$ **using** *obtain-for-t-with-fresh yf* **by** *force*
obtain *c3* **and** *b3* **where** $t3{:}\tau 3 = \{\!|\ y : b3 \mid c3\ |\!\}$ **using** *obtain-for-t-with-fresh yf* **by** *force*

obtain $x::x$ **where** xf:*atom* $x \sharp (\Gamma,y,c1,c2,c3,\Theta, \mathcal{B}, \Gamma, y,\ c1\ ,\ c3)$ **using** *obtain-fresh* **by** *metis*
have *beq*: $b1 = b2 \wedge b2 = b3$ **using** *assms subtype-eq-base t1 t2 t3* **by** *simp*

have *vld1*: $\Theta\ ;\ \mathcal{B}\ ;\ ((x,\ b1,\ c1[y::=V\text{-}var\ x]_v)\ \#_\Gamma\ \Gamma) \models c2[y::=V\text{-}var\ x]_v$ **using** *subtype-valid fresh-prod5 assms t1 t2 xf beq* **by** *simp*
have *vld2*: $\Theta\ ;\ \mathcal{B}\ ;\ ((x,\ b1,\ c2[y::=V\text{-}var\ x]_v)\ \#_\Gamma\ \Gamma) \models c3[y::=V\text{-}var\ x]_v$ **using** *subtype-valid fresh-prod5 assms t3 t2 xf beq* **by** *simp*
thm *valid-trans*[**where** $z{=}x$ **and** $v{=}V\text{-}var\ x$]
have $\Theta\ ;\ \mathcal{B}\ ;\ ((x,\ b1,\ c1[y::=V\text{-}var\ x]_v)\ \#_\Gamma\ \Gamma) \models c3[y::=V\text{-}var\ x]_v$ **using** *valid-trans-2 vld1 vld2* **by** *metis*

moreover have $\Theta\ ;\ \mathcal{B}\ ;\ \Gamma \vdash_{wf} \{\!|\ y : b1 \mid c1\ |\!\}$ **using** *t1 subtype-wfT assms* **by** *simp*
moreover have $\Theta\ ;\ \mathcal{B}\ ;\ \Gamma \vdash_{wf} \{\!|\ y : b1 \mid c3\ |\!\}$ **using** *t3 beq subtype-wfT assms* **by** *simp*
moreover have *atom* $x \sharp (\Theta, \mathcal{B}, \Gamma, y\ ,\ c1\ ,\ y\ ,\ c3)$ **using** *xf fresh-prod5 fresh-prod10* **by** *simp*
ultimately have $\Theta\ ;\ \mathcal{B}\ ;\ \Gamma\ \vdash \{\!|\ y : b1\ \mid c1\ |\!\} \lesssim \{\!|\ y : b1\ \mid c3\ |\!\}$ **using** *beq subtype-baseI fresh-prod5* **by** *metis*
thus *?thesis* **using** *t1 t3 beq* **by** *simp*
qed

lemma *subtype-eq-e*:
assumes $\forall\ i\ s1\ s2\ G.\ wfG\ P\ \mathcal{B}\ G \wedge wfI\ P\ G\ i \wedge eval\text{-}e\ i\ e1\ s1 \wedge eval\text{-}e\ i\ e2\ s2 \longrightarrow s1 = s2$ **and** *atom z1* $\sharp$ *e1* **and** *atom z2* $\sharp$ *e2* **and** *atom z1* $\sharp\ \Gamma$ **and** *atom z2* $\sharp\ \Gamma$
and *wfCE P* $\mathcal{B}\ \Gamma$ *e1 b* **and** *wfCE P* $\mathcal{B}\ \Gamma$ *e2 b*
shows $P\ ;\ \mathcal{B}\ ;\ \Gamma\ \vdash \{\!|\ z1 : b\ \mid CE\text{-}val\ (V\text{-}var\ z1)\ ==\ e1\ |\!\} \lesssim (\{\!|\ z2 : b\ \mid CE\text{-}val\ (V\text{-}var\ z2)\ ==\ e2\ |\!\})$
proof −

have *wfCE P* $\mathcal{B}\ \Gamma$ *e1 b* **and** *wfCE P* $\mathcal{B}\ \Gamma$ *e2 b* **using** *assms* **by** *auto*

have *wst1*: *wfT P* $\mathcal{B}\ \Gamma$ $(\{\!|\ z1 : b\ \mid CE\text{-}val\ (V\text{-}var\ z1)\ ==\ e1\ |\!\})$
using *wfC-e-eq wfTI assms wfX-wfB wfG-fresh-x*
by (*simp add*: *wfT-e-eq*)

moreover have *wst2*:*wfT P* $\mathcal{B}\ \Gamma$ $(\{\!|\ z2 : b\ \mid CE\text{-}val\ (V\text{-}var\ z2)\ ==\ e2\ |\!\})$
using *wfC-e-eq wfX-wfB wfTI assms wfG-fresh-x*
by (*simp add*: *wfT-e-eq*)

moreover obtain $x::x$ **where** *xf*: *atom* $x \sharp (P,\ \mathcal{B}\ ,\ z1,\ CE\text{-}val\ (V\text{-}var\ z1)\ ==\ e1\ ,\ z2,\ CE\text{-}val\ (V\text{-}var\ z2)\ ==\ e2\ ,\ \Gamma)$ **using** *obtain-fresh* **by** *blast*
moreover have *vld*: $P\ ;\ \mathcal{B}\ ;\ (x,\ b,\ (CE\text{-}val\ (V\text{-}var\ z1)\ ==\ e1\ )[z1::=V\text{-}var\ x]_v)\ \#_\Gamma\ \Gamma\ \models (CE\text{-}val\ (V\text{-}var\ z2)\ ==\ e2\ )[z2::=V\text{-}var\ x]_v$ (**is** $P\ ;\ \mathcal{B}\ ;\ ?G \models ?c$)
proof −

have *wbg*: $P\ ;\ \mathcal{B} \vdash_{wf} ?G\ \wedge\ P\ ;\ \mathcal{B}\ \vdash_{wf} \Gamma \wedge\ setG\ \Gamma \subseteq setG\ ?G$ **proof** −
have $P\ ;\ \mathcal{B} \vdash_{wf} ?G$ **proof**(*rule wfG-consI*)

show P ; $\mathcal{B} \vdash_{wf} \Gamma$ **using** *assms wfX-wfY* **by** *metis*
show *atom* $x \sharp \Gamma$ **using** *xf* **by** *auto*
show P ; $\mathcal{B} \vdash_{wf} b$ **using** *assms*(6) *wfX-wfB* **by** *auto*
show P ; $\mathcal{B}$; $(x, b, TRUE) \#_\Gamma \Gamma \vdash_{wf} (CE\text{-}val\ (V\text{-}var\ z1) == e1)[z1::=V\text{-}var\ x]_v$
using *wfC-e-eq*[*OF assms*(6)] *wf-subst*(2)
by (*simp add*: ‹*atom* $x \sharp \Gamma$› *assms*(2) *subst-v-c-def*)
qed
moreover hence P ; $\mathcal{B} \vdash_{wf} \Gamma$ **using** *wfG-elims* **by** *metis*
ultimately show *?thesis* **using** *setG.simps* **by** *auto*
qed

have *wsc*: *wfC P* $\mathcal{B}$ *?G ?c* **proof** −
have *wfCE P* $\mathcal{B}$ *?G* (*CE-val* (*V-var x*)) *b* **proof**
show ‹ P ; $\mathcal{B}$; $(x, b, (CE\text{-}val\ (V\text{-}var\ z1) == e1)[z1::=V\text{-}var\ x]_v) \#_\Gamma \Gamma \vdash_{wf} V\text{-}var\ x : b$ ›
using *wfV-varI lookup.simps wbg* **by** *auto*
qed
moreover have *wfCE P* $\mathcal{B}$ *?G e2 b* **using** *wf-weakening assms wbg* **by** *metis*
ultimately have *wfC P* $\mathcal{B}$ *?G* (*CE-val* (*V-var x*) == *e2*) **using** *wfC-eqI* **by** *simp*
thus *?thesis* **using** *subst-cv.simps*(6) ‹*atom z2* $\sharp$ *e2*› *subst-v-c-def* **by** *simp*
qed

moreover have $\forall i.$ *wfI P ?G i* $\wedge$ *is-satis-g i ?G* $\longrightarrow$ *is-satis i ?c* **proof**(*rule allI* , *rule impI*)
fix *i*
assume *as*: *wfI P ?G i* $\wedge$ *is-satis-g i ?G*
hence *is-satis i* ((*CE-val* (*V-var z1*) == *e1*)[*z1*::=*V-var x*]$_v$)
by (*simp add*: *is-satis-g.simps*(2))
hence *is-satis i* (*CE-val* (*V-var x*) == *e1*) **using** *subst-cv.simps assms subst-v-c-def* **by** *auto*
then obtain *s1* **and** *s2* **where** ∗:*eval-e i* (*CE-val* (*V-var x*)) *s1* $\wedge$ *eval-e i e1 s2* $\wedge$ *s1*=*s2* **using** *is-satis.simps eval-c-elims* **by** *metis*
moreover hence *eval-e i e2 s1* **proof** −
have ∗∗:*wfI P ?G i* **using** *as* **by** *auto*
moreover have *wfCE P* $\mathcal{B}$ *?G e1 b* $\wedge$ *wfCE P* $\mathcal{B}$ *?G e2 b* **using** *assms xf wf-weakening wbg* **by** *metis*
moreover then obtain *s2'* **where** *eval-e i e2 s2'* **using** *assms wfI-wfCE-eval-e* ∗∗ **by** *metis*
ultimately show *?thesis* **using** ∗ *assms*(1) *wfX-wfY* **by** *metis*
qed
ultimately have *is-satis i* (*CE-val* (*V-var x*) == *e2*) **using** *is-satis.simps eval-c-eqI* **by** *force*
thus *is-satis i* ((*CE-val* (*V-var z2*) == *e2*)[*z2*::=*V-var x*]$_v$) **using** *is-satis.simps eval-c-eqI assms subst-cv.simps subst-v-c-def* **by** *auto*
qed
ultimately show *?thesis* **using** *valid.simps* **by** *simp*
qed
moreover have *atom x* $\sharp$ (P, $\mathcal{B}$, Γ, *z1* , *CE-val* (*V-var z1*) == *e1*, *z2*, *CE-val* (*V-var z2*) == *e2*)
unfolding *fresh-prodN* **using** *xf fresh-prod7* τ*.fresh* **by** *fast*
ultimately show *?thesis* **using** *subtype-baseI*[*OF - wst1 wst2 vld*] *xf* **by** *simp*
qed

lemma *subtype-eq-e-nil*:
assumes $\forall i\ s1\ s2\ G.$ *wfG P* $\mathcal{B}$ *G* $\wedge$ *wfI P G i* $\wedge$ *eval-e i e1 s1* $\wedge$ *eval-e i e2 s2* $\longrightarrow$ *s1* = *s2* **and** *supp e1* = {} **and** *supp e2* = {} **and** *wfTh P*
and *wfCE P* $\mathcal{B}$ *GNil e1 b* **and** *wfCE P* $\mathcal{B}$ *GNil e2 b* **and** *atom z1* $\sharp$ *GNil* **and** *atom z2* $\sharp$ *GNil*

shows $P\ ;\ \mathcal{B}\ ;\ GNil\ \vdash\ \{\!|\ z1 : b\ |\ CE\text{-}val\ (V\text{-}var\ z1)\ ==\ e1\ |\!\}\ \lesssim\ (\{\!|\ z2 : b\ |\ CE\text{-}val\ (V\text{-}var\ z2)\ ==\ e2\ |\!\})$
apply(*rule subtype-eq-e*,*auto simp add*: *assms e.fresh*)
using *assms fresh-def e.fresh supp-GNil* **apply** *metis*+
done

lemma *subtype-gnil-fst-aux*:
assumes $supp\ v_1 = \{\}$ **and** $supp\ (V\text{-}pair\ v_1\ v_2) = \{\}$ **and** *wfTh P* **and** *wfCE P* $\mathcal{B}$ *GNil* (*CE-val* v_1) *b* **and** *wfCE P* $\mathcal{B}$ *GNil* (*CE-fst* $[V\text{-}pair\ v_1\ v_2]^{ce}$) *b* **and**
wfCE P $\mathcal{B}$ *GNil* (*CE-val* v_2) *b2* **and** *atom z1* $\sharp$ *GNil* **and** *atom z2* $\sharp$ *GNil*
shows $P\ ;\ \mathcal{B}\ ;\ GNil\ \vdash\ (\{\!|\ z1 : b\ |\ CE\text{-}val\ (V\text{-}var\ z1)\ ==\ CE\text{-}val\ v_1\ |\!\})\ \lesssim\ (\{\!|\ z2 : b\ |\ CE\text{-}val\ (V\text{-}var\ z2)\ ==\ CE\text{-}fst\ [V\text{-}pair\ v_1\ v_2]^{ce}\ |\!\})$
proof −
have $\forall\ i\ s1\ s2\ G\ .\ wfG\ P\ \mathcal{B}\ G \wedge wfI\ P\ G\ i \wedge eval\text{-}e\ i\ (\ CE\text{-}val\ v_1)\ s1 \wedge eval\text{-}e\ i\ (CE\text{-}fst\ [V\text{-}pair\ v_1\ v_2]^{ce})\ s2 \longrightarrow s1 = s2$ **proof**(*rule+*)
fix *i s1 s2 G*
assume *as*: *wfG P* $\mathcal{B}$ *G* $\wedge$ *wfI P G i* $\wedge$ *eval-e i* (*CE-val* v_1) *s1* $\wedge$ *eval-e i* (*CE-fst* $[V\text{-}pair\ v_1\ v_2]^{ce}$) *s2*
hence *wfCE P* $\mathcal{B}$ *G* (*CE-val* v_2) *b2* **using** *assms wf-weakening*
by (*metis empty-subsetI setG.simps*(*1*))
then obtain *s3* **where** *eval-e i* (*CE-val* v_2) *s3* **using** *wfI-wfCE-eval-e as* **by** *metis*
hence *eval-v i* $((V\text{-}pair\ v_1\ v_2))$ (*SPair s1 s3*)
by (*meson as eval-e-elims*(*1*) *eval-v-pairI*)
hence *eval-e i* (*CE-fst* $[V\text{-}pair\ v_1\ v_2]^{ce}$) *s1* **using** *eval-e-fstI eval-e-valI* **by** *metis*
show *s1 = s2* **using** *as eval-e-uniqueness*
using ‹*eval-e i* (*CE-fst* $[V\text{-}pair\ v_1\ v_2]^{ce}$) *s1*› **by** *auto*
qed
thus *?thesis* **using** *subtype-eq-e-nil ce.supp assms* **by** *auto*
qed

lemma *subtype-gnil-snd-aux*:
assumes $supp\ v_2 = \{\}$ **and** $supp\ (V\text{-}pair\ v_1\ v_2) = \{\}$ **and** *wfTh P* **and** *wfCE P* $\mathcal{B}$ *GNil* (*CE-val* v_2) *b* **and**
wfCE P $\mathcal{B}$ *GNil* (*CE-snd* $[(V\text{-}pair\ v_1\ v_2)]^{ce}$) *b* **and**
wfCE P $\mathcal{B}$ *GNil* (*CE-val* v_1) *b2* **and** *atom z1* $\sharp$ *GNil* **and** *atom z2* $\sharp$ *GNil*
shows $P\ ;\ \mathcal{B}\ ;\ GNil\ \vdash\ (\{\!|\ z1 : b\ |\ CE\text{-}val\ (V\text{-}var\ z1)\ ==\ CE\text{-}val\ v_2\ |\!\})\ \lesssim\ (\{\!|\ z2 : b\ |\ CE\text{-}val\ (V\text{-}var\ z2)\ ==\ CE\text{-}snd\ [(V\text{-}pair\ v_1\ v_2)]^{ce}\ |\!\})$
proof −
have $\forall\ i\ s1\ s2\ G.\ wfG\ P\ \mathcal{B}\ G \wedge wfI\ P\ G\ i \wedge eval\text{-}e\ i\ (\ CE\text{-}val\ v_2)\ s1 \wedge eval\text{-}e\ i\ (CE\text{-}snd\ [(V\text{-}pair\ v_1\ v_2)]^{ce})\ s2 \longrightarrow s1 = s2$ **proof**(*rule+*)
fix *i s1 s2 G*
assume *as*: *wfG P* $\mathcal{B}$ *G* $\wedge$ *wfI P G i* $\wedge$ *eval-e i* (*CE-val* v_2) *s1* $\wedge$ *eval-e i* (*CE-snd* $[(V\text{-}pair\ v_1\ v_2)]^{ce}$) *s2*
hence *wfCE P* $\mathcal{B}$ *G* (*CE-val* v_1) *b2* **using** *assms wf-weakening*
by (*metis empty-subsetI setG.simps*(*1*))
then obtain *s3* **where** *eval-e i* (*CE-val* v_1) *s3* **using** *wfI-wfCE-eval-e as* **by** *metis*
hence *eval-v i* $((V\text{-}pair\ v_1\ v_2))$ (*SPair s3 s1*)
by (*meson as eval-e-elims eval-v-pairI*)
hence *eval-e i* (*CE-snd* $[(V\text{-}pair\ v_1\ v_2)]^{ce}$) *s1* **using** *eval-e-sndI eval-e-valI* **by** *metis*

show *s1 = s2* **using** *as eval-e-uniqueness*
using ‹*eval-e i* (*CE-snd* [*V-pair* v_1 v_2]^{ce}) *s1*› **by** *auto*
qed
thus *?thesis* **using** *assms subtype-eq-e-nil* **by** (*simp add*: *ce.supp ce.supp*)
qed

lemma *subtype-gnil-fst*:
assumes Θ ; {||} ; *GNil* ⊢$_{wf}$ [#*1*[[v_1,v_2]^v]^{ce}]^{ce} : *b*
shows Θ ; {||} ; *GNil* ⊢ ({| z_1 : *b* | [[z_1]^v]^{ce} == [v_1]^{ce} |}) ≲
({| z_2 : *b* | [[z_2]^v]^{ce} == [#*1*[[v_1, v_2]^v]^{ce}]^{ce} |})
proof −
obtain *b2* **where** ∗∗: Θ ; {||} ; *GNil* ⊢$_{wf}$ *V-pair* v_1 v_2 : *B-pair b b2* **using** *wfCE-elims*(*4*)[*OF assms*] *wfCE-elims* **by** *metis*
obtain *b1' b2'* **where** ∗:*B-pair b b2* = *B-pair b1' b2'* ∧ Θ ; {||} ; *GNil* ⊢$_{wf}$ v_1 : *b1'* ∧ Θ ; {||} ; *GNil* ⊢$_{wf}$ v_2 : *b2'*
using *wfV-elims*(*3*)[*OF* ∗∗] **by** *metis*

show *?thesis* **proof**(*rule subtype-gnil-fst-aux*)
show ‹*supp* v_1 = {}› **using** ∗ *wfV-supp-nil* **by** *auto*
show ‹*supp* (*V-pair* v_1 v_2) = {}› **using** ∗∗ *wfV-supp-nil e.supp* **by** *metis*
show ‹⊢$_{wf}$ Θ› **using** *assms wfX-wfY* **by** *metis*
show ‹Θ; {||}; *GNil* ⊢$_{wf}$ *CE-val* v_1 : *b* › **using** *wfCE-valI* ∗ **by** *auto*
show ‹Θ; {||}; *GNil* ⊢$_{wf}$ *CE-fst* [*V-pair* v_1 v_2]^{ce} : *b* › **using** *assms* **by** *auto*
show ‹Θ; {||}; *GNil* ⊢$_{wf}$ *CE-val* v_2 : *b2* ›**using** *wfCE-valI* ∗ **by** *auto*
show ‹*atom* z_1 ♯ *GNil*› **using** *fresh-GNil* **by** *metis*
show ‹*atom* z_2 ♯ *GNil*› **using** *fresh-GNil* **by** *metis*
qed
qed

lemma *subtype-gnil-snd*:
assumes *wfCE P* {||} *GNil* (*CE-snd* ([*V-pair* v_1 v_2]^{ce})) *b*
shows *P* ; {||} ; *GNil* ⊢ ({| *z1* : *b* | *CE-val* (*V-var z1*) == *CE-val* v_2 |}) ≲ ({| *z2* : *b* | *CE-val* (*V-var z2*) == *CE-snd* [(*V-pair* v_1 v_2)]^{ce} |})
proof −
obtain *b1* **where** ∗∗: *P* ; {||} ; *GNil* ⊢$_{wf}$ *V-pair* v_1 v_2 : *B-pair b1 b* **using** *wfCE-elims assms* **by** *metis*
obtain *b1' b2'* **where** ∗:*B-pair b1 b* = *B-pair b1' b2'* ∧ *P* ; {||} ; *GNil* ⊢$_{wf}$ v_1 : *b1'* ∧ *P* ; {||} ; *GNil* ⊢$_{wf}$ v_2 : *b2'* **using** *wfV-elims*(*3*)[*OF* ∗∗] **by** *metis*

show *?thesis* **proof**(*rule subtype-gnil-snd-aux*)
show ‹*supp* v_2 = {}› **using** ∗ *wfV-supp-nil* **by** *auto*
show ‹*supp* (*V-pair* v_1 v_2) = {}› **using** ∗∗ *wfV-supp-nil e.supp* **by** *metis*
show ‹⊢$_{wf}$ *P*› **using** *assms wfX-wfY* **by** *metis*
show ‹*P*; {||}; *GNil* ⊢$_{wf}$ *CE-val* v_1 : *b1* › **using** *wfCE-valI* ∗ **by** *simp*
show ‹*P*; {||}; *GNil* ⊢$_{wf}$ *CE-snd* [(*V-pair* v_1 v_2)]^{ce} : *b* › **using** *assms* **by** *auto*
show ‹*P*; {||}; *GNil* ⊢$_{wf}$ *CE-val* v_2 : *b* ›**using** *wfCE-valI* ∗ **by** *simp*
show ‹*atom z1* ♯ *GNil*› **using** *fresh-GNil* **by** *metis*
show ‹*atom z2* ♯ *GNil*› **using** *fresh-GNil* **by** *metis*
qed
qed

lemma *subtype-fresh-tau*:
fixes $x::x$
assumes *atom* $x \sharp t1$ **and** *atom* $x \sharp \Gamma$ **and** $P\ ;\ \mathcal{B}\ ;\ \Gamma\ \vdash t1 \lesssim t2$
shows *atom* $x \sharp t2$
proof −
have *wfg*: $P\ ;\ \mathcal{B} \vdash_{wf}\ \Gamma$ **using** *subtype-wf wfX-wfY assms* **by** *metis*
have *wft*: *wfT P* $\mathcal{B}$ Γ *t2* **using** *subtype-wf wfX-wfY assms* **by** *blast*
hence *supp t2* $\subseteq$ *atom-dom* $\Gamma \cup$ *supp* $\mathcal{B}$ **using** *wf-supp*
using *atom-dom.simps* **by** *auto*
moreover have *atom* $x \notin$ *atom-dom* Γ **using** ‹*atom* $x \sharp \Gamma$› *wfG-atoms-supp-eq wfg fresh-def* **by** *blast*
ultimately show *atom* $x \sharp t2$ **using** *fresh-def*
by (*metis Un-iff contra-subsetD x-not-in-b-set*)
qed

lemma *subtype-if-simp*:
assumes *wfT P* $\mathcal{B}$ *GNil* ({‖ *z1* : *b* | *CE-val* (*V-lit l*) == *CE-val* (*V-lit l*) *IMP* $c[z::=V\text{-}var\ z1]_v$ ‖}) **and**
wfT P $\mathcal{B}$ *GNil* ({‖ *z* : *b* | *c* ‖}) **and** *atom z1* $\sharp$ *c*
shows $P\ ;\ \mathcal{B}\ ;$ *GNil* $\vdash$ ({‖ *z1* : *b* | *CE-val* (*V-lit l*) == *CE-val* (*V-lit l*) *IMP* $c[z::=V\text{-}var\ z1]_v$ ‖}) $\lesssim$ ({‖ *z* : *b* | *c* ‖})
proof −
obtain $x::x$ **where** *xx*: *atom* $x \sharp$ (P , $\mathcal{B}$, *z1*, *CE-val* (*V-lit l*) == *CE-val* (*V-lit l*) *IMP* $c[z::=V\text{-}var\ z1]_v$, *z*, *c*, *GNil*) **using** *obtain-fresh-z* **by** *blast*
hence *xx2*: *atom* $x \sharp$ (*CE-val* (*V-lit l*) == *CE-val* (*V-lit l*) *IMP* $c[z::=V\text{-}var\ z1]_v$, *c*, *GNil*) **using** *fresh-prod7 fresh-prod3* **by** *fast*
have $*$:$P\ ;\ \mathcal{B}\ ;$ (*x*, *b*, (*CE-val* (*V-lit l*) == *CE-val* (*V-lit l*) *IMP* $c[z::=V\text{-}var\ z1]_v$)$[z1::=V\text{-}var\ x]_v$) $\#_\Gamma$ *GNil* $\models c[z::=V\text{-}var\ x]_v$ (**is** $P\ ;\ \mathcal{B}\ ;\ ?G \models ?c$) **proof** −
have *wfC P* $\mathcal{B}$ *?G ?c* **using** *wfT-wfC-cons*[*OF assms*(*1*) *assms*(*2*),*of x*] *xx fresh-prod5 fresh-prod3 subst-v-c-def* **by** *metis*
moreover have ($\forall$ *i*. *wfI P ?G i* $\wedge$ *is-satis-g i ?G* $\longrightarrow$ *is-satis i ?c*) **proof**(*rule allI*, *rule impI*)
fix *i*
assume *as1*: *wfI P ?G i* $\wedge$ *is-satis-g i ?G*
have ((*CE-val* (*V-lit l*) == *CE-val* (*V-lit l*) *IMP* $c[z::=V\text{-}var\ z1]_v$)$[z1::=V\text{-}var\ x]_v$) = ((*CE-val* (*V-lit l*) == *CE-val* (*V-lit l*) *IMP* $c[z::=V\text{-}var\ x]_v$))
using *assms subst-v-c-def* **by** *auto*
hence *is-satis i* ((*CE-val* (*V-lit l*) == *CE-val* (*V-lit l*) *IMP* $c[z::=V\text{-}var\ x]_v$)) **using** *is-satis-g.simps as1* **by** *presburger*
moreover have *is-satis i* ((*CE-val* (*V-lit l*) == *CE-val* (*V-lit l*))) **using** *is-satis.simps eval-c-eqI*[*of i* (*CE-val* (*V-lit l*)) *eval-l l*] *eval-e-uniqueness*
eval-e-valI eval-v-litI **by** *metis*
ultimately show *is-satis i ?c* **using** *is-satis-mp*[*of i*] **by** *metis*
qed
ultimately show *?thesis* **using** *valid.simps* **by** *simp*
qed
moreover have *atom* $x \sharp$ (*P*, $\mathcal{B}$, *GNil*, *z1* , *CE-val* (*V-lit l*) == *CE-val* (*V-lit l*) *IMP* $c[z::=V\text{-}var\ z1]_v$, *z*, *c*)
unfolding *fresh-prod5* τ*.fresh* **using** *xx fresh-prodN x-fresh-b* **by** *metis*
ultimately show *?thesis* **using** *subtype-baseI assms xx xx2* **by** *metis*
qed

lemma *subtype-if*:
assumes $P\ ;\ \mathcal{B}\ ;\ \Gamma\ \vdash$ {‖ *z* : *b* | *c* ‖} $\lesssim$ {‖ *z'* : *b* | *c'* ‖} **and**

wfT P $\mathcal{B}$ Γ ({| *z1* : *b* | *CE-val v* == *CE-val* (*V-lit l*) *IMP* $c[z::=V\text{-}var\ z1]_v$ |}) **and**
wfT P $\mathcal{B}$ Γ ({| *z2* : *b* | *CE-val v* == *CE-val* (*V-lit l*) *IMP* $c'[z'::=V\text{-}var\ z2]_v$ |}) **and**
atom z1 $\sharp$ *v* **and** *atom z* $\sharp$ Γ **and** *atom z1* $\sharp$ *c* **and** *atom z2* $\sharp$ c' **and** *atom z2* $\sharp$ *v*

shows *P* ; $\mathcal{B}$; $\Gamma \vdash$ {| *z1* : *b* | *CE-val v* == *CE-val* (*V-lit l*) *IMP* $c[z::=V\text{-}var\ z1]_v$ |} $\lesssim$ {| *z2* : *b* | *CE-val v* == *CE-val* (*V-lit l*) *IMP* $c'[z'::=V\text{-}var\ z2]_v$ |}

proof −

obtain *x*::*x* **where** *xx*: *atom x* $\sharp$ (*P*,$\mathcal{B}$,*z*,*c*,*z'*, *c'*, *z1*, *CE-val v* == *CE-val* (*V-lit l*) *IMP* $c[z::=V\text{-}var\ z1]_v$, *z2*, *CE-val v* == *CE-val* (*V-lit l*) *IMP* $c'[z'::=V\text{-}var\ z2]_v$, Γ)
using *obtain-fresh-z* **by** *blast*

hence *xf*: *atom x* $\sharp$ (*z*, *c*, *z'*, *c'*, Γ) **by** *simp*

have *xf2*: *atom x* $\sharp$ (*z1*, *CE-val v* == *CE-val* (*V-lit l*) *IMP* $c[z::=V\text{-}var\ z1]_v$, *z2*, *CE-val v* == *CE-val* (*V-lit l*) *IMP* $c'[z'::=V\text{-}var\ z2]_v$, Γ)
using *xx fresh-prod4 fresh-prodN* **by** *metis*

moreover have *P* ; $\mathcal{B}$; (*x*, *b*, (*CE-val v* == *CE-val* (*V-lit l*) *IMP* $c[z::=V\text{-}var\ z1]_v$)[*z1*::=*V-var x*]$_v$) $\#_\Gamma$ Γ $\models$ (*CE-val v* == *CE-val* (*V-lit l*) *IMP* $c'[z'::=V\text{-}var\ z2]_v$)[*z2*::=*V-var x*]$_v$
(**is** *P* ; $\mathcal{B}$; *?G* $\models$ *?c*)

proof −

have *wbc*: *wfC P* $\mathcal{B}$ *?G ?c* **using** *assms xx fresh-prod4 fresh-prod2 wfT-wfC-cons assms subst-v-c-def* **by** *metis*

moreover have $\forall$ *i*. *wfI P ?G i* $\wedge$ *is-satis-g i ?G* $\longrightarrow$ *is-satis i ?c* **proof**(*rule allI*, *rule impI*)
fix *i*
assume *a1*: *wfI P ?G i* $\wedge$ *is-satis-g i ?G*
thm *is-satis.simps*
have $*$: *is-satis i* ((*CE-val v* == *CE-val* (*V-lit l*))) $\longrightarrow$ *is-satis i* ((*c'*[*z'*::=*V-var z2*]$_v$)[*z2*::=*V-var x*]$_v$)
proof
assume *a2*: *is-satis i* ((*CE-val v* == *CE-val* (*V-lit l*)))

have *is-satis i* ((*CE-val v* == *CE-val* (*V-lit l*) *IMP* (*c*[*z*::=*V-var z1*]$_v$))[*z1*::=*V-var x*]$_v$)
using *a1 is-satis-g.simps* **by** *simp*
moreover have ((*CE-val v* == *CE-val* (*V-lit l*) *IMP* (*c*[*z*::=*V-var z1*]$_v$))[*z1*::=*V-var x*]$_v$) = (*CE-val v* == *CE-val* (*V-lit l*) *IMP* ((*c*[*z*::=*V-var z1*]$_v$)[*z1*::=*V-var x*]$_v$))
using *assms subst-v-c-def* **by** *simp*
ultimately have *is-satis i* (*CE-val v* == *CE-val* (*V-lit l*) *IMP* ((*c*[*z*::=*V-var z1*]$_v$)[*z1*::=*V-var x*]$_v$)) **by** *argo*

hence *is-satis i* ((*c*[*z*::=*V-var z1*]$_v$)[*z1*::=*V-var x*]$_v$) **using** *a2 is-satis-mp* **by** *auto*
moreover have ((*c*[*z*::=*V-var z1*]$_v$)[*z1*::=*V-var x*]$_v$) = ((*c*[*z*::=*V-var x*]$_v$)) **using** *assms* **by** *auto*
ultimately have *is-satis i* ((*c*[*z*::=*V-var x*]$_v$)) **using** *a2 is-satis.simps* **by** *auto*

hence *is-satis-g i* ((*x*,*b*,(*c*[*z*::=*V-var x*]$_v$)) $\#_\Gamma$ Γ) **using** *a1 is-satis-g.simps* **by** *meson*
moreover have *wfI P* ((*x*,*b*,(*c*[*z*::=*V-var x*]$_v$)) $\#_\Gamma$ Γ) *i* **proof** −
obtain *s* **where** *Some s* = *i x* $\wedge$ *wfRCV P s b* $\wedge$ *wfI P* Γ *i* **using** *wfI-def a1* **by** *auto*
thus *?thesis* **using** *wfI-def* **by** *auto*
qed
ultimately have *is-satis i* ((*c'*[*z'*::=*V-var x*]$_v$)) **using** *subtype-valid assms*(*1*) *xf valid.simps* **by** *simp*

moreover have (*c'*[*z'*::=*V-var x*]$_v$) = ((*c'*[*z'*::=*V-var z2*]$_v$)[*z2*::=*V-var x*]$_v$) **using** *assms* **by**

auto
ultimately show *is-satis i* ((*c'*[*z'*::=*V-var z2*]$_v$)[*z2*::=*V-var x*]$_v$) **by** *auto*
qed

moreover have *?c* = ((*CE-val v* == *CE-val* (*V-lit l*)) *IMP* ((*c'*[*z'*::=*V-var z2*]$_v$)[*z2*::=*V-var x*]$_v$))
using *assms subst-v-c-def* **by** *simp*
thm *wfC-elims*
moreover have ∃ *b1 b2*. *eval-c i* (*CE-val v* == *CE-val* (*V-lit l*)) *b1* ∧
eval-c i c'[*z'*::=*V-var z2*]$_v$[*z2*::=*V-var x*]$_v$ *b2* **proof** −
thm *assms*(*2*)
have *wfC P* $\mathcal{B}$ *?G* (*CE-val v* == *CE-val* (*V-lit l*)) **using** *wbc wfC-elims*(*7*) *assms subst-cv.simps subst-v-c-def* **by** *fastforce*

moreover have *wfC P* $\mathcal{B}$ *?G* (*c'*[*z'*::=*V-var z2*]$_v$[*z2*::=*V-var x*]$_v$) **proof**(*rule wfT-wfC-cons*)
show ‹ *P* ; $\mathcal{B}$; Γ ⊢$_{wf}$ {‖ *z1* : *b* | *CE-val v* == *CE-val* (*V-lit l*) *IMP* (*c*[*z*::=*V-var z1*]$_v$) ‖} › **using** *assms subst-v-c-def* **by** *auto*
have {‖ *z2* : *b* | *c'*[*z'*::=*V-var z2*]$_v$ ‖} = {‖ *z'* : *b* | *c'* ‖} **using** *assms subst-v-c-def* **by** *auto*
thus ‹ *P* ; $\mathcal{B}$; Γ ⊢$_{wf}$ {‖ *z2* : *b* | *c'*[*z'*::=*V-var z2*]$_v$ ‖} › **using** *assms subtype-elims* **by** *metis*
show ‹*atom x* ♯ (*CE-val v* == *CE-val* (*V-lit l*) *IMP* *c*[*z*::=*V-var z1*]$_v$, *c'*[*z'*::=*V-var z2*]$_v$, Γ)› **using** *xx fresh-Pair c.fresh* **by** *metis*
qed

ultimately show *?thesis* **using** *wfI-wfC-eval-c a1 subst-v-c-def* **by** *simp*
qed

ultimately show *is-satis i ?c* **using** *is-satis-imp*[*OF* ∗] **by** *auto*
qed
ultimately show *?thesis* **using** *valid.simps* **by** *simp*
qed
moreover have *atom x* ♯ (*P*, $\mathcal{B}$, Γ, *z1* , *CE-val v* == *CE-val* (*V-lit l*) *IMP* *c*[*z*::=*V-var z1*]$_v$, *z2* , *CE-val v* == *CE-val* (*V-lit l*) *IMP* *c'*[*z'*::=*V-var z2*]$_v$)
unfolding *fresh-prod5* τ.*fresh* **using** *xx xf2 fresh-prodN x-fresh-b* **by** *metis*
ultimately show *?thesis* **using** *subtype-baseI assms xf2* **by** *metis*
qed

fun *single-g* :: *x*∗*b*∗*c* ⇒ Γ **where**
single-g xbc = *GCons xbc GNil*

lemma *eval-e-concat-eq*:
assumes *wfI* Θ Γ *i*
shows ∃ *s*. *eval-e i* (*CE-val* (*V-lit* (*L-bitvec* (*v1* @ *v2*)))) *s* ∧ *eval-e i* (*CE-concat* [(*V-lit* (*L-bitvec v1*))]^{ce} [(*V-lit* (*L-bitvec v2*))]^{ce}) *s*
using *eval-e-valI eval-e-concatI eval-v-litI eval-l.simps* **by** *metis*

lemma *is-satis-eval-e-eq-imp*:
assumes *wfI* Θ Γ *i* **and** *eval-e i e1 s* **and** *eval-e i e2 s*
and *is-satis i* (*CE-val* (*V-var x*) == *e1*) (**is** *is-satis i ?c1*)
shows *is-satis i* (*CE-val* (*V-var x*) == *e2*)
proof −
have ∗:*eval-c i ?c1 True* **using** *assms is-satis.simps* **by** *blast*
hence *eval-e i* (*CE-val* (*V-var x*)) *s* **using** *assms is-satis.simps eval-c-elims*

by (*metis* (*full-types*) *eval-e-uniqueness*)
thus *?thesis* **using** *is-satis.simps eval-c.intros assms* **by** *fastforce*
qed

lemma *valid-eval-e-eq*:
fixes *e1*::*ce* **and** *e2*::*ce*
assumes $\forall \Gamma\ i.$ *wfI* $\Theta\ \Gamma\ i \longrightarrow (\exists s.$ *eval-e i e1 s* $\wedge$ *eval-e i e2 s*) **and** Θ ; $\mathcal{B}$; *GNil* $\vdash_{wf}$ *e1* : *b* **and** Θ ; $\mathcal{B}$; *GNil* $\vdash_{wf}$ *e2* : *b*
shows Θ ; $\mathcal{B}$; $(x,\ b,$ (*CE-val* (*V-var x*) == *e1*)) $\#_\Gamma$ *GNil* $\models$ (*CE-val* (*V-var x*) == *e2*)
proof(*rule validI*)
show Θ ; $\mathcal{B}$; $(x,\ b,$ *CE-val* (*V-var x*) == *e1*) $\#_\Gamma$ *GNil* $\vdash_{wf}$ *CE-val* (*V-var x*) == *e2*
proof
have Θ ; $\mathcal{B}$; $(x,\ b,$ *TRUE* $)\#_\Gamma$ *GNil* $\vdash_{wf}$ *CE-val* (*V-var x*) == *e1* **using** *assms wfC-eqI wfE-valI wfV-varI wfX-wfY*
by (*simp add*: *fresh-GNil wfC-e-eq*)
hence Θ ; $\mathcal{B} \vdash_{wf} (x,\ b,$ *CE-val* (*V-var x*) == *e1*) $\#_\Gamma$ *GNil* **using** *wfG-consI fresh-GNil wfX-wfY assms wfX-wfB* **by** *metis*
thus Θ ; $\mathcal{B}$; $(x,\ b,$ *CE-val* (*V-var x*) == *e1*) $\#_\Gamma$ *GNil* $\vdash_{wf}$ *CE-val* (*V-var x*) : *b* **using** *wfCE-valI wfV-varI wfX-wfY*
lookup.simps assms wfX-wfY **by** *simp*
show Θ ; $\mathcal{B}$; $(x,\ b,$ *CE-val* (*V-var x*) == *e1*) $\#_\Gamma$ *GNil* $\vdash_{wf}$ *e2* : *b* **using** *assms wf-weakening wfX-wfY*
by (*metis* (*full-types*) ‹Θ ; $\mathcal{B}$; $(x,\ b,$ *CE-val* (*V-var x*) == *e1*) $\#_\Gamma$ *GNil* $\vdash_{wf}$ *CE-val* (*V-var x*) : *b*› *empty-iff subsetI setG.simps*(*1*))
qed
show $\forall i.$ *wfI* Θ $((x,\ b,$ *CE-val* (*V-var x*) == *e1*) $\#_\Gamma$ *GNil*) *i* $\wedge$ *is-satis-g i* $((x,\ b,$ *CE-val* (*V-var x*) == *e1*) $\#_\Gamma$ *GNil*) $\longrightarrow$ *is-satis i* (*CE-val* (*V-var x*) == *e2*)
proof(*rule*,*rule*)
fix *i*
assume *wfI* Θ $((x,\ b,$ *CE-val* (*V-var x*) == *e1*) $\#_\Gamma$ *GNil*) *i* $\wedge$ *is-satis-g i* $((x,\ b,$ *CE-val* (*V-var x*) == *e1*) $\#_\Gamma$ *GNil*)
moreover then obtain *s* **where** *eval-e i e1 s* $\wedge$ *eval-e i e2 s* **using** *assms* **by** *auto*
ultimately show *is-satis i* (*CE-val* (*V-var x*) == *e2*) **using** *assms is-satis-eval-e-eq-imp is-satis-g.simps* **by** *meson*
qed
qed

lemma *subtype-concat*:
assumes $\vdash_{wf} \Theta$
shows Θ ; $\mathcal{B}$; *GNil* $\vdash$ $\{\!|$ *z* : *B-bitvec* | *CE-val* (*V-var z*) == *CE-val* (*V-lit* (*L-bitvec* (*v1* @ *v2*))) $|\!\}$ $\lesssim$
$\{\!|$ *z* : *B-bitvec* | *CE-val* (*V-var z*) == *CE-concat* $[$(*V-lit* (*L-bitvec v1*))$]^{ce}$ $[$(*V-lit* (*L-bitvec v2*))$]^{ce}$ $|\!\}$ (**is** Θ ; $\mathcal{B}$; *GNil* $\vdash$ *?t1* $\lesssim$ *?t2*)
proof −
obtain *x*::*x* **where** *x*: *atom x* $\sharp$ (Θ, $\mathcal{B}$, *GNil*, *z* , *CE-val* (*V-var z*) == *CE-val* (*V-lit* (*L-bitvec* (*v1* @ *v2*))),
z , *CE-val* (*V-var z*) == *CE-concat* $[$*V-lit* (*L-bitvec v1*)$]^{ce}$ $[$*V-lit* (*L-bitvec v2*)$]^{ce}$)
(**is** *?xfree*)
using *obtain-fresh* **by** *auto*

have *wb1*: Θ ; $\mathcal{B}$; *GNil* $\vdash_{wf}$ *CE-val* (*V-lit* (*L-bitvec* (*v1* @ *v2*))) : *B-bitvec* **using** *wfX-wfY wfCE-valI wfV-litI assms base-for-lit.simps wfG-nilI* **by** *metis*
hence *wb2*: Θ ; $\mathcal{B}$; *GNil* $\vdash_{wf}$ *CE-concat* $[$(*V-lit* (*L-bitvec v1*))$]^{ce}$ $[$(*V-lit* (*L-bitvec v2*))$]^{ce}$: *B-bitvec*
using *wfCE-concatI wfX-wfY wfV-litI base-for-lit.simps wfCE-valI* **by** *metis*

show *?thesis* **proof**
show Θ ; $\mathcal{B}$; *GNil* $\vdash_{wf}$ *?t1* **using** *wfT-e-eq fresh-GNil wb1 wb2* **by** *metis*
show Θ ; $\mathcal{B}$; *GNil* $\vdash_{wf}$ *?t2* **using** *wfT-e-eq fresh-GNil wb1 wb2* **by** *metis*
show *?xfree* **using** *x* **by** *auto*
show Θ ; $\mathcal{B}$; (*x*, *B-bitvec*, (*CE-val* (*V-var z*) == *CE-val* (*V-lit* (*L-bitvec* (*v1* @ *v2*))))$[z::=$*V-var x*$]_v$) $\#_\Gamma$
GNil $\models$ (*CE-val* (*V-var z*) == *CE-concat* $[$(*V-lit* (*L-bitvec v1*))$]^{ce}$ $[$(*V-lit* (*L-bitvec v2*))$]^{ce}$)$[z::=$*V-var x*$]_v$
using *valid-eval-e-eq eval-e-concat-eq wb1 wb2 subst-v-c-def* **by** *fastforce*
qed
qed

lemma *subtype-len*:
assumes $\vdash_{wf}$ Θ
shows Θ ; $\mathcal{B}$; *GNil* $\vdash$ $\{\!|$ *z'* : *B-int* | *CE-val* (*V-var z'*) == *CE-val* (*V-lit* (*L-num* (*int* (*length v*)))) $|\!\}$ $\lesssim$
$\{\!|$ *z* : *B-int* | *CE-val* (*V-var z*) == *CE-len* $[$(*V-lit* (*L-bitvec v*))$]^{ce}$ $|\!\}$ (**is** Θ ; $\mathcal{B}$; *GNil* $\vdash$ *?t1* $\lesssim$ *?t2*)
proof −

have $*$: Θ $\vdash_{wf}$ [] $\wedge$ Θ ; $\mathcal{B}$; *GNil* $\vdash_{wf}$ $[]_\Delta$ **using** *assms wfG-nilI wfD-emptyI wfPhi-emptyI* **by** *auto*
obtain *x*::*x* **where** *x*: *atom x* $\sharp$ (Θ, $\mathcal{B}$, *GNil*, *z'*, *CE-val* (*V-var z'*) == *CE-val* (*V-lit* (*L-num* (*int* (*length v*)))) , *z*, *CE-val* (*V-var z*) == *CE-len* $[$(*V-lit* (*L-bitvec v*))$]^{ce}$)
(**is** *atom x* $\sharp$ *?F*)
using *obtain-fresh* **by** *metis*
then show *?thesis* **proof**
have Θ ; $\mathcal{B}$; *GNil* $\vdash_{wf}$ *CE-val* (*V-lit* (*L-num* (*int* (*length v*)))) : *B-int*
using *wfCE-valI* $*$ *wfV-litI base-for-lit.simps*
by (*metis wfE-valI wfX-wfY*)

thus Θ ; $\mathcal{B}$; *GNil* $\vdash_{wf}$ *?t1* **using** *wfT-e-eq fresh-GNil* **by** *auto*

have Θ ; $\mathcal{B}$; *GNil* $\vdash_{wf}$ *CE-len* $[$(*V-lit* (*L-bitvec v*))$]^{ce}$: *B-int*
using *wfE-valI* $*$ *wfV-litI base-for-lit.simps wfE-valI wfX-wfY*
by (*metis wfCE-lenI wfCE-valI*)

thus Θ ; $\mathcal{B}$; *GNil* $\vdash_{wf}$ *?t2* **using** *wfT-e-eq fresh-GNil* **by** *auto*

show Θ ; $\mathcal{B}$; (*x*, *B-int*, (*CE-val* (*V-var z'*) == *CE-val* (*V-lit* (*L-num* (*int* (*length v*)))))$[z'::=$*V-var x*$]_v$) $\#_\Gamma$ *GNil* $\models$ (*CE-val* (*V-var z*) == *CE-len* $[$(*V-lit* (*L-bitvec v*))$]^{ce}$)$[z::=$*V-var x*$]_v$
(**is** Θ ; $\mathcal{B}$; *?G* $\models$ *?c*) **using** *valid-len assms subst-v-c-def* **by** *auto*
qed
qed

lemma *subtype-base-fresh*:
assumes Θ ; $\mathcal{B}$; $\Gamma \vdash_{wf} \{\!|\ z : b\ |\ c\ |\!\}$ **and** Θ ; $\mathcal{B}$; $\Gamma \vdash_{wf} \{\!|\ z : b\ |\ c'\ |\!\}$ **and**
atom $z \sharp \Gamma$ **and** Θ ; $\mathcal{B}$; $(z, b, c) \#_\Gamma \Gamma \models c'$
shows Θ ; $\mathcal{B}$; $\Gamma \vdash \{\!|\ z : b\ |\ c\ |\!\} \lesssim \{\!|\ z : b\ |\ c'\ |\!\}$
proof −
obtain $x::x$ **where** $*$:*atom* $x \sharp ((\Theta , \mathcal{B} , z, c, z, c', \Gamma) , (\Theta, \mathcal{B}, \Gamma, \{\!|\ z : b\ |\ c\ |\!\}, \{\!|\ z : b\ |\ c'\ |\!\}))$ **using** *obtain-fresh* **by** *metis*
moreover hence *atom* $x \sharp \Gamma$ **using** *fresh-Pair* **by** *auto*
moreover hence Θ ; $\mathcal{B}$; $(x, b, c[z::=V\text{-}var\ x]_v) \#_\Gamma \Gamma \models c'[z::=V\text{-}var\ x]_v$ **using** *assms valid-flip-simple* $*$ *subst-v-c-def* **by** *auto*
ultimately show *?thesis* **using** *subtype-baseI assms* τ*.fresh fresh-Pair* **by** *metis*
qed

lemma *subtype-bop*:
assumes *wfG* Θ $\mathcal{B}$ Γ **and** $opp = Plus \wedge ll = (L\text{-}num\ (n1+n2)) \vee (opp = LEq \wedge ll = (\ if\ n1 \leq n2\ then\ L\text{-}true\ else\ L\text{-}false))$
and $(opp = Plus \longrightarrow b = B\text{-}int) \wedge (opp = LEq \longrightarrow b = B\text{-}bool)$
shows Θ ; $\mathcal{B}$; $\Gamma \vdash (\{\!|\ z : b \mid C\text{-}eq\ (CE\text{-}val\ (V\text{-}var\ z))\ (CE\text{-}val\ (V\text{-}lit\ (ll)))\ |\!\}) \lesssim$
$\{\!|\ z : b \mid C\text{-}eq\ (CE\text{-}val\ (V\text{-}var\ z))\ (CE\text{-}op\ opp\ [(V\text{-}lit\ (L\text{-}num\ n1))]^{ce}\ [(V\text{-}lit\ (L\text{-}num\ n2))]^{ce})\ |\!\}$ (**is** Θ ; $\mathcal{B}$; $\Gamma \vdash ?T1 \lesssim ?T2$)
proof −
obtain $x::x$ **where** *xf*: *atom* $x \sharp (z, CE\text{-}val\ (V\text{-}var\ z) == CE\text{-}val\ (V\text{-}lit\ (ll)) , z, CE\text{-}val\ (V\text{-}var\ z) == CE\text{-}op\ opp\ [(V\text{-}lit\ (L\text{-}num\ n1))]^{ce}\ [(V\text{-}lit\ (L\text{-}num\ n2))]^{ce} , \Gamma)$
using *obtain-fresh* **by** *blast*

have Θ ; $\mathcal{B}$; $\Gamma \vdash (\{\!|\ x : b \mid C\text{-}eq\ (CE\text{-}val\ (V\text{-}var\ x))\ (CE\text{-}val\ (V\text{-}lit\ (ll)))\ |\!\}) \lesssim$
$\{\!|\ x : b \mid C\text{-}eq\ (CE\text{-}val\ (V\text{-}var\ x))\ (CE\text{-}op\ opp\ [(V\text{-}lit\ (L\text{-}num\ n1))]^{ce}\ [(V\text{-}lit\ (L\text{-}num\ n2))]^{ce})\ |\!\}$ (**is** Θ ; $\mathcal{B}$; $\Gamma \vdash ?S1 \lesssim ?S2$)
proof(*rule subtype-base-fresh*)

show *atom* $x \sharp \Gamma$ **using** *xf fresh-Pair* **by** *auto*

show $wfT\ \Theta\ \mathcal{B}\ \Gamma\ (\{\!|\ x : b \mid CE\text{-}val\ (V\text{-}var\ x) == CE\text{-}val\ (V\text{-}lit\ ll)\ |\!\})$ (**is** $wfT\ \Theta\ \mathcal{B}\ ?A\ ?B$)
proof(*rule wfT-e-eq*)
have Θ ; $\mathcal{B}$; $\Gamma \vdash_{wf} (V\text{-}lit\ ll) : b$ **using** *wfV-litI base-for-lit.simps assms* **by** *metis*
thus Θ ; $\mathcal{B}$; $\Gamma \vdash_{wf} CE\text{-}val\ (V\text{-}lit\ ll) : b$ **using** *wfCE-valI* **by** *auto*
show *atom* $x \sharp \Gamma$ **using** *xf fresh-Pair* **by** *auto*
qed

show $wfT\ \Theta\ \mathcal{B}\ \Gamma\ (\{\!|\ x : b\ |\ CE\text{-}val\ (V\text{-}var\ x) == CE\text{-}op\ opp\ [(V\text{-}lit\ (L\text{-}num\ n1))]^{ce}\ [(V\text{-}lit\ (L\text{-}num\ n2))]^{ce}\ |\!\})$ (**is** $wfT\ \Theta\ \mathcal{B}\ ?A\ ?C$)
proof(*rule wfT-e-eq*,*rule opp.exhaust*[*of opp*])
{ **assume** $opp = Plus$
thus Θ ; $\mathcal{B}$; $\Gamma \vdash_{wf} CE\text{-}op\ opp\ [(V\text{-}lit\ (L\text{-}num\ n1))]^{ce}\ [(V\text{-}lit\ (L\text{-}num\ n2))]^{ce} : b$ **using** *wfCE-valI wfCE-plusI assms wfV-litI base-for-lit.simps assms* **by** *metis*
}
next
{ **assume** $opp = LEq$
thus Θ ; $\mathcal{B}$; $\Gamma \vdash_{wf} CE\text{-}op\ opp\ [(V\text{-}lit\ (L\text{-}num\ n1))]^{ce}\ [(V\text{-}lit\ (L\text{-}num\ n2))]^{ce} : b$ **using** *wfCE-valI wfCE-leqI assms wfV-litI base-for-lit.simps assms* **by** *metis*
}
show *atom* $x \sharp \Gamma$ **using** *xf fresh-Pair* **by** *auto*

qed

show $\Theta; \mathcal{B}$; $(x, b, (CE\text{-}val\ (V\text{-}var\ x)\ ==\ CE\text{-}val\ (V\text{-}lit\ (ll))\ ))\ \#_\Gamma\ \Gamma$
$\models (CE\text{-}val\ (V\text{-}var\ x)\ ==\ CE\text{-}op\ opp\ [V\text{-}lit\ (L\text{-}num\ n1)]^{ce}\ [V\text{-}lit\ (L\text{-}num\ n2)]^{ce})$
(**is** Θ ; $\mathcal{B}$; *?G* $\models$ *?c*)
using *valid-bop assms xf* **by** *simp*

qed
moreover have *?S1 = ?T1* **using** *type-l-eq* **by** *auto*
moreover have *?S2 = ?T2* **using** *type-e-eq ce.fresh v.fresh supp-l-empty fresh-def empty-iff fresh-e-opp*

by (*metis ms-fresh-all*(*4*))
ultimately show *?thesis* **by** *auto*

qed

lemma *subtype-top*:
assumes *wfT* $\Theta\ \mathcal{B}\ G$ $(\{\!|\ z : b \mid c\ |\!\})$
shows Θ ; $\mathcal{B}$; $G\ \vdash\ (\{\!|\ z : b \mid c\ |\!\}) \lesssim (\{\!|\ z : b \mid TRUE\ |\!\})$
proof −
obtain *x*::*x* **where** ∗: $atom\ x \sharp (\Theta, \mathcal{B}, G, z, c, z, TRUE)$ **using** *obtain-fresh* **by** *blast*
then show *?thesis* **proof**(*rule subtype-baseI*)
show ‹ Θ ; $\mathcal{B}$; $G\ \vdash_{wf} \{\!|\ z : b \mid c\ |\!\}$ › **using** *assms* **by** *auto*
show ‹ Θ ; $\mathcal{B}$;$G\ \vdash_{wf} \{\!|\ z : b \mid TRUE\ |\!\}$ › **using** *wfT-TRUE assms wfX-wfY b-of.simps wfT-wf*
by (*metis wfX-wfB*(*8*))
hence Θ ; $\mathcal{B}\ \vdash_{wf}\ (x, b, c[z::=V\text{-}var\ x]_v)\ \#_\Gamma\ G$ **using** *wfT-wf-cons3 assms fresh-Pair* ∗ *subst-v-c-def*
by *auto*
thus ‹Θ ; $\mathcal{B}$; $(x, b, c[z::=V\text{-}var\ x]_v)\ \#_\Gamma\ G \models (TRUE)[z::=V\text{-}var\ x]_v$ › **using** *valid-trueI subst-cv.simps*
subst-v-c-def **by** *metis*
qed
qed

thm *valid-split*

thm *valid-wf-all*

lemma *if-simp*:
(*if x = x then e1 else e2*) = *e1*
by *auto*

lemma *subtype-split*:
assumes *split n v* (*v1*,*v2*) **and** $\vdash_{wf} \Theta$
shows Θ ; {||} ; *GNil* $\vdash \{\!|\ z : [\ B\text{-}bitvec\ ,\ B\text{-}bitvec\ ]^b\ |\ [\ [\ z\ ]^v\ ]^{ce}\ ==\ [\ [\ [\ L\text{-}bitvec$
v1 $]^v$, [*L-bitvec*
v2 $]^v\ ]^v\ ]^{ce}\ |\!\} \lesssim \{\!|\ z : [\ B\text{-}bitvec\ ,\ B\text{-}bitvec\ ]^b\ |\ [\ [\ L\text{-}bitvec$
$v\ ]^v\ ]^{ce}\ ==\ [\ [\#1[\ [\ z\ ]^v\ ]^{ce}]^{ce}\ @@\ [\#2[\ [\ z\ ]^v\ ]^{ce}]^{ce}\ ]^{ce}\ \ AND\ \ [|\ [\#1[\ [\ z\ ]^v\ ]^{ce}]^{ce}\ |]^{ce}\ ==\ [$
[*L-num*
$n\ ]^v\ ]^{ce}\ \ |\!\}$
(**is** Θ ;*?B* ; *GNil* $\vdash \{\!|\ z : [\ B\text{-}bitvec\ ,\ B\text{-}bitvec\ ]^b\ |\ ?c1\ |\!\} \lesssim \{\!|\ z : [\ B\text{-}bitvec\ ,\ B\text{-}bitvec\ ]^b\ |\ ?c2\ |\!\}$)
proof −

obtain $x::x$ **where** xf:$atom\ x \sharp (\Theta,\ ?B,\ GNil,\ z,\ ?c1,z,\ ?c2\ )$ **using** *obtain-fresh* **by** *auto*
then show *?thesis* **proof**(*rule subtype-baseI*)
show $*$: $\langle \Theta\ ;\ ?B\ ;\ (x,\ [\ B\text{-}bitvec\ ,\ B\text{-}bitvec\ ]^b,\ (?c1)[z::=[\ x\ ]^v]_v)\ \#_\Gamma$
$GNil\ \models (?c2)[z::=[\ x\ ]^v]_v\ \rangle$
unfolding *subst-v-c-def subst-cv.simps subst-cev.simps subst-vv.simps if-simp*
using *valid-split*[*OF assms*, *of x*] **by** *simp*
show $\langle\ \Theta\ ;\ ?B\ ;\ GNil\ \vdash_{wf} \{\!|\ z : [\ B\text{-}bitvec\ ,\ B\text{-}bitvec\ ]^b|\ ?c1\ |\!\}\ \rangle$ **using** *valid-wfT*[*OF* $*$] *xf fresh-prodN* **by** *metis*
show $\langle\ \Theta\ ;\ ?B\ ;\ GNil\ \vdash_{wf} \{\!|\ z : [\ B\text{-}bitvec\ ,\ B\text{-}bitvec\ ]^b|\ ?c2\ |\!\}\ \rangle$ **using** *valid-wfT*[*OF* $*$] *xf fresh-prodN* **by** *metis*
qed
qed

lemma *subtype-range*:
fixes $n::int$ **and** $\Gamma::\Gamma$
assumes $0 \le n \wedge n \le int\ (length\ v)$ **and** $\Theta\ ;\ \{||\} \vdash_{wf} \Gamma$
shows $\Theta\ ;\ \{||\}\ ;\ \Gamma\ \vdash \{\!|\ z : B\text{-}int\ |\ [\ [\ z\ ]^v\ ]^{ce} == [\ [\ L\text{-}num\ n\ ]^v\ ]^{ce}\ |\!\} \lesssim$
$\{\!|\ z : B\text{-}int\ |\ ([\ leq\ [\ [\ L\text{-}num\ 0\ ]^v\ ]^{ce}\ [\ [\ z\ ]^v\ ]^{ce}\ ]^{ce} == [\ [\ L\text{-}true\ ]^v\ ]^{ce})\ AND\ (\ [\ leq\ [\ [\ z\ ]^v\ ]^{ce}\ [|\ [\ [\ L\text{-}bitvec\ v\ ]^v\ ]^{ce}\ |]^{ce}\ ]^{ce} == [\ [\ L\text{-}true\ ]^v\ ]^{ce})\ |\!\}$
(**is** $\Theta\ ;\ ?B\ ;\ \Gamma\ \vdash \{\!|\ z : B\text{-}int\ |\ ?c1\ |\!\} \lesssim \{\!|\ z : B\text{-}int\ |\ ?c2\ AND\ ?c3\ |\!\}$)
proof −
obtain $x::x$ **where** $*$:$\langle atom\ x \sharp (\Theta,\ ?B,\ \Gamma,\ z,\ ?c1\ ,\ z,\ ?c2\ AND\ ?c3)\rangle$ **using** *obtain-fresh* **by** *auto*
moreover have $**$:$\langle\Theta\ ;\ ?B\ ;\ (x,\ B\text{-}int,\ (?c1)[z::=[\ x\ ]^v]_v)\ \#_\Gamma\ \Gamma\ \models (?c2\ AND\ ?c3)[z::=[\ x\ ]^v]_v\rangle$
unfolding *subst-v-c-def subst-cv.simps subst-cev.simps subst-vv.simps if-simp* **using** *valid-range-length*[*OF assms(1)*] *assms fresh-prodN* $*$ **by** *simp*

moreover hence $\langle\ \Theta\ ;\ ?B\ ;\ \Gamma\ \vdash_{wf} \{\!|\ z : B\text{-}int\ |\ [\ [\ z\ ]^v\ ]^{ce}\ ==\ [\ [\ L\text{-}num\ n\ ]^v\ ]^{ce}\ |\!\}\ \rangle$ **using** *valid-wfT* $*$ *fresh-prodN* **by** *metis*
moreover have $\langle\ \Theta\ ;\ ?B\ ;\ \Gamma\ \vdash_{wf} \{\!|\ z : B\text{-}int\ |\ ?c2\ AND\ ?c3\ |\!\}\ \rangle$
using *valid-wfT*[*OF* $**$] $*$ *fresh-prodN* **by** *metis*
ultimately show *?thesis* **using** *subtype-baseI* **by** *auto*
qed

lemma *check-num-range*:
assumes $0 \le n \wedge n \le int\ (length\ v)$ **and** $\vdash_{wf} \Theta$
shows $\Theta\ ;\ \{||\}\ ;\ GNil\ \vdash [\ L\text{-}num\ n\ ]^v \Leftarrow \{\!|\ z : B\text{-}int\ |\ [\ leq\ [\ [\ L\text{-}num\ 0\ ]^v\ ]^{ce}\ [\ [\ z\ ]^v\ ]^{ce}\ ]^{ce} == [\ [\ L\text{-}true\ ]^v\ ]^{ce}\ \ AND\ \ [\ leq\ [\ [\ z\ ]^v\ ]^{ce}\ [|\ [\ [\ L\text{-}bitvec\ v\ ]^v\ ]^{ce}\ |]^{ce}\ ]^{ce} == [\ [\ L\text{-}true\ ]^v\ ]^{ce}\ |\!\}$
using *assms subtype-range check-v.intros infer-v-litI wfG-nilI*
by (*meson infer-natI*)

12.2 Literals

nominal-function *type-for-lit* :: $l \Rightarrow \tau$ **where**
type-for-lit (*L-true*) = ($\{\!|\ z : B\text{-}bool\ |\ [[z]^v]^{ce} == [V\text{-}lit\ L\text{-}true]^{ce}\ |\!\}$)
| *type-for-lit* (*L-false*) = ($\{\!|\ z : B\text{-}bool\ |\ [[z]^v]^{ce} == [V\text{-}lit\ L\text{-}false]^{ce}\ |\!\}$)
| *type-for-lit* (*L-num n*) = ($\{\!|\ z : B\text{-}int\ |\ [[z]^v]^{ce} == [V\text{-}lit\ (L\text{-}num\ n)]^{ce}\ |\!\}$)
| *type-for-lit* (*L-unit*) = ($\{\!|\ z : B\text{-}unit\ |\ [[z]^v]^{ce} == [V\text{-}lit\ (L\text{-}unit\ )]^{ce}\ |\!\}$)
| *type-for-lit* (*L-bitvec v*) = ($\{\!|\ z : B\text{-}bitvec\ |\ [[z]^v]^{ce} == [V\text{-}lit\ (L\text{-}bitvec\ v)]^{ce}\ |\!\}$)

by (*auto simp*: *eqvt-def type-for-lit-graph-aux-def*, *metis l.strong-exhaust*,(*simp add*: *permute-int-def flip-bitvec0*)+)
nominal-termination (*eqvt*) **by** *lexicographic-order*

nominal-function *type-for-var* :: $\Gamma \Rightarrow \tau \Rightarrow x \Rightarrow \tau$ **where**
type-for-var G τ x = (*case lookup G x of*
None ⇒ *τ*
| *Some* (*b*,*c*) ⇒ ({| *x* : *b* | *c* |}))
apply *auto* **unfolding** *eqvt-def* **apply**(*rule allI*) **unfolding** *type-for-var-graph-aux-def eqvt-def* **by** *simp*
nominal-termination (*eqvt*) **by** *lexicographic-order*

lemma *infer-l-form*:
fixes *l*::*l* **and** *tm*::'*a*::*fs*
assumes ⊢ *l* ⇒ *τ*
shows ∃ *z b*. *τ* = ({| *z* : *b* | *C-eq* (*CE-val* (*V-var z*)) (*CE-val* (*V-lit l*)) |}) ∧ *atom z* ♯ *tm*
proof −
obtain *z*′ **and** *b* **where** *t*:*τ* = ({| *z*′ : *b* | *C-eq* (*CE-val* (*V-var z*′)) (*CE-val* (*V-lit l*)) |}) **using** *infer-l-elims assms* **using** *infer-l.simps type-for-lit.simps*
type-for-lit.cases **by** *blast*
obtain *z*::*x* **where** *zf*: *atom z* ♯ *tm* **using** *obtain-fresh* **by** *metis*
have *τ* = {| *z* : *b* | *C-eq* (*CE-val* (*V-var z*)) (*CE-val* (*V-lit l*)) |} **using** *type-e-eq ce.fresh v.fresh l.fresh*

by (*metis t type-l-eq*)
thus *?thesis* **using** *zf* **by** *auto*
qed

lemma *infer-l-form3*:
fixes *l*::*l*
assumes ⊢ *l* ⇒ *τ*
shows ∃ *z*. *τ* = ({| *z* : *base-for-lit l* | *C-eq* (*CE-val* (*V-var z*)) (*CE-val* (*V-lit l*)) |})
using *infer-l-elims* **using** *assms* **using** *infer-l.simps type-for-lit.simps base-for-lit.simps* **by** *auto*

lemma *infer-l-form4* [*simp*]:
fixes Γ::Γ
assumes Θ ; $\mathcal{B} \vdash_{wf}$ Γ
shows ∃ *z*. ⊢ *l* ⇒ ({| *z* : *base-for-lit l* | *C-eq* (*CE-val* (*V-var z*)) (*CE-val* (*V-lit l*)) |})
using *assms* *infer-l-form2 infer-l-form3* **by** *metis*

lemma *infer-v-unit-form*:
fixes *v*::*v*
assumes *P* ; $\mathcal{B}$; Γ ⊢ *v* ⇒ ({| *z1* : *B-unit* | *c1* |}) **and** *supp v* = {}
shows *v* = *V-lit L-unit*
using *assms* **proof**(*nominal-induct* Γ *v* {| *z1* : *B-unit* | *c1* |} *rule*: *infer-v.strong-induct*)
case (*infer-v-varI* Θ $\mathcal{B}$ *c x z*)
then show *?case* **using** *supp-at-base* **by** *auto*
next
case (*infer-v-litI* Θ $\mathcal{B}$ Γ *l*)
from ‹⊢ *l* ⇒ {| *z1* : *B-unit* | *c1* |}› **show** *?case* **by**(*nominal-induct* {| *z1* : *B-unit* | *c1* |} *rule*:

infer-l.strong-induct,*auto*)
qed

lemma *base-for-lit-wf*:
assumes $\vdash_{wf} \Theta$
shows Θ ; $\mathcal{B}$ $\vdash_{wf}$ *base-for-lit l*
using *base-for-lit.simps* **using** *wfV-elims wf-intros assms l.exhaust* **by** *metis*

lemma *infer-l-t-wf*:
fixes Γ::Γ
assumes Θ ; $\mathcal{B}$ $\vdash_{wf}$ Γ $\wedge$ *atom z* $\sharp$ Γ
shows Θ ; $\mathcal{B}$; Γ $\vdash_{wf}$ $\{\!|$ *z* : *base-for-lit l* | *C-eq* (*CE-val* (*V-var z*)) (*CE-val* (*V-lit l*)) $|\!\}$
proof
show *atom z* $\sharp$ (Θ, $\mathcal{B}$, Γ) **using** *wfG-fresh-x assms* **by** *auto*
show Θ ; $\mathcal{B}$ $\vdash_{wf}$ *base-for-lit l* **using** *base-for-lit-wf assms wfX-wfY* **by** *metis*
thus Θ ; $\mathcal{B}$; (*z*, *base-for-lit l*, *TRUE*) $\#_\Gamma$ Γ $\vdash_{wf}$ *CE-val* (*V-var z*) == *CE-val* (*V-lit l*) **using** *wfC-v-eq wfV-litI assms wfX-wfY* **by** *metis*
qed

lemma *infer-l-wf*:
fixes *l*::*l* **and** Γ::Γ **and** τ::τ **and** Θ::Θ
assumes $\vdash$ *l* $\Rightarrow$ τ **and** Θ ; $\mathcal{B}$ $\vdash_{wf}$ Γ
shows $\vdash_{wf}$ Θ **and** Θ ; $\mathcal{B}$ $\vdash_{wf}$ Γ **and** Θ ; $\mathcal{B}$; Γ $\vdash_{wf}$ τ
proof −
show $*$:Θ ; $\mathcal{B}$ $\vdash_{wf}$ Γ **using** *assms infer-l-elims* **by** *auto*
thus $\vdash_{wf}$ Θ **using** *wfX-wfY* **by** *auto*
show $*$:Θ ; $\mathcal{B}$; Γ $\vdash_{wf}$ τ **using** *infer-l-t-wf assms infer-l-form3* $*$
by (*metis* ‹$\vdash_{wf}$ Θ›) *fresh-GNil wfG-nilI wfT-weakening-nil*)
qed

lemma *infer-l-uniqueness*:
fixes *l*::*l*
assumes $\vdash$ *l* $\Rightarrow$ τ **and** $\vdash$ *l* $\Rightarrow$ τ'
shows $\tau = \tau'$
using *assms*
proof −
obtain *z* **and** *b* **where** *zt*: τ = ($\{\!|$ *z* : *b* | *C-eq* (*CE-val* (*V-var z*)) (*CE-val* (*V-lit l*)) $|\!\}$) **using** *infer-l-form assms* **by** *blast*
obtain *z'* **and** *b* **where** *z't*: τ' = ($\{\!|$ *z'* : *b* | *C-eq* (*CE-val* (*V-var z'*)) (*CE-val* (*V-lit l*)) $|\!\}$) **using** *infer-l-form assms* **by** *blast*
thus *?thesis* **using** *type-l-eq zt z't assms infer-l.simps infer-l-elims l.distinct*
by (*metis infer-l-form3*)
qed

12.3 Values

lemma *type-v-eq*:
assumes $\{\!|$ *z1* : *b1* | *c1* $|\!\}$ = $\{\!|$ *z* : *b* | *C-eq* (*CE-val* (*V-var z*)) (*CE-val* (*V-var x*)) $|\!\}$ **and** *atom z* $\sharp$ *x*
shows *b* = *b1* **and** *c1* = *C-eq* (*CE-val* (*V-var z1*)) (*CE-val* (*V-var x*))
using *assms* **by** (*auto*,*metis Abs1-eq-iff* τ*.eq-iff assms c.fresh ce.fresh type-e-eq v.fresh*)

lemma *infer-var2* [*elim*]:

assumes P ; $\mathcal{B}$; $G \vdash$ *V-var* $x \Rightarrow \tau$
shows $\exists$ b c. *Some* (b,c) = *lookup* G x
using *assms infer-v-elims lookup-iff* **by** (*metis* (*no-types*, *lifting*))

lemma *infer-var3* [*elim*]:
assumes Θ ; $\mathcal{B}$; $\Gamma \vdash$ *V-var* $x \Rightarrow \tau$
shows $\exists$ z b c. *Some* (b,c) = *lookup* Γ $x \wedge \tau = (\{\!|\ z : b \mid$ *C-eq* (*CE-val* (*V-var* z)) (*CE-val* (*V-var* x)) $|\!\}) \wedge$ *atom* $z \sharp x \wedge$ *atom* $z \sharp \Gamma$
using *infer-v-elims*(*1*)[*OF assms*(*1*)] **by** *metis*

lemma *infer-bool-options2*:
fixes v::v
assumes Θ ; $\mathcal{B}$; $\Gamma \vdash v \Rightarrow \{\!|\ z : b \mid c\ |\!\}$ **and** *supp* $v = \{\} \wedge b =$ *B-bool*
shows $v =$ *V-lit L-true* $\vee$ ($v =$ (*V-lit L-false*))
using *assms*
proof(*nominal-induct* v *arbitrary*: b *rule*: *v.strong-induct*)
case (*V-lit* l)
then show *?case* **proof**(*nominal-induct* l *rule*: *l.strong-induct*)
case (*L-num nat*)
hence $\vdash$ *L-num nat* $\Rightarrow \{\!|\ z : b \mid c\ |\!\}$ **using** *infer-v-elims*(*2*) **by** (*metis* (*no-types*, *lifting*))
hence $b =$ *B-int* **using** *infer-l-elims*(*3*) *type-for-lit.simps*(*3*) **by** (*metis* *τ.eq-iff*)
then show *?case* **using** *L-num* **by** *fastforce*
next
case *L-true*
hence $\vdash$ *L-true* $\Rightarrow \{\!|\ z : b \mid c\ |\!\}$ **using** *infer-v-elims*(*2*) **by** (*metis* (*no-types*, *lifting*))
hence $b =$ *B-bool* **using** *infer-l-elims type-for-lit.simps* **by** (*metis* *τ.eq-iff*)
then show *?case* **by** *blast*
next
case *L-false*
hence $\vdash$ *L-false* $\Rightarrow \{\!|\ z : b \mid c\ |\!\}$ **using** *infer-v-elims*(*2*) **by** (*metis* (*no-types*, *lifting*))
hence $b =$ *B-bool* **using** *infer-l-elims type-for-lit.simps* **by** (*metis* *τ.eq-iff*)
then show *?case* **by** *blast*
next
case *L-unit*
hence $\vdash$ *L-unit* $\Rightarrow \{\!|\ z : b \mid c\ |\!\}$ **using** *infer-v-elims*(*2*) **by** (*metis* (*no-types*, *lifting*))
hence $b =$ *B-unit* **using** *infer-l-elims type-for-lit.simps* **by** (*metis* *τ.eq-iff*)
then show *?case* **using** *L-unit* **by** *fastforce*
next
case (*L-bitvec x*)
hence $\vdash$ *L-bitvec* $x \Rightarrow \{\!|\ z : b \mid c\ |\!\}$ **using** *infer-v-elims* **by** (*metis* (*no-types*, *lifting*))
hence $b =$ *B-bitvec* **using** *infer-l-elims type-for-lit.simps* **by** (*metis* *τ.eq-iff*)
then show *?case* **using** *L-bitvec* **by** *fastforce*
qed
next
case (*V-var x*)
then show *?case* **using** *v.supp V-var supp-at-base*[*of x*] **by** *auto*
next
case (*V-pair v1 v2*)
then show *?case* **using** *infer-v.simps*
τ.eq-iff infer-v-elims **by** (*metis b.distinct*)
next
case (*V-cons dc v*)

then show *?case* **using** *infer-v.simps*
τ.eq-iff infer-v-elims **by** (*metis b.distinct*)
next
case (*V-consp tyid dc b′ v*)
then show *?case* **using** *infer-v.simps*
τ.eq-iff infer-v-elims **by** (*metis b.distinct*)
qed

lemma *infer-bool-options*:
fixes *v*::*v*
assumes Θ ; $\mathcal{B}$; Γ ⊢ *v* ⇒ {| *z* : *B-bool* | *c* |} **and** *supp v* = {}
shows *v* = *V-lit L-true* ∨ (*v* = (*V-lit L-false*))
using *infer-bool-options2 assms* **by** *blast*

lemma *infer-int2*:
fixes *v*::*v*
assumes Θ ; $\mathcal{B}$; Γ ⊢ *v* ⇒ {| *z* : *b* | *c* |}
shows *supp v* = {} ∧ *b* = *B-int* ⟶ (∃ *n*. *v* = *V-lit* (*L-num n*))
using *assms*
proof(*nominal-induct v rule*: *v.strong-induct*)
case (*V-lit l*)
then show *?case* **proof**(*nominal-induct l rule*: *l.strong-induct*)
case (*L-num nat*)
hence ⊢ *L-num nat* ⇒ {| *z* : *b* | *c* |} **using** *infer-v-elims*(*2*) **by** (*metis* (*no-types*, *lifting*))
hence *b* = *B-int* **using** *infer-l-elims*(*3*) *type-for-lit.simps*(*3*) **by** (*metis τ.eq-iff*)
then show *?case* **by** *fastforce*
next
case *L-true*
hence ⊢ *L-true* ⇒ {| *z* : *b* | *c* |} **using** *infer-v-elims*(*2*) **by** (*metis* (*no-types*, *lifting*))
hence *b* = *B-bool* **using** *infer-l-elims type-for-lit.simps* **by** (*metis τ.eq-iff*)
then show *?case* **by** *simp*
next
case *L-false*
hence ⊢ *L-false* ⇒ {| *z* : *b* | *c* |} **using** *infer-v-elims*(*2*) **by** (*metis* (*no-types*, *lifting*))
hence *b* = *B-bool* **using** *infer-l-elims type-for-lit.simps* **by** (*metis τ.eq-iff*)
then show *?case* **by** *simp*
next
case *L-unit*
hence ⊢ *L-unit* ⇒ {| *z* : *b* | *c* |} **using** *infer-v-elims* **by** (*metis* (*no-types*, *lifting*))
hence *b* = *B-unit* **using** *infer-l-elims type-for-lit.simps* **by** (*metis τ.eq-iff*)
then show *?case* **by** *simp*
next
case (*L-bitvec x*)
hence ⊢ *L-bitvec x* ⇒ {| *z* : *b* | *c* |} **using** *infer-v-elims* **by** (*metis* (*no-types*, *lifting*))
hence *b* = *B-bitvec* **using** *infer-l-elims type-for-lit.simps* **by** (*metis τ.eq-iff*)
then show *?case* **by** *fastforce*
qed
next
case (*V-var x*)
then show *?case* **using** *v.supp supp-at-base* **by** *auto*
next
case (*V-pair v1 v2*)

then show *?case* **using** *infer-v.simps*
τ.eq-iff infer-v-elims **by** (*metis b.distinct*)
next
case (*V-cons s dc v*)
then show *?case* **using** *infer-v.simps*
τ.eq-iff infer-v-elims **by** (*metis b.distinct*)
next
case (*V-consp s dc b v*)
then show *?case* **using** *infer-v.simps*
τ.eq-iff infer-v-elims **by** (*metis b.distinct*)
qed

lemma *infer-bitvec*:
fixes Θ::Θ **and** *v*::*v*
assumes Θ ; $\mathcal{B}$; Γ ⊢ *v* ⇒ {| *z'* : *B-bitvec* | *c'* |} **and** *supp v* = {}
shows ∃ *bv*. *v* = *V-lit* (*L-bitvec bv*)
using *assms* **proof**(*nominal-induct v rule*: *v.strong-induct*)
case (*V-lit l*)
then show *?case* **by**(*nominal-induct l rule*: *l.strong-induct*,*force*+)
next
case (*V-consp s dc b v*)
then show *?case* **using** *infer-v-elims*(*7*)[*OF V-consp*(*2*)] *τ.eq-iff* **by** *auto*
next
case (*V-var x*)
then show *?case* **using** *supp-at-base* **by** *auto*
qed(*force*+)

lemma *infer-int*:
assumes *infer-v* Θ $\mathcal{B}$ Γ *v* ({| *z* : *B-int* | *c* |}) **and** *supp v*= {}
shows ∃ *n*. *V-lit* (*L-num n*) = *v*
using *assms infer-int2* **by** (*metis* (*no-types*, *lifting*))

lemma *infer-v-form*[*simp*]:
fixes *v*::*v*
assumes Θ ; $\mathcal{B}$; Γ ⊢ *v* ⇒ *τ*
shows ∃ *z b*. *τ* = ({| *z* : *b* | *C-eq* (*CE-val* (*V-var z*)) (*CE-val v*)|}) ∧ *atom z* ♯ *v* ∧ *atom z* ♯ Γ
using *assms*
proof(*nominal-induct v arbitrary*: *τ rule*: *v.strong-induct*)
case (*V-lit l*)
hence ⊢ *l* ⇒ *τ* **using** *infer-v-elims* **by** *metis*
then obtain *z* **and** *b* **where** *τ* = {| *z* : *b* | *CE-val* (*V-var z*) == *CE-val* (*V-lit l*) |} ∧*atom z* ♯ Γ
using *infer-l-form* **by** *metis*
moreover hence *atom z* ♯ (*V-lit l*) **using** *supp-l-empty v.fresh*(*1*) *fresh-prod2 fresh-def* **by** *blast*
ultimately show *?case* **by** *metis*
next
case (*V-var x*)
then show *?case* **using** *infer-v-elims V-var*
by (*metis finite.emptyI fresh-atom-at-base fresh-finite-insert v.fresh*(*2*))
next
case (*V-pair v1 v2*)

obtain *z* **and** *z1* **and** *b1* **and** *c1* **and** *z2* **and** *b2* **and** *c2* **where**

zbc: $\tau = (\{\!|\ z : \textit{B-pair b1 b2}\ |\ \textit{CE-val}\ (\textit{V-var z})\ ==\ \textit{CE-val}\ (\textit{V-pair v1 v2})\ |\!\}) \wedge$
atom z $\sharp$ (*v1*, *v2*) $\wedge$ Θ ; $\mathcal{B}$; $\Gamma \vdash$ *v1* $\Rightarrow \{\!|$ *z1* : *b1* | *c1* $|\!\} \wedge$ Θ ; $\mathcal{B}$; $\Gamma \vdash$ *v2* $\Rightarrow \{\!|$ *z2* : *b2* | *c2* $|\!\} \wedge$ *atom z* $\sharp$ Γ
using *infer-v-elims*(3)[*OF V-pair*(3)] **by** *metis*
moreover hence *atom z* $\sharp$ (*V-pair v1 v2*) **by** *simp*
moreover obtain *b* **where** *b* = *B-pair b1 b2* **using** *zbc* **by** *auto*
ultimately show *?case* **by** *fast*
next
case (*V-cons s dc v*)
thm *infer-v-elims*
obtain *x* **and** *b* **and** *c* **and** *z* **and** *c'* **and** *dclist* **and** *z'* **where**
$\tau = (\{\!|\ z : \textit{B-id s}\ |\ \textit{CE-val}\ (\textit{V-var z})\ ==\ \textit{CE-val}\ (\textit{V-cons s dc v})\ |\!\}) \wedge$
AF-typedef s dclist $\in$ *set* $\Theta \wedge$ (*dc*, $\{\!|$ *x* : *b* | *c* $|\!\}$) $\in$ *set dclist* $\wedge$ Θ ; $\mathcal{B}$; $\Gamma \vdash$ *v* $\Rightarrow \{\!|$ *z'* : *b* | *c'* $|\!\} \wedge$ Θ ; $\mathcal{B}$; $\Gamma \vdash \{\!|$ *z'* : *b* | *c'* $|\!\} \lesssim \{\!|$ *x* : *b* | *c* $|\!\} \wedge$ *atom z* $\sharp$ *v* $\wedge$ *atom z* $\sharp$ Γ
using *infer-v-elims*(4)[*OF V-cons*(2)] **by** *metis*
moreover hence *atom z* $\sharp$ (*V-cons s dc v*) **using**
Un-commute b.supp(3) *fresh-def sup-bot.right-neutral supp-b-empty v.supp*(4) *pure-supp* **by** *metis*
ultimately show *?case* **by** *metis*
next
case (*V-consp s dc bc v*)
from *V-consp*(2) **show** *?case* **proof**(*nominal-induct V-consp s dc bc v* τ *rule*:*infer-v.strong-induct*)
case (*infer-v-conspI bv dclist* Θ *tc* $\mathcal{B}$ Γ *tv z*)
moreover hence *atom z* $\sharp$ (*V-consp s dc bc v*) **unfolding** *v.fresh* **using** *pure-fresh fresh-prodN* $*$ **by** *metis*
ultimately show *?case* **using** *fresh-prodN* **by** *metis*
qed
qed

lemma *infer-v-form2*:
fixes *v*::*v*
assumes Θ ; $\mathcal{B}$; $\Gamma \vdash$ *v* $\Rightarrow$ ($\{\!|$ *z* : *b* | *c* $|\!\}$) **and** *atom z* $\sharp$ *v*
shows *c* = *C-eq* (*CE-val* (*V-var z*)) (*CE-val v*)
using *assms*
proof −
obtain *z'* **and** *b'* **where** ($\{\!|$ *z* : *b* | *c* $|\!\}$) = ($\{\!|$ *z'* : *b'* | *CE-val* (*V-var z'*) == *CE-val v* $|\!\}$) $\wedge$ *atom z'* $\sharp$ *v*
using *infer-v-form assms* **by** *meson*
thus *?thesis* **using** *Abs1-eq-iff*(3) τ.*eq-iff type-e-eq*
by (*metis assms*(2) *ce.fresh*(1))
qed

lemma *infer-v-form3*:
fixes *v*::*v*
assumes Θ ; $\mathcal{B}$; $\Gamma \vdash$ *v* $\Rightarrow \tau$ **and** *atom z* $\sharp$ (*v*,Γ)
shows Θ ; $\mathcal{B}$; $\Gamma \vdash$ *v* $\Rightarrow \{\!|$ *z* : *b-of* τ | *C-eq* (*CE-val* (*V-var z*)) (*CE-val v*)$|\!\}$
proof −
obtain *z'* **and** *b'* **where** $\tau = \{\!|$ *z'* : *b'* | *C-eq* (*CE-val* (*V-var z'*)) (*CE-val v*)$|\!\} \wedge$ *atom z'* $\sharp$ *v* $\wedge$ *atom z'* $\sharp$ Γ **using** *infer-v-form assms* **by** *metis*
moreover hence $\{\!|$ *z'* : *b'* | *C-eq* (*CE-val* (*V-var z'*)) (*CE-val v*)$|\!\} = \{\!|$ *z* : *b'* | *C-eq* (*CE-val* (*V-var z*)) (*CE-val v*)$|\!\}$
using *assms type-e-eq fresh-Pair ce.fresh* **by** *auto*
ultimately show *?thesis* **using** *b-of.simps assms* **by** *auto*

qed

lemma *infer-v-form4*:
fixes *v*::*v*
assumes Θ ; $\mathcal{B}$; Γ ⊢ *v* ⇒ τ **and** *atom z* ♯ (*v*,Γ) **and** *b* = *b-of* τ
shows Θ ; $\mathcal{B}$; Γ ⊢ *v* ⇒ {| *z* : *b* | *C-eq* (*CE-val* (*V-var z*)) (*CE-val v*)|}
using *assms infer-v-form3* **by** *simp*

lemma *infer-v-v-wf*:
fixes *v*::*v*
shows Θ; $\mathcal{B}$; *G* ⊢ *v* ⇒ τ ⟹ Θ ; $\mathcal{B}$; *G* ⊢$_{wf}$ *v* : (*b-of* τ)
proof(*induct rule*: *infer-v.induct*)
case (*infer-v-litI* Θ $\mathcal{B}$ Γ *l* τ)
hence *b-of* τ = *base-for-lit l* **using** *infer-l-form3 b-of.simps* **by** *metis*
then show *?case* **using** *wfV-litI infer-l-wf infer-v-litI wfG-b-weakening*
by (*metis fempty-fsubsetI*)
next
case (*infer-v-conspI s bv dclist* Θ *dc tc* $\mathcal{B}$ Γ *v tv b z*)
obtain *z1 b1 c1* **where** *t*:*tc* = {| *z1* : *b1* | *c1* |} **using** *obtain-fresh-z* **by** *metis*
show *?case* **unfolding** *b-of.simps* **proof**(*rule wfV-conspI*)
show ‹*AF-typedef-poly s bv dclist* ∈ *set* Θ› **using** *infer-v-conspI* **by** *auto*
show ‹(*dc*, {| *z1* : *b1* | *c1* |}) ∈ *set dclist*› **using** *infer-v-conspI t* **by** *auto*
show ‹ Θ ; $\mathcal{B}$ ⊢$_{wf}$ *b* › **using** *infer-v-conspI* **by** *auto*
show ‹*atom bv* ♯ (Θ, $\mathcal{B}$, Γ, *b*, *v*)› **using** *infer-v-conspI* **by** *auto*
have *b1*[*bv*::=*b*]$_{bb}$ = *b-of tv* **using** *subtype-eq-base2*[*OF infer-v-conspI*(*5*)] *b-of.simps t subst-tb.simps*
by *auto*
thus ‹ Θ ; $\mathcal{B}$; Γ ⊢$_{wf}$ *v* : *b1*[*bv*::=*b*]$_{bb}$ › **using** *infer-v-conspI* **by** *auto*
qed
qed(*auto simp add*: *wfC-elims wf-intros*)+

lemma *infer-v-t-form-wf*:
assumes *wfB* Θ $\mathcal{B}$ *b* **and** *wfV* Θ $\mathcal{B}$ Γ *v b* **and** *atom z* ♯ Γ
shows *wfT* Θ $\mathcal{B}$ Γ {| *z* : *b* | *C-eq* (*CE-val* (*V-var z*)) (*CE-val v*)|}
using *wfT-v-eq assms* **by** *auto*

lemma *infer-v-t-wf*:
fixes *v*::*v*
assumes Θ ; $\mathcal{B}$; *G* ⊢ *v* ⇒ τ
shows *wfT* Θ $\mathcal{B}$ *G* τ ∧ *wfB* Θ $\mathcal{B}$ (*b-of* τ)
proof −
obtain *z* **and** *b* **where** τ = {| *z* : *b* | *CE-val* (*V-var z*) == *CE-val v* |} ∧ *atom z* ♯ *v* ∧ *atom z* ♯ *G* **using** *infer-v-form assms* **by** *metis*
moreover have *wfB* Θ $\mathcal{B}$ *b* **using** *infer-v-v-wf b-of.simps wfX-wfB*(*1*) *assms*
using *calculation* **by** *fastforce*
ultimately show *wfT* Θ $\mathcal{B}$ *G* τ ∧ *wfB* Θ $\mathcal{B}$ (*b-of* τ) **using** *infer-v-v-wf infer-v-t-form-wf assms*
by *fastforce*
qed

lemma *infer-v-wf*:
fixes *v*::*v*
assumes Θ ; $\mathcal{B}$; *G* ⊢ *v* ⇒ τ
shows Θ ; $\mathcal{B}$; *G* ⊢$_{wf}$ *v* : (*b-of* τ) **and** *wfT* Θ $\mathcal{B}$ *G* τ **and** *wfTh* Θ **and** *wfG* Θ $\mathcal{B}$ *G*

proof −
show Θ ; $\mathcal{B}$; $G \vdash_{wf} v$: *b-of* τ **using** *infer-v-v-wf assms* **by** *auto*
show Θ ; $\mathcal{B}$; $G \vdash_{wf} \tau$ **using** *infer-v-t-wf assms* **by** *auto*
thus Θ ; $\mathcal{B} \vdash_{wf} G$ **using** *wfX-wfY* **by** *auto*
thus $\vdash_{wf} \Theta$ **using** *wfX-wfY* **by** *auto*
qed

lemma *check-bool-options*:
assumes Θ ; $\mathcal{B}$; $\Gamma \vdash v \Leftarrow \{\!|\, z : B\text{-}bool \mid TRUE \,|\!\}$ **and** *supp* $v = \{\}$
shows $v = V\text{-}lit\ L\text{-}true \vee v = V\text{-}lit\ L\text{-}false$
proof −
obtain *t1* **where** Θ ; $\mathcal{B}$; $\Gamma \vdash t1 \lesssim \{\!|\, z : B\text{-}bool \mid TRUE \,|\!\} \wedge \Theta$; $\mathcal{B}$; $\Gamma \vdash v \Rightarrow t1$ **using** *check-v-elims*
using *assms* **by** *blast*
thus *?thesis* **using** *infer-bool-options assms*
by (*metis* τ.*exhaust b-of.simps subtype-eq-base2*)
qed

lemma *check-v-wf*:
fixes *v*::*v* **and** Γ::Γ **and** τ::τ
assumes Θ ; $\mathcal{B}$; $\Gamma \vdash v \Leftarrow \tau$
shows Θ ; $\mathcal{B} \vdash_{wf} \Gamma$ **and** Θ ; $\mathcal{B}$;$\Gamma \vdash_{wf} v$: *b-of* τ **and** Θ ; $\mathcal{B}$;$\Gamma \vdash_{wf} \tau$
proof −
obtain τ' **where** $*$: Θ ; $\mathcal{B}$; $\Gamma \vdash \tau' \lesssim \tau \wedge \Theta$; $\mathcal{B}$; $\Gamma \vdash v \Rightarrow \tau'$ **using** *check-v-elims assms* **by** *auto*
thus Θ ; $\mathcal{B} \vdash_{wf} \Gamma$ **and** Θ ; $\mathcal{B}$;$\Gamma \vdash_{wf} v$: *b-of* τ **and** Θ ; $\mathcal{B}$; $\Gamma \vdash_{wf} \tau$
using *infer-v-wf infer-v-v-wf subtype-eq-base2* $*$ *subtype-wf* **by** *metis+*
qed

lemma *infer-v-form-fresh*:
fixes *v*::*v* **and** *t*::′*a*::*fs*
assumes Θ ; $\mathcal{B}$; $\Gamma \vdash v \Rightarrow \tau$
shows $\exists z\ b.\ \tau = \{\!|\, z : b \mid C\text{-}eq\ (CE\text{-}val\ (V\text{-}var\ z))\ (CE\text{-}val\ v) |\!\} \wedge atom\ z \sharp (t,v)$
proof −
obtain z' **and** b' **where** $\tau = \{\!|\, z' : b' \mid C\text{-}eq\ (CE\text{-}val\ (V\text{-}var\ z'))\ (CE\text{-}val\ v) |\!\}$ **using** *infer-v-form assms* **by** *blast*
moreover then obtain *z* **and** *b* **and** *c* **where** $\tau = \{\!|\, z : b \mid c \,|\!\} \wedge atom\ z \sharp (t,v)$ **using** *obtain-fresh-z* **by** *metis*
ultimately have $\tau = \{\!|\, z : b \mid C\text{-}eq\ (CE\text{-}val\ (V\text{-}var\ z))\ (CE\text{-}val\ v) |\!\} \wedge atom\ z \sharp (t,v)$
using *assms infer-v-form2* **by** *auto*
thus *?thesis* **by** *blast*
qed

More generally, if support of a term is empty then any G will do

lemma *infer-v-form-consp*:
assumes Θ ; $\mathcal{B}$; $\Gamma \vdash V\text{-}consp\ s\ dc\ b\ v \Rightarrow \tau$
shows *b-of* τ = *B-app s b*
using *assms* **proof**(*nominal-induct V-consp s dc b v* τ *rule*: *infer-v.strong-induct*)
case (*infer-v-conspI bv dclist* Θ *tc* $\mathcal{B}$ Γ *tv z*)
then show *?case* **using** *b-of.simps* **by** *metis*
qed

lemma *infer-v-uniqueness-rig*:
fixes $x::x$ **and** $c::c$
assumes *infer-v P B G v* τ **and** *infer-v P B* (*replace-in-g G x c'*) *v* τ'
shows $\tau = \tau'$
using *assms*
proof(*nominal-induct v arbitrary*: τ' τ *rule*: *v.strong-induct*)
case (*V-lit l*)
hence *infer-l l* τ **and** *infer-l l* τ' **using** *assms*(1) *infer-v-elims*(2) **by** *auto*
then show *?case* **using** *infer-l-uniqueness* **by** *presburger*
next
case (*V-var y*)

obtain *b* **and** *c* **where** *bc*: *Some* (*b*,*c*) = *lookup G y*
using *assms*(1) *infer-v-elims*(2) **using** *V-var.prems*(1) *lookup-iff* **by** *force*
then obtain *c''* **where** *bc'*:*Some* (*b*,*c''*) = *lookup* (*replace-in-g G x c'*) *y*
using *lookup-in-rig* **by** *blast*

obtain *z* **where** τ = ({| *z* : *b* | *C-eq* (*CE-val* (*V-var z*)) (*CE-val* (*V-var y*)) |}) **using** *infer-v-elims*(1)[*of P B G y* τ] *V-var*
bc option.inject prod.inject lookup-in-g **by** *metis*
moreover obtain *z'* **where** τ' = ({| *z'* : *b* | *C-eq* (*CE-val* (*V-var z'*)) (*CE-val* (*V-var y*)) |}) **using** *infer-v-elims*(1)[*of P B - y* τ'] *V-var*
option.inject prod.inject lookup-in-rig **by** (*metis bc'*)
ultimately show *?case* **using** *type-e-eq*
by (*metis V-var.prems*(1) *V-var.prems*(2) τ.*eq-iff ce.fresh*(1) *finite.emptyI fresh-atom-at-base fresh-finite-insert infer-v-elims*(1) *v.fresh*(2))
next
case (*V-pair v1 v2*)
obtain *z* **and** *z1* **and** *z2* **and** *b1* **and** *b2* **and** *c1* **and** *c2* **where**
t1: τ = ({| *z* : *B-pair b1 b2* | *CE-val* (*V-var z*) == *CE-val* (*V-pair v1 v2*) |}) $\wedge$ *atom z* $\sharp$ (*v1*, *v2*) $\wedge$ *P* ; *B* ; *G* $\vdash$ *v1* $\Rightarrow$ {| *z1* : *b1* | *c1* |} $\wedge$ *P* ; *B* ; *G* $\vdash$ *v2* $\Rightarrow$ {| *z2* : *b2* | *c2* |}
using *infer-v-elims*(3)[*OF V-pair*(3)] **by** *metis*
moreover obtain *z'* **and** *z1'* **and** *z2'* **and** *b1'* **and** *b2'* **and** *c1'* **and** *c2'* **where**
t2: τ' = ({| *z'* : *B-pair b1' b2'* | *CE-val* (*V-var z'*) == *CE-val* (*V-pair v1 v2*) |}) $\wedge$ *atom z'* $\sharp$ (*v1*, *v2*) $\wedge$ *P* ; *B* ; (*replace-in-g G x c'*) $\vdash$ *v1* $\Rightarrow$ {| *z1'* : *b1'* | *c1'* |} $\wedge$ *P* ; *B* ; (*replace-in-g G x c'*) $\vdash$ *v2* $\Rightarrow$ {| *z2'* : *b2'* | *c2'* |}
using *infer-v-elims*(3)[*OF V-pair*(4)] **by** *metis*
ultimately have *b1* = *b1'* $\wedge$ *b2* = *b2'* **using** *V-pair.hyps*(1) *V-pair.hyps*(2) τ.*eq-iff* **by** *blast*
then show *?case* **using** *t1 t2* **by** *simp*
next
case (*V-cons s dc v*)
obtain *x* **and** *z* **and** *b* **and** *c* **and** *dclist* **where** *t1*: τ = ({| *z* : *B-id s* | *CE-val* (*V-var z*) == *CE-val* (*V-cons s dc v*) |}) $\wedge$ *AF-typedef s dclist* $\in$ *set P* $\wedge$
(*dc*, {| *x* : *b* | *c* |}) $\in$ *set dclist* $\wedge$ *atom z* $\sharp$ *v*
using *infer-v-elims*(4)[*OF V-cons*(2)] **by** *metis*
moreover obtain *x'* **and** *z'* **and** *b'* **and** *c'* **and** *dclist'* **where** *t2*: τ' = ({| *z'* : *B-id s* | *CE-val* (*V-var z'*) == *CE-val* (*V-cons s dc v*) |}) $\wedge$ *AF-typedef s dclist'* $\in$ *set P* $\wedge$ (*dc*, {| *x'* : *b'* | *c'* |}) $\in$ *set dclist'* $\wedge$ *atom z'* $\sharp$ *v*
using *infer-v-elims*(4)[*OF V-cons*(3)] **by** *metis*
moreover have *a*: *AF-typedef s dclist'* $\in$ *set P* **and** *b*:(*dc*,{| *x'* : *b'* | *c'* |}) $\in$ *set dclist'* **and** *c*:*AF-typedef s dclist* $\in$ *set P* **and**
d:(*dc*, {| *x* : *b* | *c* |}) $\in$ *set dclist* **using** *t1 t2* **by** *auto*

ultimately have $\{\!|\ x : b\ \mid c\ |\!\} = \{\!|\ x' : b'\ \mid c'\ |\!\}$ **using** *wfTh-dc-t-unique infer-v-wf V-cons* **by** *metis*

moreover have *atom z* $\sharp$ *CE-val* (*V-cons s dc v*) $\wedge$ *atom z'* $\sharp$ *CE-val* (*V-cons s dc v*)
using *e.fresh*(*1*) *v.fresh*(*4*) *t1 t2 pure-fresh* **by** *auto*
ultimately have ($\{\!|\ z$: *B-id s* $\mid$ *CE-val* (*V-var z*) == *CE-val* (*V-cons s dc v*) $|\!\}$) = ($\{\!|\ z'$: *B-id s* $\mid$ *CE-val* (*V-var z'*) == *CE-val* (*V-cons s dc v*) $|\!\}$)
using *type-e-eq* **by** *metis*
thus *?case* **using** *t1 t2* **by** *simp*
next
case (*V-consp s dc b v*)
from *V-consp*(*2*) *V-consp* **show** *?case* **proof**(*nominal-induct V-consp s dc b v* τ *arbitrary*: *v rule*:*infer-v.strong-induct*)

case (*infer-v-conspI bv dclist* Θ *tc* $\mathcal{B}$ Γ *v tv z*)

obtain *z3* **and** *b3* **where** $*:\tau' = \{\!|\ z3 : b3 \mid [\ [\ z3\ ]^v\ ]^{ce}$ == $[$ *V-consp s dc b v* $]^{ce}\ |\!\} \wedge$ *atom z3* $\sharp$ *V-consp s dc b v*
using *infer-v-form*[*OF* $\langle\Theta$; $\mathcal{B}$; $\Gamma[x\longmapsto c'] \vdash$ *V-consp s dc b v* $\Rightarrow \tau'\rangle$] **by** *metis*
moreover then have *b3* = *B-app s b* **using** *infer-v-form-consp b-of.simps* $*$ *infer-v-conspI* **by** *metis*

moreover have $\{\!|\ z3 : B\text{-}app\ s\ b \mid [\ [\ z3\ ]^v\ ]^{ce}$ == $[$ *V-consp s dc b v* $]^{ce}\ |\!\} = \{\!|\ z : B\text{-}app\ s\ b \mid [\ [\ z\ ]^v\ ]^{ce}$ == $[$ *V-consp s dc b v* $]^{ce}\ |\!\}$
proof −
have *atom z3* $\sharp$ [*V-consp s dc b v*]^{ce} **using** $*$ *ce.fresh* **by** *auto*
moreover have *atom z* $\sharp$ [*V-consp s dc b v*]^{ce} **using** $*$ *infer-v-conspI ce.fresh v.fresh pure-fresh* **by** *metis*
ultimately show *?thesis* **using** *type-e-eq infer-v-conspI v.fresh ce.fresh* **by** *metis*
qed
ultimately show *?case* **using** $*$ **by** *auto*
qed

qed

lemma *infer-v-uniqueness*:
assumes *infer-v P B G v* τ **and** *infer-v P B G v* τ'
shows $\tau = \tau'$
proof −
obtain *x*::*x* **where** *atom x* $\sharp$ *G* **using** *obtain-fresh* **by** *metis*
hence *G* [*x* $\longmapsto$ *C-true*] = *G* **using** *replace-in-g-forget assms infer-v-wf* **by** *fast*
thus *?thesis* **using** *infer-v-uniqueness-rig assms* **by** *metis*
qed

lemma *infer-v-tid-form*:
fixes *v*::*v*
assumes Θ ; *B* ; Γ $\vdash$ *v* $\Rightarrow \{\!|\ z$: *B-id tid* $\mid$ *c* $|\!\}$ **and** *AF-typedef tid dclist* $\in$ *set* Θ **and** *supp v* = {}
shows $\exists$ *dc v' t*. *v* = *V-cons tid dc v'* $\wedge$ (*dc* , *t*) $\in$ *set dclist*
using *assms* **proof**(*nominal-induct v* $\{\!|\ z$: *B-id tid* $\mid$ *c* $|\!\}$ *rule*: *infer-v.strong-induct*)
case (*infer-v-varI* Θ $\mathcal{B}$ *c x z*)
then show *?case* **using** *v.supp supp-at-base* **by** *auto*
next
case (*infer-v-litI* Θ $\mathcal{B}$ *l*)

then show *?case* **by** *auto*
next
case (*infer-v-consI dclist1* Θ *dc x b c* $\mathcal{B}$ Γ *v z' c' z*)
hence *supp v* = {} **using** *v.supp* **by** *simp*
then obtain *dca* **and** *v'* **where** ∗:*V-cons tid dc v = V-cons tid dca v'* **using** *infer-v-consI* **by** *auto*
hence *dca = dc* **using** *v.eq-iff*(*4*) **by** *auto*
hence *V-cons tid dc v = V-cons tid dca v'* ∧ (*dca*, {| *x* : *b* | *c* |}) ∈ *set dclist1* **using** *infer-v-consI* ∗ **by** *auto*
moreover have *dclist = dclist1* **using** *wfTh-dclist-unique infer-v-consI wfX-wfY* ‹*dca=dc*›
proof −
show *?thesis*
by (*meson* ‹*AF-typedef tid dclist1* ∈ *set* Θ› ‹Θ ; $\mathcal{B}$; Γ ⊢ *v* ⇒ {| *z'* : *b* | *c'* |}› *infer-v-consI.prems infer-v-wf*(*4*) *wfTh-dclist-unique wfX-wfY*)
qed
ultimately show *?case* **by** *auto*
qed

lemma *check-v-tid-form*:
assumes Θ ; *B* ; Γ ⊢ *v* ⇐ {| *z* : *B-id tid* | *TRUE* |} **and** *AF-typedef tid dclist* ∈ *set* Θ **and** *supp v* = {}
shows ∃ *dc v' t*. *v* = *V-cons tid dc v'* ∧ (*dc* , *t*) ∈ *set dclist*
using *assms* **proof**(*nominal-induct v* {| *z* : *B-id tid* | *TRUE* |} *rule*: *check-v.strong-induct*)
case (*check-v-subtypeI* Θ $\mathcal{B}$ Γ *τ1 v*)
then obtain *z* **and** *c* **where** *τ1* = {| *z* : *B-id tid* | *c* |} **using** *subtype-eq-base2 b-of.simps*
by (*metis obtain-fresh-z2*)
then show *?case* **using** *infer-v-tid-form check-v-subtypeI* **by** *simp*
qed

lemma *check-v-num-leq*:
fixes *n*::*int* **and** Γ::Γ
assumes *0* ≤ *n* ∧ *n* ≤ *int* (*length v*) **and** ⊢$_{wf}$ Θ **and** Θ ; {||} ⊢$_{wf}$ Γ
shows Θ ; {||} ; Γ ⊢ [*L-num n*]^{v} ⇐ {| *z* : *B-int* | ([*leq* [[*L-num 0*]^{v}]^{ce} [[*z*]^{v}]^{ce}]^{ce} == [[*L-true*]^{v}]^{ce})
AND ([*leq* [[*z*]^{v}]^{ce} [|| [[*L-bitvec v*]^{v}]^{ce} ||]^{ce}]^{ce} == [[*L-true*]^{v}]^{ce}) |}
proof −
have Θ ; {||} ; Γ ⊢ [*L-num n*]^{v} ⇒ {| *z* : *B-int* | [[*z*]^{v}]^{ce} == [[*L-num n*]^{v}]^{ce} |}
using *infer-v-litI infer-natI wfG-nilI assms* **by** *auto*
thus *?thesis* **using** *subtype-range*[*OF assms*(*1*)] *assms check-v-subtypeI* **by** *metis*
qed

lemma *check-int*:
assumes *check-v* Θ $\mathcal{B}$ Γ *v* ({| *z* : *B-int* | *c* |}) **and** *supp v* = {}
shows ∃ *n*. *V-lit* (*L-num n*) = *v*
using *assms infer-int check-v-elims* **by** (*metis b-of.simps infer-v-form subtype-eq-base2*)

definition *sble* :: Θ ⇒ Γ ⇒ *bool* **where**
sble Θ Γ = (∃ *i*. *i* |= Γ ∧ Θ ; Γ ⊢ *i*)

lemma *check-v-range*:

assumes Θ ; $\{||\}$; $\Gamma \vdash$ *v2* $\Leftarrow$ $\{\!|$ *z* : *B-int* | [*leq* [[*L-num 0*]^v]^{ce} [[*z*]^v]^{ce}]^{ce} == [[*L-true*]^v]^{ce} *AND*
[*leq* [[*z*]^v]^{ce} [|| [*v1*]^{ce} ||]^{ce}]^{ce} == [[*L-true*]^v]^{ce} $|\!\}$
(**is** Θ ; *?B* ; $\Gamma \vdash$ *v2* $\Leftarrow$ $\{\!|$ *z* : *B-int* | *?c1* $|\!\}$)
and *v1* = *V-lit* (*L-bitvec bv*) $\wedge$ *v2* = *V-lit* (*L-num n*) **and** *atom z* $\sharp$ Γ **and** *sble* Θ Γ
shows $0 \leq n \wedge n \leq$ *int* (*length bv*)
proof −
have Θ ; *?B* ; $\Gamma \vdash$ $\{\!|$ *z* : *B-int* | [[*z*]^v]^{ce} == [[*L-num n*]^v]^{ce} $|\!\} \lesssim \{\!|$ *z* : *B-int* | *?c1* $|\!\}$
using *check-v-elims assms*
by (*metis infer-l-uniqueness infer-natI infer-v-elims*(*2*))
moreover have *atom z* $\sharp$ Γ **using** *fresh-GNil assms* **by** *simp*
ultimately have Θ ; *?B* ; ((z, *B-int*, [[*z*]^v]^{ce} == [[*L-num n*]^v]^{ce}) $\#_\Gamma$ Γ) $\models$ *?c1*
using *subtype-valid-simple* **by** *auto*
thus *?thesis* **using** *assms valid-range-length-inv check-v-wf wfX-wfY sble-def* **by** *metis*
qed

12.4 Expressions

lemma *infer-e-plus*[*elim*]:
fixes *v1*::*v* **and** *v2*::*v*
assumes Θ ; Φ ; $\mathcal{B}$; Γ ; $\Delta \vdash$ *AE-op Plus v1 v2* $\Rightarrow \tau$
shows $\exists z$. ($\{\!|$ *z* : *B-int* | *C-eq* (*CE-val* (*V-var z*)) (*CE-op Plus* [*v1*]^{ce} [*v2*]^{ce}) $|\!\} = \tau$)
using *infer-e-elims assms* **by** *metis*

lemma *infer-e-leq*[*elim*]:
assumes Θ ; Φ ; $\mathcal{B}$; Γ ; $\Delta \vdash$ *AE-op LEq v1 v2* $\Rightarrow \tau$
shows $\exists z$. ($\{\!|$ *z* : *B-bool* | *C-eq* (*CE-val* (*V-var z*)) (*CE-op LEq* [*v1*]^{ce} [*v2*]^{ce}) $|\!\} = \tau$)
using *infer-e-elims assms* **by** *metis*

lemmas *subst-defs* = *subst-b-b-def subst-b-c-def subst-b-τ-def subst-v-v-def subst-v-c-def subst-v-τ-def*

lemma *infer-e-e-wf*:
fixes *e*::*e*
assumes Θ ; Φ ; $\mathcal{B}$; Γ ; $\Delta \vdash$ *e* $\Rightarrow \tau$
shows Θ ; Φ ; $\mathcal{B}$; Γ ; $\Delta \vdash_{wf}$ *e* : *b-of* τ
using *assms* **proof**(*nominal-induct* τ *avoiding*: τ *rule*: *infer-e.strong-induct*)
case (*infer-e-valI* Θ $\mathcal{B}$ Γ Δ' Φ *v* τ)
then show *?case* **using** *infer-v-v-wf wf-intros* **by** *metis*
next
case (*infer-e-plusI* Θ $\mathcal{B}$ Γ Δ' Φ *v1 z1 c1 v2 z2 c2 z3*)
then show *?case* **using** *b-of.simps infer-v-v-wf wf-intros* **by** *metis*
next
case (*infer-e-leqI* Θ $\mathcal{B}$ Γ Δ' *v1 z1 c1 v2 z2 c2 z3*)
then show *?case* **using** *b-of.simps infer-v-v-wf wf-intros* **by** *metis*
next
case (*infer-e-appI* Θ $\mathcal{B}$ Γ Δ Φ *f x b c* τ' *s'* *v* τ'')
have Θ ; Φ ; $\mathcal{B}$; Γ ; $\Delta \vdash_{wf}$ *AE-app f v* : *b-of* τ' **proof**
show ‹ Θ $\vdash_{wf}$ Φ › **using** *infer-e-appI* **by** *auto*
show ‹ Θ ; $\mathcal{B}$; $\Gamma \vdash_{wf}$ Δ › **using** *infer-e-appI* **by** *auto*
show ‹*Some* (*AF-fundef f* (*AF-fun-typ-none* (*AF-fun-typ x b c* τ' *s'*))) = *lookup-fun* Φ *f*› **using** *infer-e-appI* **by** *auto*
show Θ ; $\mathcal{B}$;$\Gamma \vdash_{wf}$ *v* : *b* **using** *infer-e-appI check-v-wf b-of.simps* **by** *metis*

qed
moreover have *b-of* $\tau' = b\text{-}of\ (\tau'[x::=v]_v)$ **using** *subst-tbase-eq subst-v-τ-def* **by** *auto*
ultimately show *?case* **using** *infer-e-appI subst-v-c-def subst-b-τ-def* **by** *auto*
next
case (*infer-e-appPI* $\Theta\ \mathcal{B}\ \Gamma\ \Delta\ \Phi\ b'\ f\ bv\ x\ b\ c\ \tau''\ s'\ v\ \tau'$)

have $\Theta\ ;\ \Phi\ ;\ \mathcal{B}\ ;\ \Gamma\ ;\ \Delta \vdash_{wf} AE\text{-}appP\ f\ b'\ v : (b\text{-}of\ \tau'')[bv::=b']_b$ **proof**
show $\langle\ \Theta\ \vdash_{wf} \Phi\ \rangle$ **using** *infer-e-appPI* **by** *auto*
show $\langle\ \Theta\ ;\ \mathcal{B}\ ;\ \Gamma \vdash_{wf} \Delta\ \rangle$ **using** *infer-e-appPI* **by** *auto*
show $\langle Some\ (AF\text{-}fundef\ f\ (AF\text{-}fun\text{-}typ\text{-}some\ bv\ (AF\text{-}fun\text{-}typ\ x\ b\ c\ \tau''\ s'))) = lookup\text{-}fun\ \Phi\ f\rangle$ **using** $*$ *infer-e-appPI* **by** *metis*
show $\Theta\ ;\ \mathcal{B}\ \vdash_{wf} b'$ **using** *infer-e-appPI* **by** *auto*
show $\Theta\ ;\ \mathcal{B}\ ;\Gamma \vdash_{wf} v : (b[bv::=b']_b)$ **using** *infer-e-appPI check-v-wf b-of.simps subst-b-b-def* **by** *metis*
have $atom\ bv \sharp (b\text{-}of\ \tau'')[bv::=b']_{bb}$ **using** *fresh-subst-if subst-b-b-def infer-e-appPI* **by** *metis*
thus $atom\ bv \sharp (\Phi,\ \Theta,\ \mathcal{B},\ \Gamma,\ \Delta,\ b',\ v,\ (b\text{-}of\ \tau'')[bv::=b']_b)$ **using** *infer-e-appPI fresh-prodN subst-b-b-def* **by** *metis*
qed
moreover have $b\text{-}of\ \tau' = (b\text{-}of\ \tau'')[bv::=b']_b$
using $\langle \tau''[bv::=b']_b[x::=v]_v = \tau'\rangle$ *b-of-subst-bb-commute subst-tbase-eq subst-b-b-def subst-v-τ-def subst-b-τ-def* **by** *auto*
ultimately show *?case* **using** *infer-e-appI* **by** *auto*
next
case (*infer-e-fstI* $\Theta\ \mathcal{B}\ \Gamma\ \Delta'\ \Phi\ v\ z'\ b1\ b2\ c\ z$)
then show *?case* **using** *b-of.simps infer-v-v-wf wf-intros* **by** *metis*
next
case (*infer-e-sndI* $\Theta\ \mathcal{B}\ \Gamma\ \Delta'\ \Phi\ v\ z'\ b1\ b2\ c\ z$)
then show *?case* **using** *b-of.simps infer-v-v-wf wf-intros* **by** *metis*
next
case (*infer-e-lenI* $\Theta\ \mathcal{B}\ \Gamma\ \Delta'\ \Phi\ v\ z'\ c\ z$)
then show *?case* **using** *b-of.simps infer-v-v-wf wf-intros* **by** *metis*
next
case (*infer-e-mvarI* $\Theta\ \Gamma\ \Phi\ \Delta\ u\ \tau$)
then show *?case* **using** *b-of.simps infer-v-v-wf wf-intros* **by** *metis*
next
case (*infer-e-concatI* $\Theta\ \mathcal{B}\ \Gamma\ \Delta'\ \Phi$ *v1 z1 c1 v2 z2 c2 z3*)
then show *?case* **using** *b-of.simps infer-v-v-wf wf-intros* **by** *metis*
next
case (*infer-e-splitI* $\Theta\ \mathcal{B}\ \Gamma\ \Delta\ \Phi$ *v1 z1 c1 v2 z2 z3*)
have $\Theta\ ;\ \Phi\ ;\ \mathcal{B}\ ;\ \Gamma\ ;\ \Delta \vdash_{wf} AE\text{-}split\ v1\ v2 : B\text{-}pair\ B\text{-}bitvec\ B\text{-}bitvec$
proof
show $\Theta\ \vdash_{wf} \Phi$ **using** *infer-e-splitI* **by** *auto*
show $\Theta\ ;\ \mathcal{B}\ ;\ \Gamma \vdash_{wf} \Delta$ **using** *infer-e-splitI* **by** *auto*
show $\Theta\ ;\ \mathcal{B}\ ;\ \Gamma \vdash_{wf} v1 : B\text{-}bitvec$ **using** *infer-e-splitI b-of.simps infer-v-wf* **by** *metis*
show $\Theta\ ;\ \mathcal{B}\ ;\ \Gamma \vdash_{wf} v2 : B\text{-}int$ **using** *infer-e-splitI b-of.simps check-v-wf* **by** *metis*
qed
then show *?case* **using** *b-of.simps* **by** *auto*
qed

lemma *infer-e-t-wf*:
fixes $e::e$ **and** $\Gamma::\Gamma$ **and** $\tau::\tau$ **and** $\Delta::\Delta$ **and** $\Phi::\Phi$
assumes $\Theta\ ;\ \Phi\ ;\ \mathcal{B}\ ;\ \Gamma\ ;\ \Delta \vdash e \Rightarrow \tau$

shows Θ ; $\mathcal{B}$;$\Gamma \vdash_{wf} \tau \wedge \Theta \vdash_{wf} \Phi$
using *assms* **proof**(*induct rule*: *infer-e.induct*)
case (*infer-e-valI* Θ $\mathcal{B}$ Γ Δ' Φ v τ)
then show *?case* **using** *infer-v-t-wf* **by** *auto*
next
case (*infer-e-plusI* Θ $\mathcal{B}$ Γ Δ Φ *v1 z1 c1 v2 z2 c2 z3*)
hence Θ ; $\mathcal{B}$; $\Gamma \vdash_{wf}$ *CE-op Plus* $[v1]^{ce}$ $[v2]^{ce}$: *B-int* **using** *wfCE-plusI wfD-emptyI wfPhi-emptyI infer-v-v-wf wfCE-valI*
by (*metis b-of.simps infer-v-wf*)
then show *?case* **using** *wfT-e-eq infer-e-plusI* **by** *auto*
next
case (*infer-e-leqI* Θ $\mathcal{B}$ Γ Δ Φ *v1 z1 c1 v2 z2 c2 z3*)
hence Θ ; $\mathcal{B}$; $\Gamma \vdash_{wf}$ *CE-op LEq* $[v1]^{ce}$ $[v2]^{ce}$: *B-bool* **using** *wfCE-leqI wfD-emptyI wfPhi-emptyI infer-v-v-wf wfCE-valI*
by (*metis b-of.simps infer-v-wf*)
then show *?case* **using** *wfT-e-eq infer-e-leqI* **by** *auto*
next
case (*infer-e-appI* Θ $\mathcal{B}$ Γ Δ Φ *f x b c* τ *s'* v τ')
show *?case* **proof**
show $\Theta \vdash_{wf} \Phi$ **using** *infer-e-appI* **by** *auto*
show Θ ; $\mathcal{B}$; $\Gamma \vdash_{wf} \tau'$ **proof** −
have $*$: Θ ; $\mathcal{B}$; $\Gamma \vdash_{wf} v : b$ **using** *infer-e-appI check-v-wf*(*2*) *b-of.simps* **by** *metis*
moreover have $*$:Θ ; $\mathcal{B}$; $(x, b, c) \#_\Gamma \Gamma \vdash_{wf} \tau$ **proof**(*rule wf-weakening1*(*4*))
show ‹ Θ ; $\mathcal{B}$; $(x,b,c)\#_\Gamma$ *GNil* $\vdash_{wf} \tau$ › **using** *wfPhi-f-simple-wfT wfD-wf infer-e-appI wb-b-weakening* **by** *fastforce*
have Θ ; $\mathcal{B}$; $\Gamma \vdash_{wf} \{\!|\, x : b \mid c \,|\!\}$ **using** *infer-e-appI check-v-wf*(*3*) **by** *auto*
thus ‹ Θ ; $\mathcal{B} \vdash_{wf} (x, b, c) \#_\Gamma \Gamma$ › **using** *infer-e-appI wfT-wfC*[*THEN wfG-consI*[*rotated 3*]] $*$ *wfX-wfY wfT-wf-cons* **by** *metis*
show ‹*setG* $((x,b,c)\#_\Gamma$ *GNil*) $\subseteq$ *setG* $((x, b, c) \#_\Gamma \Gamma)$› **using** *setG.simps* **by** *auto*
qed
moreover have $((x, b, c) \#_\Gamma \Gamma)[x::=v]_{\Gamma v} = \Gamma$ **using** *subst-gv.simps* **by** *auto*

ultimately show *?thesis* **using** *infer-e-appI wf-subst1*(*4*)[*OF* $*$, *of GNil x b c* Γ v] *subst-v-*τ*-def* **by** *auto*
qed
qed
next
case (*infer-e-appPI* Θ $\mathcal{B}$ Γ Δ Φ *b' f bv x b c* τ' *s'* v τ)

have Θ ; $\mathcal{B}$; $((x, b[bv::=b']_{bb}, c[bv::=b']_{cb}) \#_\Gamma \Gamma)[x::=v]_{\Gamma v} \vdash_{wf} (\tau'[bv::=b']_b)[x::=v]_{\tau v}$
proof(*rule wf-subst*(*4*))
show ‹ Θ ; $\mathcal{B}$; $(x, b[bv::=b']_{bb}, c[bv::=b']_{cb}) \#_\Gamma \Gamma \vdash_{wf} \tau'[bv::=b']_b$ ›
proof(*rule wf-weakening1*(*4*))
have ‹ Θ ; $\{|bv|\}$; $(x,b,c)\#_\Gamma$ *GNil* $\vdash_{wf} \tau'$ › **using** *wfPhi-f-poly-wfT infer-e-appI infer-e-appPI* **by** *simp*
thus ‹ Θ ; $\mathcal{B}$; $(x,b[bv::=b']_{bb},c[bv::=b']_{cb})\#_\Gamma$ *GNil* $\vdash_{wf} \tau'[bv::=b']_b$ ›
using *wfT-subst-wfT infer-e-appPI wb-b-weakening subst-b-*τ*-def subst-v-*τ*-def* **by** *presburger*
have Θ ; $\mathcal{B}$; $\Gamma \vdash_{wf} \{\!|\, x : b[bv::=b']_{bb} \mid c[bv::=b']_{cb} \,|\!\}$
using *infer-e-appPI check-v-wf*(*3*) *subst-b-b-def subst-b-c-def* **by** *metis*
thus ‹ Θ ; $\mathcal{B} \vdash_{wf} (x, b[bv::=b']_{bb}, c[bv::=b']_{cb}) \#_\Gamma \Gamma$ ›
using *infer-e-appPI wfT-wfC*[*THEN wfG-consI*[*rotated 3*]] $*$ *wfX-wfY wfT-wf-cons wb-b-weakening* **by** *metis*

show ‹*setG* ((*x*,*b*[*bv*::=*b'*]$_{bb}$,*c*[*bv*::=*b'*]$_{cb}$)#$_\Gamma$ *GNil*) ⊆ *setG* ((*x*, *b*[*bv*::=*b'*]$_{bb}$, *c*[*bv*::=*b'*]$_{cb}$) #$_\Gamma$ Γ)›
using *setG.simps* **by** *auto*
qed
show ‹(*x*, *b*[*bv*::=*b'*]$_{bb}$, *c*[*bv*::=*b'*]$_{cb}$) #$_\Gamma$ Γ = *GNil* @ (*x*, *b*[*bv*::=*b'*]$_{bb}$, *c*[*bv*::=*b'*]$_{cb}$) #$_\Gamma$ Γ› **using** *append-g.simps* **by** *auto*
show ‹ Θ ; $\mathcal{B}$; Γ ⊢$_{wf}$ *v* :*b*[*bv*::=*b'*]$_{bb}$ › **using** *infer-e-appPI check-v-wf*(*2*) *b-of.simps subst-b-b-def* **by** *metis*
qed
moreover have ((*x*, *b*[*bv*::=*b'*]$_{bb}$, *c*[*bv*::=*b'*]$_{cb}$) #$_\Gamma$ Γ)[*x*::=*v*]$_{\Gamma v}$ = Γ **using** *subst-gv.simps* **by** *auto*
ultimately show *?case* **using** *infer-e-appPI subst-v-τ-def* **by** *simp*
next
case (*infer-e-fstI* Θ $\mathcal{B}$ Γ Δ Φ *v z' b1 b2 c z*)
hence Θ ; $\mathcal{B}$; Γ ⊢$_{wf}$ *CE-fst* [*v*]^{ce}: *b1* **using** *wfCE-fstI wfD-emptyI wfPhi-emptyI infer-v-v-wf b-of.simps* **using** *wfCE-valI* **by** *fastforce*
then show *?case* **using** *wfT-e-eq infer-e-fstI* **by** *auto*
next
case (*infer-e-sndI* Θ $\mathcal{B}$ Γ Δ Φ *v z' b1 b2 c z*)
hence Θ ; $\mathcal{B}$; Γ ⊢$_{wf}$ *CE-snd* [*v*]^{ce}: *b2* **using** *wfCE-sndI wfD-emptyI wfPhi-emptyI infer-v-v-wf wfCE-valI*
by (*metis b-of.simps infer-v-wf*)
then show *?case* **using** *wfT-e-eq infer-e-sndI* **by** *auto*
next
case (*infer-e-lenI* Θ $\mathcal{B}$ Γ Δ Φ *v z' c z*)
hence Θ ; $\mathcal{B}$; Γ ⊢$_{wf}$ *CE-len* [*v*]^{ce}: *B-int* **using** *wfCE-lenI wfD-emptyI wfPhi-emptyI infer-v-v-wf wfCE-valI*
by (*metis b-of.simps infer-v-wf*)
then show *?case* **using** *wfT-e-eq infer-e-lenI* **by** *auto*
next
case (*infer-e-mvarI* Θ Γ Φ Δ *u τ*)
then show *?case* **using** *wfD-wfT* **by** *blast*
next
case (*infer-e-concatI* Θ $\mathcal{B}$ Γ Δ Φ *v1 z1 c1 v2 z2 c2 z3*)
hence Θ ; $\mathcal{B}$; Γ ⊢$_{wf}$ *CE-concat* [*v1*]^{ce} [*v2*]^{ce}: *B-bitvec* **using** *wfCE-concatI wfD-emptyI wfPhi-emptyI infer-v-v-wf wfCE-valI*
by (*metis b-of.simps infer-v-wf*)
then show *?case* **using** *wfT-e-eq infer-e-concatI* **by** *auto*
next
case (*infer-e-splitI* Θ $\mathcal{B}$ Γ Δ Φ *v1 z1 c1 v2 z2 z3*)

hence *wfg*: Θ ; $\mathcal{B}$ ⊢$_{wf}$ (*z3*, [*B-bitvec* , *B-bitvec*]^b, *TRUE*) #$_\Gamma$ Γ
using *infer-v-wf wfG-cons2I wfB-pairI wfB-bitvecI* **by** *simp*
have *wfz*: Θ ; $\mathcal{B}$; (*z3*, [*B-bitvec* , *B-bitvec*]^b, *TRUE*) #$_\Gamma$ Γ ⊢$_{wf}$ [[*z3*]^v]^{ce} : [*B-bitvec* , *B-bitvec*]^b
apply(*rule wfCE-valI* , *rule wfV-varI*)
using *wfg* **apply** *simp*
using *lookup.simps*(*2*)[*of z3* [*B-bitvec* , *B-bitvec*]^b *TRUE* Γ *z3*] **by** *simp*
have *1*: Θ ; $\mathcal{B}$; (*z3*, [*B-bitvec* , *B-bitvec*]^b, *TRUE*) #$_\Gamma$ Γ ⊢$_{wf}$ [*v2*]^{ce} : *B-int*
using *check-v-wf*[*OF infer-e-splitI*(*4*)] *wf-weakening*(*1*)[*OF - wfg*] *b-of.simps setG.simps wfCE-valI* **by** *fastforce*
have *2*: Θ ; $\mathcal{B}$; (*z3*, [*B-bitvec* , *B-bitvec*]^b, *TRUE*) #$_\Gamma$ Γ ⊢$_{wf}$ [*v1*]^{ce} : *B-bitvec*
using *infer-v-wf*[*OF infer-e-splitI*(*3*)] *wf-weakening*(*1*)[*OF - wfg*] *b-of.simps setG.simps wfCE-valI* **by** *fastforce*

have Θ ; $\mathcal{B}$; Γ $\vdash_{wf}$ $\{\!|$ *z3* : [*B-bitvec* , *B-bitvec*]^b | [*v1*]^{ce} == [[*#1*[[*z3*]^v]^{ce}]^{ce} @@ [*#2*[[*z3*]^v]^{ce}]^{ce}]^{ce} *AND* [| [*#1*[[*z3*]^v]^{ce}]^{ce} |]^{ce} == [*v2*]^{ce} $|\!\}$
proof
show *atom z3* $\sharp$ (Θ, $\mathcal{B}$, Γ) **using** *infer-e-splitI wfTh-x-fresh wfX-wfY fresh-prod3 wfG-fresh-x* **by** *metis*
show Θ ; $\mathcal{B}$ $\vdash_{wf}$ [*B-bitvec* , *B-bitvec*]^b **using** *wfB-pairI wfB-bitvecI infer-e-splitI wfX-wfY* **by** *metis*
show Θ ; $\mathcal{B}$; (*z3*, [*B-bitvec* , *B-bitvec*]^b, *TRUE*) $\#_\Gamma$
Γ $\vdash_{wf}$ [*v1*]^{ce} == [[*#1*[[*z3*]^v]^{ce}]^{ce} @@ [*#2*[[*z3*]^v]^{ce}]^{ce}]^{ce} *AND* [| [*#1*[[*z3*]^v]^{ce}]^{ce} |]^{ce} == [*v2*]^{ce}
using *wfg wfz 1 2 wf-intros* **by** *meson*
qed
thus *?case* **using** *infer-e-splitI* **by** *auto*
qed

lemma *infer-e-wf*:
fixes *e*::*e* **and** Γ::Γ **and** τ::τ **and** Δ::Δ **and** Φ::Φ
assumes Θ ; Φ ; $\mathcal{B}$; Γ ; Δ $\vdash$ *e* $\Rightarrow$ τ
shows Θ ; $\mathcal{B}$; Γ $\vdash_{wf}$ τ **and** Θ ; $\mathcal{B}$ $\vdash_{wf}$ Γ **and** Θ ; $\mathcal{B}$; Γ $\vdash_{wf}$ Δ **and** Θ $\vdash_{wf}$ Φ **and** Θ ; Φ ; $\mathcal{B}$; Γ ; Δ $\vdash_{wf}$ *e* : (*b-of* τ)
using *infer-e-t-wf infer-e-e-wf wfE-wf assms* **by** *metis+*

lemma *infer-e-fresh*:
fixes *x*::*x*
assumes Θ ; Φ ; $\mathcal{B}$; Γ ; Δ $\vdash$ *e* $\Rightarrow$ τ **and** *atom x* $\sharp$ Γ
shows *atom x* $\sharp$ (*e*,τ)
proof −
have *atom x* $\sharp$ *e* **using** *infer-e-e-wf*[*THEN wfE-x-fresh*,*OF assms*(*1*)] *assms*(*2*) **by** *auto*
moreover have *atom x* $\sharp$ τ **using** *assms infer-e-wf wfT-x-fresh* **by** *metis*
ultimately show *?thesis* **using** *fresh-Pair* **by** *auto*
qed

inductive *check-e* :: Θ $\Rightarrow$ Φ $\Rightarrow$ $\mathcal{B}$ $\Rightarrow$ Γ $\Rightarrow$ Δ $\Rightarrow$ *e* $\Rightarrow$ τ $\Rightarrow$ *bool* (- ; - ; - ; - ; - $\vdash$ - $\Leftarrow$ - [*50*, *50*, *50*] *50*) **where**
check-e-subtypeI: $[\![$ *infer-e T P B G D e* τ' ; *subtype T B G* τ' τ $]\!]$ $\Longrightarrow$ *check-e T P B G D e* τ
equivariance *check-e*
nominal-inductive *check-e* **.**

inductive-cases *check-e-elims*[*elim*!]:
check-e F D B G Θ (*AE-val v*) τ
check-e F D B G Θ *e* τ

lemma *infer-e-fst-pair*:
fixes *v1*::*v*
assumes Θ ; Φ ; {||} ; *GNil* ; Δ $\vdash$ [*#1*[*v1* , *v2*]^v]^e $\Rightarrow$ τ
shows $\exists\tau'$. Θ ; Φ ; {||} ; *GNil* ; Δ $\vdash$ [*v1*]^e $\Rightarrow$ τ' $\wedge$
Θ ; {||} ; *GNil* $\vdash$ τ' $\lesssim$ τ
proof −
obtain *z'* **and** *b1* **and** *b2* **and** *c* **and** *z* **where** ** : τ = ($\{\!|$ *z* : *b1* | *CE-val* (*V-var z*) == *CE-fst*

$[(V\text{-}pair\ v1\ v2)]^{ce}$ $\}\!\rangle$) $\wedge$ *wfD* Θ $\{||\}$ *GNil* Δ $\wedge$ *wfPhi* Θ Φ $\wedge$
Θ ; $\{||\}$; *GNil* $\vdash$ *V-pair v1 v2* $\Rightarrow$ $\{\!\langle$ z' : *B-pair b1 b2* | *c* $\}\!\rangle$ $\wedge$ *atom z* $\sharp$ *V-pair v1 v2*
using *infer-e-elims assms* **by** *metis*
hence $*$: Θ ; $\{||\}$; *GNil* $\vdash$ *V-pair v1 v2* $\Rightarrow$ $\{\!\langle$ z' : *B-pair b1 b2* | *c* $\}\!\rangle$ **by** *auto*

obtain *z1* **and** *b1a* **and** *c1* **and** *z2* **and** *b2a* **and** *c2* **where**
$*$: Θ ; $\{||\}$; *GNil* $\vdash$ *v1* $\Rightarrow$ $\{\!\langle$ *z1* : *b1a* | *c1* $\}\!\rangle$ $\wedge$ Θ ; $\{||\}$; *GNil* $\vdash$ *v2* $\Rightarrow$ $\{\!\langle$ *z2* : *b2a* | *c2* $\}\!\rangle$ $\wedge$ *B-pair b1 b2* = *B-pair b1a b2a*
using *infer-v-elims*(5)[*OF* $*$] **by** *metis*

hence *suppv*: *supp v1* = $\{\}$ $\wedge$ *supp v2* = $\{\}$ $\wedge$ *supp* (*V-pair v1 v2*) = $\{\}$ **using** $**$ *infer-v-v-wf wfV-supp atom-dom.simps setG.simps supp-GNil*
by (*meson wfV-supp-nil*)

hence Θ ; $\{||\}$; *GNil* $\vdash$ *v1* $\Rightarrow$ $\{\!\langle$ *z1* : *b1* | *CE-val* (*V-var z1*) == *CE-val v1* $\}\!\rangle$ **using** *infer-v-form2* $*$
using *fresh-def* **by** *fastforce*
moreover have Θ ; $\{||\}$; *GNil* $\vdash_{wf}$ *CE-fst* $[V\text{-}pair\ v1\ v2]^{ce}$: *b1* **using** *wfCE-fstI infer-v-wf*(1) $**$ *b-of.simps wfCE-valI* **by** *metis*

moreover hence *st*: Θ ; $\{||\}$; *GNil* $\vdash$ $\{\!\langle$ *z1* : *b1* | *CE-val* (*V-var z1*) == *CE-val v1* $\}\!\rangle$ $\lesssim$ ($\{\!\langle$ *z* : *b1* | *CE-val* (*V-var z*) == *CE-fst* $[V\text{-}pair\ v1\ v2]^{ce}$ $\}\!\rangle$)
using *subtype-gnil-fst infer-v-v-wf* **by** *auto*
moreover have *wfD* Θ $\{||\}$ *GNil* Δ $\wedge$ *wfPhi* Θ Φ **using** $**$ **by** *auto*
ultimately show *?thesis* **using** *wfX-wfY* $**$ *infer-e-valI* **by** *metis*
qed

lemma *infer-e-snd-pair*:
assumes Θ ; Φ ; $\{||\}$; *GNil* ; Δ $\vdash$ *AE-snd* (*V-pair v1 v2*) $\Rightarrow$ τ
shows $\exists \tau'.$ Θ ; Φ ; $\{||\}$; *GNil* ; Δ $\vdash$ *AE-val v2* $\Rightarrow$ τ' $\wedge$ Θ ; $\{||\}$; *GNil* $\vdash$ $\tau' \lesssim \tau$
proof −
obtain z' **and** *b1* **and** *b2* **and** *c* **and** *z* **where** $**$: τ = ($\{\!\langle$ *z* : *b2* | *CE-val* (*V-var z*) == *CE-snd* $[(V\text{-}pair\ v1\ v2)]^{ce}$ $\}\!\rangle$) $\wedge$ *wfD* Θ $\{||\}$ *GNil* Δ $\wedge$
Θ ; $\{||\}$; *GNil* $\vdash$ *V-pair v1 v2* $\Rightarrow$ $\{\!\langle$ z' : *B-pair b1 b2* | *c* $\}\!\rangle$ $\wedge$ *atom z* $\sharp$ *V-pair v1 v2*
using *infer-e-elims*(9)[*OF assms*(1)] **by** *metis*
hence $*$: Θ ; $\{||\}$; *GNil* $\vdash$ *V-pair v1 v2* $\Rightarrow$ $\{\!\langle$ z' : *B-pair b1 b2* | *c* $\}\!\rangle$ **by** *auto*

obtain *z1* **and** *b1a* **and** *c1* **and** *z2* **and** *b2a* **and** *c2* **where**
$*$: Θ ; $\{||\}$; *GNil* $\vdash$ *v1* $\Rightarrow$ $\{\!\langle$ *z1* : *b1a* | *c1* $\}\!\rangle$ $\wedge$ Θ ; $\{||\}$; *GNil* $\vdash$ *v2* $\Rightarrow$ $\{\!\langle$ *z2* : *b2a* | *c2* $\}\!\rangle$ $\wedge$ *B-pair b1 b2* = *B-pair b1a b2a*
using *infer-v-elims*(5)[*OF* $*$] **by** *metis*

hence *suppv*: *supp v1* = $\{\}$ $\wedge$ *supp v2* = $\{\}$ $\wedge$ *supp* (*V-pair v1 v2*) = $\{\}$ **using** *infer-v-v-wf wfV.simps v.supp* **by** (*meson* $**$ *wfV-supp-nil*)

hence Θ ; $\{||\}$; *GNil* $\vdash$ *v2* $\Rightarrow$ $\{\!\langle$ *z2* : *b2* | *CE-val* (*V-var z2*) == *CE-val v2* $\}\!\rangle$ **using** *infer-v-form2* $*$
by (*metis b.eq-iff*(4) *empty-iff fresh-def*)
moreover have Θ ; $\{||\}$; *GNil* $\vdash_{wf}$ *CE-snd* $[(V\text{-}pair\ v1\ v2)]^{ce}$: *b2* **using** *wfCE-sndI infer-v-wf*(1) $**$ *b-of.simps wfCE-valI* **by** *metis*
moreover hence *st*: Θ ; $\{||\}$; *GNil* $\vdash$ $\{\!\langle$ *z2* : *b2* | *CE-val* (*V-var z2*) == *CE-val v2* $\}\!\rangle$ $\lesssim$ ($\{\!\langle$ *z* : *b2*

| *CE-val* (*V-var z*) == *CE-snd* [(*V-pair v1 v2*)]^{ce} ⟧)
  using *subtype-gnil-snd infer-v-v-wf* **by** *auto*
 moreover have *wfD* Θ {||} *GNil* Δ ∧ *wfPhi* Θ Φ **using** *assms infer-e-wf* **by** *meson*
 ultimately show *?thesis* **using** ∗∗ *infer-e-valI* **by** *metis*
qed

12.5 Statements

lemma *check-s-v-unit*:
 assumes Θ ; $\mathcal{B}$; Γ ⊢ (⟦ *z* : *B-unit* | *TRUE* ⟧) ≲ τ **and** *wfD* Θ $\mathcal{B}$ Γ Δ **and** *wfPhi* Θ Φ
 shows Θ ; Φ ; $\mathcal{B}$; Γ ; Δ ⊢ *AS-val* (*V-lit L-unit*) ⇐ τ
proof −
 have *wfG* Θ $\mathcal{B}$ Γ **using** *assms subtype-g-wf* **by** *meson*
 moreover hence *wfTh* Θ **using** *wfG-wf* **by** *simp*
 moreover obtain *z′*::*x* **where** *atom z′* ♯ Γ **using** *obtain-fresh* **by** *auto*
 ultimately have ∗:Θ ; $\mathcal{B}$; Γ ⊢ *V-lit L-unit* ⇒ ⟦ *z′* : *B-unit* | *CE-val* (*V-var z′*) == *CE-val* (*V-lit L-unit*) ⟧
  using *infer-v-litI infer-unitI* **by** *simp*
 moreover have *wfT* Θ $\mathcal{B}$ Γ (⟦ *z′* : *B-unit* | *CE-val* (*V-var z′*) == *CE-val* (*V-lit L-unit*) ⟧) **using** *infer-v-t-wf*
  by (*meson calculation*)
 moreover then have Θ ; $\mathcal{B}$; Γ ⊢ (⟦ *z′* : *B-unit* | *CE-val* (*V-var z′*) == *CE-val* (*V-lit L-unit*) ⟧) ≲ τ **using** *subtype-trans subtype-top assms*
  type-for-lit.simps(*4*) *wfX-wfY* **by** *metis*
 ultimately show *?thesis* **using** *check-valI assms* ∗ **by** *auto*
qed

12.6 Replacing Variables

Needed as the typing elimination rules give us facts for an alpha-equivalent version of a term and so need to be able to 'jump back' to a typing judgement for the orginal term

lemma τ-*fresh-c*[*simp*]:
 assumes *atom x* ♯ ⟦ *z* : *b* | *c* ⟧ **and** *atom z* ♯ *x*
 shows *atom x* ♯ *c*
 using τ.*fresh assms fresh-at-base*
 by (*simp add*: *fresh-at-base*(*2*))

lemma *wfT-wfT-if1*:
 assumes *wfT* Θ $\mathcal{B}$ Γ (⟦ *z* : *b-of t* | *CE-val v* == *CE-val* (*V-lit L-false*) *IMP* *c-of t z* ⟧) **and** *atom z* ♯ (Γ,*t*)
 shows *wfT* Θ $\mathcal{B}$ Γ *t*
using *assms* **proof**(*nominal-induct t avoiding*: Γ *z rule*: τ.*strong-induct*)
 case (*T-refined-type z′ b′ c′*)
 show *?case* **proof**(*rule wfT-wfT-if*)
  show ‹ Θ ; $\mathcal{B}$; Γ ⊢$_{wf}$ ⟦ *z* : *b′* | [*v*]^{ce} == [[*L-false*]^{v}]^{ce} *IMP* *c′*[*z′*::=[*z*]^{v}]$_{cv}$ ⟧ ›
   using *T-refined-type b-of.simps c-of.simps subst-defs* **by** *metis*
  show ‹*atom z* ♯ (*c′*, Γ)› **using** *T-refined-type fresh-prodN* τ-*fresh-c* **by** *metis*
 qed
qed

thm *check-s-check-branch-s-check-branch-list.inducts*

lemma *check-s-check-branch-s-wf*:
fixes s::s **and** cs::*branch-s* **and** Θ::Θ **and** Φ::Φ **and** Γ::Γ **and** Δ::Δ **and** v::v **and** τ::τ **and** *css*::*branch-list*
shows Θ ; Φ ; B ; Γ ; $\Delta \vdash s \Leftarrow \tau \quad \Longrightarrow \Theta$; $B \vdash_{wf} \Gamma \wedge$ *wfTh* $\Theta \wedge$ *wfD* Θ B Γ $\Delta \wedge$ *wfT* Θ B Γ $\tau \wedge$ *wfPhi* Θ Φ **and**
check-branch-s Θ Φ B Γ Δ *tid cons const v cs* $\tau \Longrightarrow \Theta$; $B \vdash_{wf} \Gamma \wedge$ *wfTh* $\Theta \wedge$ *wfD* Θ B Γ $\Delta \wedge$ *wfT* Θ B Γ $\tau \wedge$ *wfPhi* Θ Φ
check-branch-list Θ Φ B Γ Δ *tid dclist v css* $\tau \Longrightarrow \Theta$; $B \vdash_{wf} \Gamma \wedge$ *wfTh* $\Theta \wedge$ *wfD* Θ B Γ $\Delta \wedge$ *wfT* Θ B Γ $\tau \wedge$ *wfPhi* Θ Φ
proof(*induct rule*: *check-s-check-branch-s-check-branch-list.inducts*)
case (*check-valI* Θ B Γ Δ Φ v τ' τ)
then show *?case* **using** *infer-v-wf infer-v-wf subtype-wf wfX-wfY wfS-valI*
by (*metis subtype-eq-base2*)
next
case (*check-letI* x Θ Φ $\mathcal{B}$ Γ Δ e τ z s b c)
then have $*$:*wfT* Θ $\mathcal{B}$ $((x,\ b\ ,\ c[z::=V\text{-}var\ x]_v)\ \#_\Gamma\ \Gamma)$ τ **by** *force*
moreover have *atom* $x \sharp \tau$ **using** *check-letI fresh-prodN* **by** *force*
ultimately have Θ ; $\mathcal{B}$;$\Gamma \vdash_{wf} \tau$ **using** *wfT-restrict2* **by** *force*
then show *?case* **using** *check-letI infer-e-wf wfS-letI wfX-wfY wfG-elims* **by** *metis*
next
case (*check-assertI* x Θ Φ $\mathcal{B}$ Γ Δ c τ s)
then have $*$:*wfT* Θ $\mathcal{B}$ $((x,\ B\text{-}bool\ ,\ c)\ \#_\Gamma\ \Gamma)$ τ **by** *force*
moreover have *atom* $x \sharp \tau$ **using** *check-assertI fresh-prodN* **by** *force*
ultimately have Θ ; $\mathcal{B}$;$\Gamma \vdash_{wf} \tau$ **using** *wfT-restrict2* **by** *force*
then show *?case* **using** *check-assertI wfS-assertI wfX-wfY wfG-elims* **by** *metis*
next
case (*check-branch-s-branchI* Θ $\mathcal{B}$ Γ Δ τ *cons const x v* Φ *s tid*)
then show *?case* **using** *wfX-wfY* **by** *metis*
next
case (*check-branch-list-consI* Θ Φ $\mathcal{B}$ Γ Δ *tid dclist' v cs* τ *css*)
then show *?case* **using** *wfX-wfY* **by** *metis*
next
case (*check-branch-list-finalI* Θ Φ $\mathcal{B}$ Γ Δ *tid dclist' v cs* τ)
then show *?case* **using** *wfX-wfY* **by** *metis*
next
case (*check-ifI z* Θ Φ $\mathcal{B}$ Γ Δ *v s1 s2* τ)
hence $*$:*wfT* Θ $\mathcal{B}$ Γ ({| z : *b-of* τ | *CE-val v* == *CE-val* (*V-lit L-false*) *IMP c-of* τ z |}) (**is** *wfT* Θ $\mathcal{B}$ Γ *?tau*) **by** *auto*
hence *wfT* Θ $\mathcal{B}$ Γ τ **using** *wfT-wfT-if1 check-ifI fresh-prodN* **by** *metis*
hence Θ ; $\mathcal{B}$; $\Gamma \vdash_{wf} \tau$ **using** *check-ifI b-of-c-of-eq fresh-prodN* **by** *auto*
thus *?case* **using** *check-ifI* **by** *metis*
next
case (*check-let2I x* Θ Φ $\mathcal{B}$ G Δ *t s1* τ *s2*)
then have *wfT* Θ $\mathcal{B}$ $((x,\ b\text{-}of\ t,\ (c\text{-}of\ t\ x))\ \#_\Gamma\ G)$ τ **by** *fastforce*
moreover have *atom* $x \sharp \tau$ **using** *check-let2I* **by** *force*
ultimately have *wfT* Θ $\mathcal{B}$ G τ **using** *wfT-restrict2* **by** *metis*
then show *?case* **using** *check-let2I* **by** *argo*
next
case (*check-varI u* Δ P G v τ' Φ s τ)
then show *?case* **using** *wfG-elims wfD-elims*
list.distinct list.inject **by** *metis*

next
case (*check-assignI* Θ Φ $\mathcal{B}$ Γ Δ u τ v z τ')
obtain $z'::x$ **where** $*:atom\ z' \sharp \Gamma$ **using** *obtain-fresh* **by** *metis*
moreover have $\{\!|\ z : B\text{-}unit \mid TRUE\ |\!\} = \{\!|\ z' : B\text{-}unit \mid TRUE\ |\!\}$ **by** *auto*
moreover hence *wfT* Θ $\mathcal{B}$ Γ $\{\!|\ z' : B\text{-}unit\ |TRUE\ |\!\}$ **using** *wfT-TRUE check-assignI check-v-wf* $*$ *wfB-unitI wfG-wf* **by** *metis*
ultimately show *?case* **using** *check-v.cases infer-v-wf subtype-wf check-assignI wfT-wf check-v-wf wfG-wf*
by (*meson subtype-wf*)
next
case (*check-whileI* Φ Δ G P *s1* z *s2* τ')
then show *?case* **using** *subtype-wf subtype-wf* **by** *auto*
next
case (*check-seqI* Δ G P *s1* z *s2* τ)
then show *?case* **by** *fast*
next
case (*check-caseI* Θ Φ $\mathcal{B}$ Γ Δ *dclist cs* τ *tid* v z)
then show *?case* **by** *fast*
qed

lemma *fresh-u-replace-true*:
fixes *bv*::*bv* **and** Γ::Γ
assumes *atom bv* $\sharp$ Γ' @ (x, b, c) $\#_\Gamma$ Γ
shows *atom bv* $\sharp$ Γ' @ (x, b, *TRUE*) $\#_\Gamma$ Γ
using *fresh-append-g fresh-GCons assms fresh-Pair c.fresh*(*1*) **by** *auto*

lemma *wf-replace-true1*:
fixes Γ::Γ **and** Φ::Φ **and** Θ::Θ **and** Γ'::Γ **and** v::v **and** e::e **and** c::c **and** c''::c **and** c'::c **and** τ::τ **and** *ts*::(*string*$*\tau$) *list* **and** Δ::Δ **and** b'::b **and** b::b **and** s::s
and *ftq*::*fun-typ-q* **and** *ft*::*fun-typ* **and** *ce*::*ce* **and** *td*::*type-def* **and** *cs*::*branch-s* **and** *css*::*branch-list*

shows Θ ; $\mathcal{B}$; $G \vdash_{wf} v : b' \Longrightarrow G = \Gamma'$ @ (x, b, c) $\#_\Gamma$ $\Gamma \Longrightarrow \Theta$; $\mathcal{B}$; Γ' @ ((x, b, *TRUE*) $\#_\Gamma$ Γ) $\vdash_{wf} v : b'$ **and**
Θ ; $\mathcal{B}$; $G \vdash_{wf} c'' \Longrightarrow G = \Gamma'$ @(x, b, c) $\#_\Gamma$ $\Gamma \Longrightarrow \Theta$; $\mathcal{B}$; Γ' @ ((x, b, *TRUE*) $\#_\Gamma$ Γ) $\vdash_{wf} c''$ **and**
Θ ; $\mathcal{B} \vdash_{wf} G \Longrightarrow G = \Gamma'$ @(x, b, c) $\#_\Gamma$ $\Gamma \Longrightarrow \Theta$; $\mathcal{B} \vdash_{wf} \Gamma'$ @ ((x, b, *TRUE*) $\#_\Gamma$ Γ) **and**
Θ ; $\mathcal{B}$; $G \vdash_{wf} \tau \Longrightarrow G = \Gamma'$ @(x, b, c) $\#_\Gamma$ $\Gamma \Longrightarrow \Theta$; $\mathcal{B}$; Γ' @ ((x, b, *TRUE*) $\#_\Gamma$ Γ) $\vdash_{wf} \tau$ **and**
Θ ; $\mathcal{B}$; $\Gamma \vdash_{wf} ts \Longrightarrow$ *True* **and**
$\vdash_{wf} P \Longrightarrow$ *True* **and**
Θ ; $\mathcal{B} \vdash_{wf} b \Longrightarrow$ *True* **and**
Θ ; $\mathcal{B}$; $G \vdash_{wf} ce : b' \Longrightarrow G = \Gamma'$ @(x, b, c) $\#_\Gamma$ $\Gamma \Longrightarrow \Theta$; $\mathcal{B}$; Γ' @ ((x, b, *TRUE*) $\#_\Gamma$ Γ) $\vdash_{wf} ce : b'$ **and**
$\Theta \vdash_{wf} td \Longrightarrow$ *True*
proof(*nominal-induct*
b' **and** c'' **and** G **and** τ **and** *ts* **and** P **and** b **and** b' **and** *td*
arbitrary: Γ Γ' **and** Γ Γ' **and** Γ Γ' **and** Γ Γ' **and** Γ Γ' **and** Γ Γ' **and** Γ Γ' **and** Γ Γ' **and** Γ Γ' **and** Γ Γ' **and** Γ Γ' **and** Γ Γ' **and** Γ Γ' **and** Γ Γ'
rule:*wfV-wfC-wfG-wfT-wfTs-wfTh-wfB-wfCE-wfTD.strong-induct*)
case (*wfB-intI* Θ $\mathcal{B}$)
then show *?case* **using** *wf-intros* **by** *metis*

next
case (*wfB-boolI* Θ $\mathcal{B}$)
then show *?case* **using** *wf-intros* **by** *metis*
next
case (*wfB-unitI* Θ $\mathcal{B}$)
then show *?case* **using** *wf-intros* **by** *metis*
next
case (*wfB-bitvecI* Θ $\mathcal{B}$)
then show *?case* **using** *wf-intros* **by** *metis*
next
case (*wfB-pairI* Θ $\mathcal{B}$ *b1 b2*)
then show *?case* **using** *wf-intros* **by** *metis*
next
case (*wfB-consI* Θ *s dclist* $\mathcal{B}$)
then show *?case* **using** *wf-intros* **by** *metis*
next
case (*wfB-appI* Θ *b s bv dclist* $\mathcal{B}$)
then show *?case* **using** *wf-intros* **by** *metis*
next
case (*wfV-varI* Θ $\mathcal{B}$ Γ'' b' *c* x')
hence *wfg*: ‹ Θ ; $\mathcal{B}$ $\vdash_{wf}$ Γ' @ (x, *b*, *TRUE*) $\#_\Gamma$ Γ › **by** *auto*
show *?case* **proof**(*cases* $x{=}x'$)
case *True*
hence *Some* (*b*, *TRUE*) = *lookup* (Γ' @ (x, *b*, *TRUE*) $\#_\Gamma$ Γ) x' **using** *lookup.simps lookup-inside-wf wfg* **by** *simp*
thus *?thesis* **using** *Wellformed.wfV-varI*[*OF wfg*]
by (*metis True lookup-inside-wf old.prod.inject option.inject wfV-varI.hyps*(*1*) *wfV-varI.hyps*(*3*) *wfV-varI.prems*)
next
case *False*
hence *Some* (b', *c*) = *lookup* (Γ' @ (x, *b*, *TRUE*) $\#_\Gamma$ Γ) x' **using** *lookup-inside2 wfV-varI* **by** *metis*
then show *?thesis* **using** *Wellformed.wfV-varI*[*OF wfg*]
by (*metis wfG-elim2 wfG-suffix wfV-varI.hyps*(*1*) *wfV-varI.hyps*(*2*) *wfV-varI.hyps*(*3*) *wfV-varI.prems Wellformed.wfV-varI wf-replace-inside*(*1*))
qed
next
case (*wfV-litI* Θ $\mathcal{B}$ Γ *l*)
then show *?case* **using** *wf-intros* **using** *wf-intros* **by** *metis*
next
case (*wfV-pairI* Θ $\mathcal{B}$ Γ *v1 b1 v2 b2*)
then show *?case* **using** *wf-intros* **by** *metis*
next
case (*wfV-consI s dclist* Θ *dc x* b' *c* $\mathcal{B}$ Γ *v*)
then show *?case* **using** *wf-intros* **by** *metis*
next
case (*wfV-conspI s bv dclist* Θ *dc xc bc cc* $\mathcal{B}$ b' Γ'' *v*)
show *?case* **proof**
show ‹*AF-typedef-poly s bv dclist* ∈ *set* Θ› **using** *wfV-conspI* **by** *metis*
show ‹(*dc*, {| *xc* : *bc* | *cc* |}) ∈ *set dclist*› **using** *wfV-conspI* **by** *metis*
show ‹ Θ ;$\mathcal{B}$ $\vdash_{wf}$ b' › **using** *wfV-conspI* **by** *metis*
show ‹ Θ ; $\mathcal{B}$; Γ' @ (x, *b*, *TRUE*) $\#_\Gamma$ Γ $\vdash_{wf}$ *v* : *bc*[*bv*::=*b*']$_{bb}$ › **using** *wfV-conspI* **by** *metis*

have *atom bv* $\sharp$ Γ' @ (x, b, *TRUE*) $\#_\Gamma$ Γ **using** *fresh-u-replace-true wfV-conspI* **by** *metis*
thus ‹*atom bv* $\sharp$ (Θ, $\mathcal{B}$, Γ' @ (x, b, *TRUE*) $\#_\Gamma$ Γ, b', v)› **using** *wfV-conspI fresh-prodN* **by** *metis*
qed
next
case (*wfCE-valI* Θ $\mathcal{B}$ Γ v b)
then show *?case* **using** *wf-intros* **by** *metis*
next
case (*wfCE-plusI* Θ $\mathcal{B}$ Γ *v1 v2*)
then show *?case* **using** *wf-intros* **by** *metis*
next
case (*wfCE-leqI* Θ $\mathcal{B}$ Γ *v1 v2*)
then show *?case* **using** *wf-intros* **by** *metis*
next
case (*wfCE-fstI* Θ $\mathcal{B}$ Γ *v1 b1 b2*)
then show *?case* **using** *wf-intros* **by** *metis*
next
case (*wfCE-sndI* Θ $\mathcal{B}$ Γ *v1 b1 b2*)
then show *?case* **using** *wf-intros* **by** *metis*
next
case (*wfCE-concatI* Θ $\mathcal{B}$ Γ *v1 v2*)
then show *?case* **using** *wf-intros* **by** *metis*
next
case (*wfCE-lenI* Θ $\mathcal{B}$ Γ *v1*)
then show *?case* **using** *wf-intros* **by** *metis*
next
case (*wfTI z* Θ $\mathcal{B}$ Γ'' b' c')
show *?case* **proof**
show ‹*atom z* $\sharp$ (Θ, $\mathcal{B}$, Γ' @ (x, b, *TRUE*) $\#_\Gamma$ Γ)› **using** *wfTI fresh-append-g fresh-GCons fresh-prodN* **by** *auto*
show ‹ Θ ; $\mathcal{B}$ $\vdash_{wf}$ b' › **using** *wfTI* **by** *metis*
show ‹ Θ ; $\mathcal{B}$; (z, b', *TRUE*) $\#_\Gamma$ Γ' @ (x, b, *TRUE*) $\#_\Gamma$ Γ $\vdash_{wf}$ c' › **using** *wfTI append-g.simps* **by** *metis*
qed
next
case (*wfC-eqI* Θ $\mathcal{B}$ Γ *e1 b e2*)
then show *?case* **using** *wf-intros* **by** *metis*
next
case (*wfC-trueI* Θ $\mathcal{B}$ Γ)
then show *?case* **using** *wf-intros* **by** *metis*
next
case (*wfC-falseI* Θ $\mathcal{B}$ Γ)
then show *?case* **using** *wf-intros* **by** *metis*
next
case (*wfC-conjI* Θ $\mathcal{B}$ Γ *c1 c2*)
then show *?case* **using** *wf-intros* **by** *metis*
next
case (*wfC-disjI* Θ $\mathcal{B}$ Γ *c1 c2*)
then show *?case* **using** *wf-intros* **by** *metis*
next
case (*wfC-notI* Θ $\mathcal{B}$ Γ *c1*)
then show *?case* **using** *wf-intros* **by** *metis*
next

case (*wfC-impI* Θ $\mathcal{B}$ Γ *c1 c2*)
then show *?case* **using** *wf-intros* **by** *metis*
next
case (*wfG-nilI* Θ $\mathcal{B}$)
then show *?case* **using** *GNil-append* **by** *blast*
next
case (*wfG-cons1I c* Θ $\mathcal{B}$ Γ'' *x b*)
then show *?case* **using** *wf-intros wfG-cons-TRUE2 wfG-elims*(*2*) *wfG-replace-inside wfG-suffix*
by (*metis* (*no-types*, *lifting*))
next
case (*wfG-cons2I c* Θ $\mathcal{B}$ Γ'' *x' b*)
then show *?case* **using** *wf-intros*
by (*metis wfG-cons-TRUE2 wfG-elims*(*2*) *wfG-replace-inside wfG-suffix*)
next
case *wfTh-emptyI*
then show *?case* **using** *wf-intros* **by** *metis*
next
case (*wfTh-consI tdef* Θ)
then show *?case* **using** *wf-intros* **by** *metis*
next
case (*wfTD-simpleI* Θ *lst s*)
then show *?case* **using** *wf-intros* **by** *metis*
next
case (*wfTD-poly* Θ *bv lst s*)
then show *?case* **using** *wf-intros* **by** *metis*
next
case (*wfTs-nil* Θ $\mathcal{B}$ Γ)
then show *?case* **using** *wf-intros* **by** *metis*
next
case (*wfTs-cons* Θ $\mathcal{B}$ Γ τ *dc ts*)
then show *?case* **using** *wf-intros* **by** *metis*
qed

lemma *wf-replace-true2*:
fixes Γ::Γ **and** Φ::Φ **and** Θ::Θ **and** Γ'::Γ **and** *v*::*v* **and** *e*::*e* **and** *c*::*c* **and** c''::*c* **and** c'::*c* **and** τ::τ **and** *ts*::(*string*$*\tau$) *list* **and** Δ::Δ **and** b'::*b* **and** *b*::*b* **and** *s*::*s*
and *ftq*::*fun-typ-q* **and** *ft*::*fun-typ* **and** *ce*::*ce* **and** *td*::*type-def* **and** *cs*::*branch-s* **and** *css*::*branch-list*

shows Θ ; Φ ; $\mathcal{B}$; *G* ; *D* $\vdash_{wf}$ *e* : $b' \Longrightarrow$ *G* = Γ' @(*x*, *b*, *c*) $\#_\Gamma$ $\Gamma \Longrightarrow$ Θ ; Φ ; $\mathcal{B}$; Γ' @ ((*x*, *b*, *TRUE*) $\#_\Gamma$ Γ); *D* $\vdash_{wf}$ *e* : b' **and**
Θ ; Φ ; $\mathcal{B}$; *G* ; Δ $\vdash_{wf}$ *s* : $b' \Longrightarrow$ *G* = Γ' @(*x*, *b*, *c*) $\#_\Gamma$ $\Gamma \Longrightarrow$ Θ ; Φ ; $\mathcal{B}$; Γ' @ ((*x*, *b*, *TRUE*) $\#_\Gamma$ Γ) ; Δ $\vdash_{wf}$ *s* : b' **and**
Θ ; Φ ; $\mathcal{B}$; *G* ; Δ ; *tid* ; *dc* ; *t* $\vdash_{wf}$ *cs* : $b' \Longrightarrow$ *G* = Γ' @(*x*, *b*, *c*) $\#_\Gamma$ $\Gamma \Longrightarrow$ Θ ; Φ ; $\mathcal{B}$; Γ' @ ((*x*, *b*, *TRUE*) $\#_\Gamma$ Γ) ; Δ ; *tid* ; *dc* ; *t* $\vdash_{wf}$ *cs* : b' **and**
Θ ; Φ ; $\mathcal{B}$; *G* ; Δ ; *tid* ; *dclist* $\vdash_{wf}$ *css* : $b' \Longrightarrow$ *G* = Γ' @(*x*, *b*, *c*) $\#_\Gamma$ $\Gamma \Longrightarrow$ Θ ; Φ ; $\mathcal{B}$; Γ' @ ((*x*, *b*, *TRUE*) $\#_\Gamma$ Γ) ; Δ ; *tid* ; *dclist* $\vdash_{wf}$ *css* : b' **and**

Θ $\vdash_{wf}$ $\Phi \Longrightarrow$ *True* **and**
Θ ; $\mathcal{B}$; *G* $\vdash_{wf}$ $\Delta \Longrightarrow$ *G* = Γ' @(*x*, *b*, *c*) $\#_\Gamma$ $\Gamma \Longrightarrow$ Θ ; $\mathcal{B}$; Γ' @ ((*x*, *b*, *TRUE*) $\#_\Gamma$ Γ) $\vdash_{wf}$ Δ **and**

Θ ; Φ $\vdash_{wf}$ *ftq* $\Longrightarrow$ *True* **and**
Θ ; Φ ; $\mathcal{B}$ $\vdash_{wf}$ *ft* $\Longrightarrow$ *True*
proof(*nominal-induct*
b' **and** *b'* **and** *b'* **and** *b'* **and** Φ **and** Δ **and** *ftq* **and** *ft*
arbitrary: Γ Γ' **and** Γ Γ' **and** Γ Γ' **and** Γ Γ' **and** Γ Γ' **and** Γ Γ' **and** Γ Γ' **and** Γ Γ' **and** Γ Γ' **and** Γ Γ' **and** Γ Γ' **and** Γ Γ' **and** Γ Γ' **and** Γ Γ'
rule:*wfE-wfS-wfCS-wfCSS-wfPhi-wfD-wfFTQ-wfFT.strong-induct*)

case (*wfE-valI* Θ Φ $\mathcal{B}$ Γ Δ *v b*)
then show *?case* **using** *wf-intros* **using** *wf-intros wf-replace-true1* **by** *metis*
next
case (*wfE-plusI* Θ Φ $\mathcal{B}$ Γ Δ *v1 v2*)
then show *?case* **using** *wf-intros wf-replace-true1* **by** *metis*
next
case (*wfE-leqI* Θ Φ $\mathcal{B}$ Γ Δ *v1 v2*)
then show *?case* **using** *wf-intros wf-replace-true1* **by** *metis*
next
case (*wfE-fstI* Θ Φ $\mathcal{B}$ Γ Δ *v1 b1 b2*)
then show *?case* **using** *wf-intros wf-replace-true1* **by** *metis*
next
case (*wfE-sndI* Θ Φ $\mathcal{B}$ Γ Δ *v1 b1 b2*)
then show *?case* **using** *wf-intros wf-replace-true1* **by** *metis*
next
case (*wfE-concatI* Θ Φ $\mathcal{B}$ Γ Δ *v1 v2*)
then show *?case* **using** *wf-intros wf-replace-true1* **by** *metis*
next
case (*wfE-splitI* Θ Φ $\mathcal{B}$ Γ Δ *v1 v2*)
then show *?case* **using** *wf-intros wf-replace-true1* **by** *metis*
next
case (*wfE-lenI* Θ Φ $\mathcal{B}$ Γ Δ *v1*)
then show *?case* **using** *wf-intros wf-replace-true1* **by** *metis*
next
case (*wfE-appI* Θ Φ $\mathcal{B}$ Γ Δ *f x b c* τ *s v*)
then show *?case* **using** *wf-intros wf-replace-true1* **by** *metis*
next
case (*wfE-appPI* Θ Φ $\mathcal{B}$ Γ'' Δ *b' bv v* τ *f x1 b1 c1 s*)
show *?case* **proof**
show ‹ Θ $\vdash_{wf}$ Φ › **using** *wfE-appPI wf-replace-true1* **by** *metis*
show ‹ Θ ; $\mathcal{B}$; Γ' @ (*x*, *b*, *TRUE*) $\#_\Gamma$ Γ $\vdash_{wf}$ Δ › **using** *wfE-appPI* **by** *metis*
show ‹ Θ ; $\mathcal{B}$ $\vdash_{wf}$ *b'* › **using** *wfE-appPI* **by** *metis*
have *atom bv* $\sharp$ Γ' @ (*x*, *b*, *TRUE*) $\#_\Gamma$ Γ **using** *fresh-u-replace-true wfE-appPI fresh-prodN* **by** *metis*
thus ‹*atom bv* $\sharp$ (Φ, Θ, $\mathcal{B}$, Γ' @ (*x*, *b*, *TRUE*) $\#_\Gamma$ Γ, Δ, *b'*, *v*, (*b-of* τ)[*bv*::=*b'*]$_b$)›
using *wfE-appPI fresh-prodN* **by** *auto*
show ‹*Some* (*AF-fundef f* (*AF-fun-typ-some bv* (*AF-fun-typ x1 b1 c1* τ *s*))) = *lookup-fun* Φ *f*›
using *wfE-appPI* **by** *metis*
show ‹ Θ ; $\mathcal{B}$; Γ' @ (*x*, *b*, *TRUE*) $\#_\Gamma$ Γ $\vdash_{wf}$ *v* : *b1*[*bv*::=*b'*]$_b$ › **using** *wfE-appPI wf-replace-true1* **by** *metis*
qed
next
case (*wfE-mvarI* Θ Φ $\mathcal{B}$ Γ Δ *u* τ)
then show *?case* **using** *wf-intros wf-replace-true1* **by** *metis*

next

case (*wfS-valI* Θ Φ $\mathcal{B}$ Γ v b Δ)
then show *?case* **using** *wf-intros wf-replace-true1* **by** *metis*
next
case (*wfS-letI* Θ Φ $\mathcal{B}$ Γ'' Δ e b' *x1* s *b1*)
show *?case* **proof**
show ‹ Θ ; Φ ; $\mathcal{B}$; Γ' @ (x, b, *TRUE*) $\#_\Gamma$ Γ ; $\Delta \vdash_{wf} e : b'$ › **using** *wfS-letI wf-replace-true1* **by** *metis*
have ‹ Θ ; Φ ; $\mathcal{B}$; (*x1*, b', *TRUE*) $\#_\Gamma$ Γ') @ (x, b, *TRUE*) $\#_\Gamma$ Γ ; $\Delta \vdash_{wf} s : b1$ › **apply**(*rule wfS-letI*(*4*))
using *wfS-letI append-g.simps* **by** *simp*
thus ‹ Θ ; Φ ; $\mathcal{B}$; (*x1*, b', *TRUE*) $\#_\Gamma$ Γ'@ (x, b, *TRUE*) $\#_\Gamma$ Γ ; $\Delta \vdash_{wf} s : b1$ › **using** *append-g.simps* **by** *auto*
show ‹ Θ ; $\mathcal{B}$; Γ' @ (x, b, *TRUE*) $\#_\Gamma$ $\Gamma \vdash_{wf} \Delta$ › **using** *wfS-letI* **by** *metis*
show *atom x1* $\sharp$ (Φ, Θ, $\mathcal{B}$, Γ' @ (x, b, *TRUE*) $\#_\Gamma$ Γ, Δ, e, *b1*) **using** *fresh-append-g fresh-GCons fresh-prodN wfS-letI* **by** *auto*
qed
next
case (*wfS-assertI* Θ Φ $\mathcal{B}$ x' c Γ'' Δ s b')
show *?case* **proof**
show ‹ Θ ; Φ ; $\mathcal{B}$; (x', *B-bool*, c) $\#_\Gamma$ Γ' @ (x, b, *TRUE*) $\#_\Gamma$ Γ ; $\Delta \vdash_{wf} s : b'$ ›
using *wfS-assertI* (*2*)[*of* (x', *B-bool*, c) $\#_\Gamma$ Γ' Γ] *wfS-assertI* **by** *simp*
show ‹ Θ ; $\mathcal{B}$; Γ' @ (x, b, *TRUE*) $\#_\Gamma$ Γ $\vdash_{wf} c$ › **using** *wfS-assertI wf-replace-true1* **by** *metis*
show ‹ Θ ; $\mathcal{B}$; Γ' @ (x, b, *TRUE*) $\#_\Gamma$ $\Gamma \vdash_{wf} \Delta$ › **using** *wfS-assertI* **by** *metis*
show ‹*atom* x' $\sharp$ (Φ, Θ, $\mathcal{B}$, Γ' @ (x, b, *TRUE*) $\#_\Gamma$ Γ, Δ, c, b', s)› **using** *wfS-assertI fresh-prodN* **by** *simp*
qed
next
case (*wfS-let2I* Θ Φ $\mathcal{B}$ Γ'' Δ *s1* τ x' *s2* *ba'*)
show *?case* **proof**
show ‹ Θ ; Φ ; $\mathcal{B}$; Γ' @ (x, b, *TRUE*) $\#_\Gamma$ Γ ; $\Delta \vdash_{wf} s1 : b\text{-}of\ \tau$ › **using** *wfS-let2I wf-replace-true1* **by** *metis*
show ‹ Θ ; $\mathcal{B}$; Γ' @ (x, b, *TRUE*) $\#_\Gamma$ Γ $\vdash_{wf} \tau$ › **using** *wfS-let2I wf-replace-true1* **by** *metis*
have ‹ Θ ; Φ ; $\mathcal{B}$; ((x', *b-of* τ, *TRUE*) $\#_\Gamma$ Γ') @ (x, b, *TRUE*) $\#_\Gamma$ Γ ; $\Delta \vdash_{wf} s2 : ba'$ ›
apply(*rule wfS-let2I*(*5*))
using *wfS-let2I append-g.simps* **by** *auto*
thus ‹ Θ ; Φ ; $\mathcal{B}$; (x', *b-of* τ, *TRUE*) $\#_\Gamma$ Γ' @ (x, b, *TRUE*) $\#_\Gamma$ Γ ; $\Delta \vdash_{wf} s2 : ba'$ › **using** *wfS-let2I append-g.simps* **by** *auto*
show ‹*atom* x' $\sharp$ (Φ, Θ, $\mathcal{B}$, Γ' @ (x, b, *TRUE*) $\#_\Gamma$ Γ, Δ, *s1*, *ba'*, τ)› **using** *fresh-append-g fresh-GCons fresh-prodN wfS-let2I* **by** *auto*
qed
next
case (*wfS-ifI* Θ $\mathcal{B}$ Γ v Φ Δ *s1* b *s2*)
then show *?case* **using** *wf-intros wf-replace-true1* **by** *metis*
next
case (*wfS-varI* Θ $\mathcal{B}$ Γ'' τ v u Φ Δ b' s)
show *?case* **proof**
show ‹ Θ ; $\mathcal{B}$; Γ' @ (x, b, *TRUE*) $\#_\Gamma$ Γ $\vdash_{wf} \tau$ › **using** *wfS-varI wf-replace-true1* **by** *metis*
show ‹ Θ ; $\mathcal{B}$; Γ' @ (x, b, *TRUE*) $\#_\Gamma$ $\Gamma \vdash_{wf} v : b\text{-}of\ \tau$ › **using** *wfS-varI wf-replace-true1* **by** *metis*
show ‹*atom* u $\sharp$ (Φ, Θ, $\mathcal{B}$, Γ' @ (x, b, *TRUE*) $\#_\Gamma$ Γ, Δ, τ, v, b')› **using** *wfS-varI u-fresh-g fresh-prodN* **by** *auto*

show ‹ Θ ; Φ ; $\mathcal{B}$; Γ' @ (x, b, *TRUE*) $\#_\Gamma$ Γ ; (u, τ) $\#_\Delta$ Δ $\vdash_{wf}$ s : b' › **using** *wfS-varI* **by** *metis*
qed

next
case (*wfS-assignI* u τ Δ Θ $\mathcal{B}$ Γ Φ v)
then show *?case* **using** *wf-intros wf-replace-true1* **by** *metis*
next
case (*wfS-whileI* Θ Φ $\mathcal{B}$ Γ Δ *s1 s2 b*)
then show *?case* **using** *wf-intros wf-replace-true1* **by** *metis*
next
case (*wfS-seqI* Θ Φ $\mathcal{B}$ Γ Δ *s1 s2 b*)
then show *?case* **using** *wf-intros* **by** *metis*
next
case (*wfS-matchI* Θ $\mathcal{B}$ Γ'' *v tid dclist* Δ Φ *cs b'*)
show *?case* **proof**
show ‹ Θ ; $\mathcal{B}$; Γ' @ (x, b, *TRUE*) $\#_\Gamma$ Γ $\vdash_{wf}$ v : *B-id tid* › **using** *wfS-matchI wf-replace-true1* **by** *auto*
show ‹*AF-typedef tid dclist* ∈ *set* Θ› **using** *wfS-matchI* **by** *auto*
show ‹ Θ ; $\mathcal{B}$; Γ' @ (x, b, *TRUE*) $\#_\Gamma$ Γ $\vdash_{wf}$ Δ › **using** *wfS-matchI* **by** *auto*
show ‹ Θ $\vdash_{wf}$ Φ › **using** *wfS-matchI* **by** *auto*
show ‹ Θ ; Φ ; $\mathcal{B}$; Γ' @ (x, b, *TRUE*) $\#_\Gamma$ Γ ; Δ ; *tid* ; *dclist* $\vdash_{wf}$ *cs* : b' › **using** *wfS-matchI* **by** *auto*
qed
next
case (*wfS-branchI* Θ Φ $\mathcal{B}$ x' τ Γ'' Δ s b' *tid dc*)
show *?case* **proof**
have ‹ Θ ; Φ ; $\mathcal{B}$; ((x', *b-of* τ, *TRUE*) $\#_\Gamma$ Γ') @ (x, b, *TRUE*) $\#_\Gamma$ Γ ; Δ $\vdash_{wf}$ s : b' › **using** *wfS-branchI append-g.simps* **by** *metis*
thus ‹ Θ ; Φ ; $\mathcal{B}$; (x', *b-of* τ, *TRUE*) $\#_\Gamma$ Γ' @ (x, b, *TRUE*) $\#_\Gamma$ Γ ; Δ $\vdash_{wf}$ s : b' › **using** *wfS-branchI append-g.simps append-g.simps* **by** *metis*
show ‹*atom* x' ♯ (Φ, Θ, $\mathcal{B}$, Γ' @ (x, b, *TRUE*) $\#_\Gamma$ Γ, Δ, Γ' @ (x, b, *TRUE*) $\#_\Gamma$ Γ, τ)› **using** *wfS-branchI* **by** *auto*
show ‹ Θ ; $\mathcal{B}$; Γ' @ (x, b, *TRUE*) $\#_\Gamma$ Γ $\vdash_{wf}$ Δ › **using** *wfS-branchI* **by** *auto*
qed
next
case (*wfS-finalI* Θ Φ $\mathcal{B}$ Γ Δ *tid dc t cs b*)
then show *?case* **using** *wf-intros* **by** *metis*
next
case (*wfS-cons* Θ Φ $\mathcal{B}$ Γ Δ *tid dc t cs b dclist css*)
then show *?case* **using** *wf-intros* **by** *metis*
next
case (*wfD-emptyI* Θ $\mathcal{B}$ Γ)
then show *?case* **using** *wf-intros wf-replace-true1* **by** *metis*
next
case (*wfD-cons* Θ $\mathcal{B}$ Γ Δ τ u)
then show *?case* **using** *wf-intros wf-replace-true1* **by** *metis*
next
case (*wfPhi-emptyI* Θ)
then show *?case* **using** *wf-intros* **by** *metis*
next
case (*wfPhi-consI f* Θ Φ *ft*)
then show *?case* **using** *wf-intros* **by** *metis*
next

case (*wfFTNone* Θ Φ *ft*)
then show *?case* **using** *wf-intros* **by** *metis*
next
case (*wfFTSome* Θ Φ *bv ft*)
then show *?case* **using** *wf-intros* **by** *metis*
next
case (*wfFTI* Θ *B b* Φ *x c s* τ)
then show *?case* **using** *wf-intros* **by** *metis*
qed

lemmas *wf-replace-true* = *wf-replace-true1 wf-replace-true2*

lemma *check-s-check-branch-s-wfS*:
fixes *s*::*s* **and** *cs*::*branch-s* **and** Θ::Θ **and** Φ::Φ **and** Γ::Γ **and** Δ::Δ **and** *v*::*v* **and** τ::τ **and** *css*::*branch-list*
shows Θ ; Φ ; *B* ; Γ ; $\Delta \vdash s \Leftarrow \tau \quad \Longrightarrow \quad \Theta$; Φ ; *B* ; Γ ; $\Delta \vdash_{wf} s$: *b-of* τ **and**
check-branch-s Θ Φ *B* Γ Δ *tid cons const v cs* $\tau \Longrightarrow$ *wfCS* Θ Φ *B* Γ Δ *tid cons const cs* (*b-of* τ)
check-branch-list Θ Φ *B* Γ Δ *tid dclist v css* $\tau \Longrightarrow$ *wfCSS* Θ Φ *B* Γ Δ *tid dclist css* (*b-of* τ)
proof(*induct rule*: *check-s-check-branch-s-check-branch-list.inducts*)
case (*check-valI* Θ $\mathcal{B}$ Γ Δ Φ *v* τ' τ)
then show *?case* **using** *infer-v-wf infer-v-wf subtype-wf wfX-wfY wfS-valI*
by (*metis subtype-eq-base2*)
next
case (*check-letI x* Θ Φ $\mathcal{B}$ Γ Δ *e* τ *z s b c*)
show *?case* **proof**
show ‹ Θ ; Φ ; $\mathcal{B}$; Γ ; $\Delta \vdash_{wf} e$: *b* › **using** *infer-e-wf check-letI b-of.simps* **by** *metis*
show ‹ Θ ; Φ ; $\mathcal{B}$; (*x*, *b*, *TRUE*) $\#_\Gamma$ Γ ; $\Delta \vdash_{wf} s$: *b-of* τ ›
using *check-letI b-of.simps wf-replace-true2*(*2*)[*OF check-letI*(*5*)] *append-g.simps* **by** *metis*
show ‹ Θ ; $\mathcal{B}$; $\Gamma \vdash_{wf} \Delta$ › **using** *infer-e-wf check-letI b-of.simps* **by** *metis*
show ‹*atom x* ♯ (Φ, Θ, $\mathcal{B}$, Γ, Δ, *e*, *b-of* τ)›
apply(*simp add*: *fresh-prodN*, *intro conjI*)
using *check-letI*(*1*) *fresh-prod7* **by** *simp+*
qed
next
case (*check-assertI x* Θ Φ $\mathcal{B}$ Γ Δ *c* τ *s*)
show *?case* **proof**

show ‹ Θ ; Φ ; $\mathcal{B}$; (*x*, *B-bool*, *c*) $\#_\Gamma$ Γ ; $\Delta \vdash_{wf} s$: *b-of* τ › **using** *check-assertI* **by** *auto*
next
show ‹ Θ ; $\mathcal{B}$; $\Gamma \vdash_{wf} c$ › **using** *check-assertI* **by** *auto*
next
show ‹ Θ ; $\mathcal{B}$; $\Gamma \vdash_{wf} \Delta$ › **using** *check-assertI* **by** *auto*
next
show ‹*atom x* ♯ (Φ, Θ, $\mathcal{B}$, Γ, Δ, *c*, *b-of* τ, *s*)› **using** *check-assertI* **by** *auto*
qed
next
case (*check-branch-s-branchI* Θ $\mathcal{B}$ Γ Δ τ *const x* Φ *tid cons v s*)
show *?case* **proof**
show ‹ Θ ; Φ ; $\mathcal{B}$; (*x*, *b-of const*, *TRUE*) $\#_\Gamma$ Γ ; $\Delta \vdash_{wf} s$: *b-of* τ ›
using *wf-replace-true append-g.simps check-branch-s-branchI* **by** *metis*
show ‹*atom x* ♯ (Φ, Θ, $\mathcal{B}$, Γ, Δ, Γ, *const*)›
using *wf-replace-true append-g.simps check-branch-s-branchI fresh-prodN* **by** *metis*

show ‹ Θ ; $\mathcal{B}$; $\Gamma \vdash_{wf} \Delta$ › **using** *wf-replace-true append-g.simps check-branch-s-branchI* **by** *metis*
qed
next
case (*check-branch-list-consI* Θ Φ $\mathcal{B}$ Γ Δ *tid cons const v cs* τ *dclist css*)
then show *?case* **using** *wf-intros* **by** *metis*
next
case (*check-branch-list-finalI* Θ Φ $\mathcal{B}$ Γ Δ *tid cons const v cs* τ)
then show *?case* **using** *wf-intros* **by** *metis*
next
case (*check-ifI z* Θ Φ $\mathcal{B}$ Γ Δ *v s1 s2* τ)
show *?case* **using** *wfS-ifI check-v-wf check-ifI b-of.simps* **by** *metis*
next
case (*check-let2I x* Θ Φ $\mathcal{B}$ *G* Δ *t s1* τ *s2*)
show *?case* **proof**
show ‹ Θ ; Φ ; $\mathcal{B}$; *G* ; $\Delta \vdash_{wf}$ *s1* : *b-of t* › **using** *check-let2I b-of.simps* **by** *metis*
show ‹ Θ ; $\mathcal{B}$; *G* $\vdash_{wf}$ *t* › **using** *check-let2I check-s-check-branch-s-wf* **by** *metis*
show ‹ Θ ; Φ ; $\mathcal{B}$; (*x*, *b-of t*, *TRUE*) $\#_\Gamma$ *G* ; $\Delta \vdash_{wf}$ *s2* : *b-of* τ ›
using *check-let2I*(*5*) *wf-replace-true2*(*2*) *append-g.simps check-let2I* **by** *metis*
show ‹*atom x* ♯ (Φ, Θ, $\mathcal{B}$, *G*, Δ, *s1*, *b-of* τ, *t*)›
apply(*simp add*: *fresh-prodN*, *intro conjI*)
using *check-let2I*(*1*) *fresh-prod7* **by** *simp+*
qed

next
case (*check-varI u* Θ Φ $\mathcal{B}$ Γ Δ τ' *v* τ *s*)
show *?case* **proof**
show ‹ Θ ; $\mathcal{B}$; Γ $\vdash_{wf}$ τ' › **using** *check-v-wf check-varI* **by** *metis*
show ‹ Θ ; $\mathcal{B}$; $\Gamma \vdash_{wf}$ *v* : *b-of* τ' › **using** *check-v-wf check-varI* **by** *metis*
show ‹*atom u* ♯ (Φ, Θ, $\mathcal{B}$, Γ, Δ, τ', *v*, *b-of* τ)› **using** *check-varI fresh-prodN u-fresh-b* **by** *metis*
show ‹ Θ ; Φ ; $\mathcal{B}$; Γ ; (*u*, τ') $\#_\Delta \Delta \vdash_{wf}$ *s* : *b-of* τ › **using** *check-varI* **by** *metis*
qed
next
case (*check-assignI* Θ Φ $\mathcal{B}$ Γ Δ *u* τ *v z* τ')
then show *?case* **using** *wf-intros check-v-wf subtype-eq-base2 b-of.simps* **by** *metis*
next
case (*check-whileI* Θ Φ $\mathcal{B}$ Γ Δ *s1 z s2* τ')
thus *?case* **using** *wf-intros b-of.simps check-v-wf subtype-eq-base2 b-of.simps* **by** *metis*
next
case (*check-seqI* Θ Φ $\mathcal{B}$ Γ Δ *s1 z s2* τ)
thus *?case* **using** *wf-intros b-of.simps* **by** *metis*
next
case (*check-caseI* Θ Φ $\mathcal{B}$ Γ Δ *tid dclist v cs* τ *z*)
show *?case* **proof**
show ‹ Θ ; $\mathcal{B}$; $\Gamma \vdash_{wf}$ *v* : *B-id tid* › **using** *check-caseI check-v-wf b-of.simps* **by** *metis*
show ‹*AF-typedef tid dclist* ∈ *set* Θ› **using** *check-caseI* **by** *metis*
show ‹ Θ ; $\mathcal{B}$; $\Gamma \vdash_{wf} \Delta$ › **using** *check-caseI check-s-check-branch-s-wf* **by** *metis*
show ‹ Θ $\vdash_{wf}$ Φ › **using** *check-caseI check-s-check-branch-s-wf* **by** *metis*
show ‹ Θ ; Φ ; $\mathcal{B}$; Γ ; Δ ; *tid* ; *dclist* $\vdash_{wf}$ *cs* : *b-of* τ › **using** *check-caseI* **by** *metis*
qed
qed

lemma *check-s-wf*:

fixes $s::s$
assumes $\Theta\ ;\ \Phi\ ;\ B\ ;\ \Gamma\ ;\ \Delta \vdash s \Leftarrow \tau$
shows $\Theta\ ;\ B \vdash_{wf} \Gamma \wedge \mathit{wfT}\ \Theta\ B\ \Gamma\ \tau \wedge \mathit{wfPhi}\ \Theta\ \Phi \wedge \mathit{wfTh}\ \Theta \wedge \mathit{wfD}\ \Theta\ B\ \Gamma\ \Delta \wedge \mathit{wfS}\ \ \Theta\ \Phi\ B\ \Gamma\ \Delta\ s\ (\mathit{b\text{-}of}\ \tau)$
using *check-s-check-branch-s-wf check-s-check-branch-s-wfS assms* **by** *meson*

lemma *check-s-flip-u1*:
fixes $s::s$ **and** $u::u$ **and** $u'::u$
assumes $\Theta\ ;\ \Phi\ ;\ \mathcal{B}\ ;\ \Gamma\ ;\ \Delta \vdash s \Leftarrow \tau$
shows $\Theta\ ;\ \Phi\ ;\ \mathcal{B}\ ;\ \Gamma\ ;\ (\ u \leftrightarrow u')\cdot\Delta\ \vdash (u \leftrightarrow u')\cdot s \Leftarrow\ \tau$
proof −
have $(u \leftrightarrow u')\cdot\Theta\ ;\ (u \leftrightarrow u')\cdot\Phi\ ;\ \ (u \leftrightarrow u')\cdot\ \mathcal{B}\ ;\ (u \leftrightarrow u')\cdot\ \Gamma\ ;\ \ (u \leftrightarrow u')\cdot\Delta\ \ \ \vdash (u \leftrightarrow u')\cdot s \Leftarrow (u \leftrightarrow u')\cdot\tau$
using *check-s.eqvt assms* **by** *blast*
thus *?thesis* **using** *check-s-wf[OF assms] flip-u-eq phi-flip-eq* **by** *metis*
qed

lemma *check-s-flip-u2*:
fixes $s::s$ **and** $u::u$ **and** $u'::u$
assumes $\Theta\ ;\ \Phi\ ;\ \ B\ ;\ \Gamma\ ;\ (\ u \leftrightarrow u')\cdot\Delta\ \ \vdash (u \leftrightarrow u')\cdot s \Leftarrow\ \tau$
shows $\Theta\ ;\ \Phi\ ;\ \ B\ ;\ \Gamma\ ;\ \Delta\ \vdash s \Leftarrow \tau$
proof −
have $\Theta\ ;\ \Phi\ ;\ B\ ;\ \Gamma\ ;\ (\ u \leftrightarrow u')\cdot(\ u \leftrightarrow u')\cdot\Delta\ \ \vdash (\ u \leftrightarrow u')\cdot(u \leftrightarrow u')\cdot s \Leftarrow\ \tau$
using *check-s-flip-u1 assms* **by** *blast*
thus *?thesis* **using** *permute-flip-cancel* **by** *force*
qed

lemma *check-s-flip-u*:
fixes $s::s$ **and** $u::u$ **and** $u'::u$
shows $\Theta\ ;\ \Phi\ ;\ B\ ;\ \Gamma\ ;\ (\ u \leftrightarrow u')\cdot\Delta\ \vdash (u \leftrightarrow u')\cdot s \Leftarrow\ \tau = (\Theta\ ;\ \Phi\ ;\ B\ ;\ \Gamma\ ;\ \Delta \vdash s \Leftarrow \tau)$
using *check-s-flip-u1 check-s-flip-u2* **by** *metis*

lemma *check-s-abs-u*:
fixes $s::s$ **and** $s'::s$ **and** $u::u$ **and** $u'::u$ **and** $\tau'::\tau$
assumes $[[atom\ u]]lst.\ s = [[atom\ u']]lst.\ s'$ **and** $atom\ u\ \sharp\ \Delta$ **and** $atom\ u'\ \sharp\ \Delta$
and $\Theta\ ;\ B\ ;\ \Gamma \vdash_{wf} \tau'$
and $\Theta\ ;\ \Phi\ ;\ B\ ;\ \Gamma\ ;\ (\ u\ ,\ \tau')\ \#_\Delta\Delta\ \vdash s \Leftarrow\ \tau$
shows $\Theta\ ;\ \Phi\ ;\ B\ ;\ \Gamma\ ;\ (\ u',\ \tau'\ )\ \#_\Delta\Delta \vdash s' \Leftarrow \tau$
proof −
have $\Theta\ ;\ \Phi\ ;\ B\ ;\ \Gamma\ ;\ (\ u' \leftrightarrow u)\cdot((\ u\ ,\ \tau')\ \#_\Delta\Delta)\ \vdash (\ u' \leftrightarrow u)\cdot s \Leftarrow\ \tau$
using *assms check-s-flip-u* **by** *metis*
moreover have $(\ u' \leftrightarrow u)\cdot((\ u\ ,\ \tau')\ \#_\Delta\ \Delta) = (\ u'\ ,\ \tau')\ \#_\Delta\Delta$ **proof** −
have $(\ u' \leftrightarrow u)\cdot((\ u\ ,\ \tau')\ \#_\Delta\ \Delta) = ((u' \leftrightarrow u)\cdot u,\ (u' \leftrightarrow u)\cdot\tau')\ \#_\Delta\ (u' \leftrightarrow u)\cdot\Delta$
using *DCons-eqvt Pair-eqvt* **by** *auto*
also have $\ldots = (\ u'\ ,\ (u' \leftrightarrow u)\cdot\tau')\ \#_\Delta\ \Delta$
using *assms flip-fresh-fresh* **by** *auto*
also have $\ldots = (\ u'\ ,\ \tau')\ \#_\Delta\ \Delta$ **using**
u-not-in-t fresh-def flip-fresh-fresh assms **by** *metis*
finally show *?thesis* **by** *auto*
qed
moreover have $(\ u' \leftrightarrow u)\cdot s = s'$ **using** *assms Abs1-eq-iff(3)[of u' s' u s]* **by** *auto*
ultimately show *?thesis* **by** *auto*

qed

12.7 Additional Elimination and Intros

12.7.1 Values

nominal-function *b-for* :: *opp* ⇒ *b* **where**
b-for Plus = *B-int*
| *b-for LEq* = *B-bool*
apply(*auto*,*simp add*: *eqvt-def b-for-graph-aux-def*)
by (*meson opp.exhaust*)
nominal-termination (*eqvt*) **by** *lexicographic-order*

lemma *infer-v-pair2I*:
fixes v_1::*v* **and** v_2::*v*
assumes Θ ; $\mathcal{B}$; Γ ⊢ v_1 ⇒ τ_1 **and** Θ ; $\mathcal{B}$; Γ ⊢ v_2 ⇒ τ_2
shows ∃τ. Θ ; $\mathcal{B}$; Γ ⊢ *V-pair* v_1 v_2 ⇒ τ ∧ *b-of* τ = *B-pair* (*b-of* τ_1) (*b-of* τ_2)
proof −
obtain *z1* **and** *b1* **and** *c1* **and** *z2* **and** *b2* **and** *c2* **where** *zbc*: τ_1 = ({| *z1* : *b1* | *c1* |}) ∧ τ_2 = ({| *z2* : *b2* | *c2* |})
using τ.*exhaust* **by** *meson*
obtain *z*::*x* **where** *atom z* ♯ (v_1, v_2,Γ) **using** *obtain-fresh*
by *blast*
hence *atom z* ♯ (v_1, v_2) ∧ *atom z* ♯ Γ **by** *auto*
hence Θ ; $\mathcal{B}$; Γ ⊢ *V-pair* v_1 v_2 ⇒ {| *z* : *B-pair b1 b2* | *CE-val* (*V-var z*) == *CE-val* (*V-pair* v_1 v_2) |}
using *assms infer-v-pairI zbc* **by** *auto*
moreover obtain τ **where** τ = ({| *z* : *B-pair b1 b2* | *CE-val* (*V-var z*) == *CE-val* (*V-pair* v_1 v_2) |}) **by** *blast*
moreover hence *b-of* τ = *B-pair* (*b-of* τ_1) (*b-of* τ_2) **using** *b-of.simps zbc* **by** *presburger*
ultimately show *?thesis* **by** *meson*
qed

lemma *infer-v-pair2I-zbc*:
fixes v_1::*v* **and** v_2::*v*
assumes Θ ; $\mathcal{B}$; Γ ⊢ v_1 ⇒ τ_1 **and** Θ ; $\mathcal{B}$; Γ ⊢ v_2 ⇒ τ_2
shows ∃*z* τ. Θ ; $\mathcal{B}$; Γ ⊢ *V-pair* v_1 v_2 ⇒ τ ∧ τ = ({| *z* : *B-pair* (*b-of* τ_1) (*b-of* τ_2) | *C-eq* (*CE-val* (*V-var z*)) (*CE-val* (*V-pair* v_1 v_2)) |}) ∧ *atom z* ♯ (v_1,v_2) ∧ *atom z* ♯ Γ
proof −
obtain *z1* **and** *b1* **and** *c1* **and** *z2* **and** *b2* **and** *c2* **where** *zbc*: τ_1 = ({| *z1* : *b1* | *c1* |}) ∧ τ_2 = ({| *z2* : *b2* | *c2* |})
using τ.*exhaust* **by** *meson*
obtain *z*::*x* **where** ∗ : *atom z* ♯ (v_1, v_2,Γ) **using** *obtain-fresh*
by *blast*
hence *vinf*: Θ ; $\mathcal{B}$; Γ ⊢ *V-pair* v_1 v_2 ⇒ {| *z* : *B-pair b1 b2* | *CE-val* (*V-var z*) == *CE-val* (*V-pair* v_1 v_2) |}
using *assms infer-v-pairI*[*of z* v_1 v_2 Γ Θ $\mathcal{B}$ *z1 b1 c1 z2 b2 c2*] *zbc* **by** *simp*
moreover obtain τ **where** τ = ({| *z* : *B-pair b1 b2* | *CE-val* (*V-var z*) == *CE-val* (*V-pair* v_1 v_2) |}) **by** *blast*
moreover have *b-of* τ_1 = *b1* ∧ *b-of* τ_2 = *b2* **using** *zbc b-of.simps* **by** *auto*
ultimately have Θ ; $\mathcal{B}$; Γ ⊢ *V-pair* v_1 v_2 ⇒ τ ∧ τ = ({| *z* : *B-pair* (*b-of* τ_1) (*b-of* τ_2) | *CE-val*

$(V\text{-}var\ z)\ ==\ CE\text{-}val\ (V\text{-}pair\ v_1\ v_2)\ \}\!\rangle$ **by** *auto*
thus *?thesis* **using** $*$ *fresh-prod2 fresh-prod3* **by** *metis*
qed

lemma *infer-v-pair2E*:
assumes $\Theta\ ;\ \mathcal{B}\ ;\ \Gamma \vdash V\text{-}pair\ v_1\ v_2 \Rightarrow \tau$
shows $\exists \tau_1\ \tau_2\ z\ .\ \Theta\ ;\ \mathcal{B}\ ;\ \Gamma \vdash v_1 \Rightarrow \tau_1 \wedge \Theta\ ;\ \mathcal{B}\ ;\ \Gamma \vdash v_2 \Rightarrow \tau_2 \wedge$
$\tau = (\{\!|\ z : B\text{-}pair\ (b\text{-}of\ \tau_1)\ (b\text{-}of\ \tau_2)\ |\ C\text{-}eq\ (CE\text{-}val\ (V\text{-}var\ z))\ (CE\text{-}val\ (V\text{-}pair\ v_1\ v_2))\ |\!\}) \wedge$
$atom\ z \sharp (v_1, v_2)$
proof −
obtain *z* **and** *z1* **and** *b1* **and** *c1* **and** *z2* **and** *b2* **and** *c2* **where**
$\tau = (\{\!|\ z : B\text{-}pair\ b1\ b2\ |\ CE\text{-}val\ (V\text{-}var\ z)\ ==\ CE\text{-}val\ (V\text{-}pair\ v_1\ v_2)\ |\!\}) \wedge$
$atom\ z \sharp (v_1, v_2) \wedge\ \Theta\ ;\ \mathcal{B}\ ;\ \Gamma \vdash v_1 \Rightarrow \{\!|\ z1 : b1\ |\ c1\ |\!\} \wedge\ \Theta\ ;\ \mathcal{B}\ ;\ \Gamma \vdash v_2 \Rightarrow \{\!|\ z2 : b2\ |\ c2\ |\!\}$
using *infer-v-elims assms*
by *blast*
moreover then obtain τ_1 **and** τ_2 **where** $\tau_1 = (\{\!|\ z1 : b1\ |\ c1\ |\!\}) \wedge \tau_2 = (\{\!|\ z2 : b2\ |\ c2\ |\!\})$
by *blast*
moreover hence $b1 = b\text{-}of\ \tau_1 \wedge b2 = b\text{-}of\ \tau_2$ **using** *b-of.simps* **by** *auto*
ultimately show *?thesis* **using** *b-of.simps* **by** *metis*
qed

12.7.2 Expressions

lemma *infer-e-app2E*:
fixes Φ::Φ **and** Θ::Θ
assumes $\Theta\ ;\ \Phi\ ;\ \mathcal{B}\ ;\ \Gamma\ ;\ \Delta \vdash AE\text{-}app\ f\ v \Rightarrow \tau$
shows $\exists x\ b\ c\ s'\ \tau'.\ wfD\ \Theta\ \mathcal{B}\ \Gamma\ \Delta \wedge Some\ (AF\text{-}fundef\ f\ (AF\text{-}fun\text{-}typ\text{-}none\ (AF\text{-}fun\text{-}typ\ x\ b\ c\ \tau'\ s'))) = lookup\text{-}fun\ \Phi\ f \wedge\ \Theta \vdash_{wf} \Phi \wedge$
$\Theta\ ;\ \mathcal{B}\ ;\ \Gamma \vdash v \Leftarrow \{\!|\ x : b\ |\ c\ |\!\} \wedge \tau = \tau'[x::=v]_{\tau v} \wedge atom\ x \sharp \Gamma$
using *infer-e-elims(6)[OF assms] b-of.simps subst-defs* **by** *metis*

lemma *infer-e-appP2E*:
fixes Φ::Φ **and** Θ::Θ
assumes $\Theta\ ;\ \Phi\ ;\ \mathcal{B}\ ;\ \Gamma\ ;\ \Delta \vdash AE\text{-}appP\ f\ b\ v \Rightarrow \tau$
shows $\exists bv\ x\ ba\ c\ s'\ \tau'.\ wfD\ \Theta\ \mathcal{B}\ \Gamma\ \Delta \wedge Some\ (AF\text{-}fundef\ f\ (AF\text{-}fun\text{-}typ\text{-}some\ bv\ (AF\text{-}fun\text{-}typ\ x\ ba\ c\ \tau'\ s'))) = lookup\text{-}fun\ \Phi\ f \wedge\ \Theta \vdash_{wf} \Phi \wedge\ \Theta\ ;\ \mathcal{B} \vdash_{wf} b\ \wedge$
$(\Theta\ ;\ \mathcal{B}\ ;\ \Gamma \vdash v \Leftarrow \{\!|\ x : ba[bv::=b]_{bb}\ |\ c[bv::=b]_{cb}\ |\!\}) \wedge (\tau = \tau'[bv::=b]_{\tau b}[x::=v]_{\tau v}) \wedge atom\ x \sharp \Gamma \wedge atom\ bv \sharp v$
proof −
obtain *bv x ba c s' τ'* **where** $*$:*wfD* $\Theta\ \mathcal{B}\ \Gamma\ \Delta \wedge Some\ (AF\text{-}fundef\ f\ (AF\text{-}fun\text{-}typ\text{-}some\ bv\ (AF\text{-}fun\text{-}typ\ x\ ba\ c\ \tau'\ s'))) = lookup\text{-}fun\ \Phi\ f \wedge\ \Theta \vdash_{wf} \Phi \wedge\ \Theta\ ;\ \mathcal{B} \vdash_{wf} b\ \wedge$
$(\Theta\ ;\ \mathcal{B}\ ;\ \Gamma \vdash v \Leftarrow \{\!|\ x : ba[bv::=b]_{bb}\ |\ c[bv::=b]_{cb}\ |\!\}) \wedge (\tau = \tau'[bv::=b]_{\tau b}[x::=v]_{\tau v}) \wedge atom\ x \sharp \Gamma \wedge atom\ bv \sharp (\Theta, \Phi, \mathcal{B}, \Gamma, \Delta, b, v, \tau)$
using *infer-e-elims(21)[OF assms] subst-defs* **by** *metis*
moreover then have *atom bv* $\sharp$ *v* **using** *fresh-prodN* **by** *metis*
ultimately show *?thesis* **by** *metis*
qed

12.8 Weakening

Lemmas showing that typing judgements hold when a context is extended

lemma *subtype-weakening*:
fixes $\Gamma'::\Gamma$
assumes Θ ; $\mathcal{B}$; $\Gamma \vdash \tau 1 \lesssim \tau 2$ **and** *setG* $\Gamma \subseteq$ *setG* Γ' **and** Θ ; $\mathcal{B} \vdash_{wf} \Gamma'$
shows Θ ; $\mathcal{B}$; $\Gamma' \vdash \tau 1 \lesssim \tau 2$
using *assms* **proof**(*nominal-induct* $\tau 2$ *avoiding*: Γ' *rule*: *subtype.strong-induct*)

case (*subtype-baseI* x Θ $\mathcal{B}$ Γ z c z' c' b)
show *?case* **proof**
show $*$:Θ ; $\mathcal{B}$; Γ' $\vdash_{wf} \{\!|\, z : b \mid c \,|\!\}$ **using** *wfT-weakening subtype-baseI* **by** *metis*
show Θ ; $\mathcal{B}$; Γ' $\vdash_{wf} \{\!|\, z' : b \mid c' \,|\!\}$ **using** *wfT-weakening subtype-baseI* **by** *metis*
show *atom* $x \sharp (\Theta, \mathcal{B}, \Gamma', z, c, z', c')$ **using** *subtype-baseI fresh-Pair* **by** *metis*
have *setG* $((x, b, c[z::=V\text{-}var\ x]_v) \#_\Gamma \Gamma) \subseteq$ *setG* $((x, b, c[z::=V\text{-}var\ x]_v) \#_\Gamma \Gamma')$ **using** *subtype-baseI setG.simps* **by** *blast*
moreover have Θ ; $\mathcal{B} \vdash_{wf} (x, b, c[z::=V\text{-}var\ x]_v) \#_\Gamma \Gamma'$ **using** *wfT-wf-cons3*[*OF* $*$, *of* x] *subtype-baseI fresh-Pair subst-defs* **by** *metis*
ultimately show Θ ; $\mathcal{B}$; $(x, b, c[z::=V\text{-}var\ x]_v) \#_\Gamma \Gamma' \models c'[z'::=V\text{-}var\ x]_v$ **using** *valid-weakening subtype-baseI* **by** *metis*
qed
qed

method *many-rules* **uses** *add* = ((*rule*+),((*simp add*: *add*)+)?)

lemma *infer-v-g-weakening*:
fixes $e::e$ **and** $\Gamma'::\Gamma$ **and** $v::v$
assumes Θ; $\mathcal{B}$; $\Gamma \vdash v \Rightarrow \tau$ **and** *setG* $\Gamma \subseteq$ *setG* Γ' **and** Θ ; $\mathcal{B} \vdash_{wf} \Gamma'$
shows Θ; $\mathcal{B}$; $\Gamma' \vdash v \Rightarrow \tau$
using *assms* **proof**(*nominal-induct* v *arbitrary*: τ *rule*: *v.strong-induct*)
case (*V-lit* l)
obtain z' **and** b' **where** *zbc1*: $\tau = (\{\!|\, z' : b' \mid CE\text{-}val\ (V\text{-}var\ z') == CE\text{-}val\ (V\text{-}lit\ l) \,|\!\})$
using *infer-v-form V-lit* **by** *meson*
obtain z **and** b **where** $\vdash l \Rightarrow (\{\!|\, z : b \mid CE\text{-}val\ (V\text{-}var\ z) == CE\text{-}val\ (V\text{-}lit\ l) \,|\!\})$
using *infer-l-form2 assms infer-v-wf* **by** *metis*
hence *xx*: Θ; $\mathcal{B}$; $\Gamma' \vdash V\text{-}lit\ l \Rightarrow (\{\!|\, z : b \mid CE\text{-}val\ (V\text{-}var\ z) == CE\text{-}val\ (V\text{-}lit\ l) \,|\!\})$
using *infer-v-litI assms*(*1*) *infer-v-wf*
proof −
show *?thesis*
by (*metis* ‹$\vdash l \Rightarrow \{\!|\, z : b \mid [\,[\,z\,]^v\,]^{ce} == [\,[\,l\,]^v\,]^{ce} \,|\!\}$› *assms*(*3*) *infer-v-litI*)
qed
have $b' = b$
using *V-lit.prems*(*1*) ‹$\vdash l \Rightarrow \{\!|\, z : b \mid CE\text{-}val\ (V\text{-}var\ z) == CE\text{-}val\ (V\text{-}lit\ l) \,|\!\}$› τ.*eq-iff infer-l-uniqueness zbc1*
by (*meson infer-v-elims*(*2*))
hence $\tau = (\{\!|\, z : b \mid CE\text{-}val\ (V\text{-}var\ z) == CE\text{-}val\ (V\text{-}lit\ l) \,|\!\})$ **using** *zbc1*
using *type-l-eq* **by** *blast*
then show *?case* **using** *xx* **by** *auto*
next
case (*V-var* x)
obtain z **and** b **and** c **where** $*$:*Some* (b,c) = *lookup* Γ x $\wedge$ *atom* $z \sharp x$ $\wedge$ *atom* $z \sharp \Gamma$ $\wedge$ $\tau = (\{\!|\, z : b \mid CE\text{-}val\ (V\text{-}var\ z) == CE\text{-}val\ (V\text{-}var\ x) \,|\!\})$
using *infer-v-elims*(*1*) *V-var fresh-atom-at-base fresh-finite-insert lookup-iff*
by (*metis finite.emptyI*)

moreover obtain $z'::x$ **where** z':*atom* $z' \sharp (x, \Gamma')$ **using** *obtain-fresh* **by** *blast*
moreover hence $t{:}\tau = (\{\!|\ z' : b\ |\ CE\text{-}val\ (V\text{-}var\ z')\ ==\ CE\text{-}val\ (V\text{-}var\ x)\ |\!\})$ **using** $*$ **by** *force*
moreover hence $**$:*Some* (b,c) = *lookup* $\Gamma'\ x$ **using** *lookup-weakening assms*
using *infer-v-wf* $*$ **by** *metis*

hence $\Theta;\ \mathcal{B}\ ;\ \Gamma' \vdash V\text{-}var\ x \Rightarrow (\{\!|\ z' : b\ |\ CE\text{-}val\ (V\text{-}var\ z')\ ==\ CE\text{-}val\ (V\text{-}var\ x)\ |\!\})$
using *infer-v-varI V-var* $**$ z' **by** *simp*
thus *?case* **using** t **by** *auto*
next
case (*V-pair v1 v2*)
obtain *z z1 b1 c1 z2 b2 c2* **where** $*{:}\tau = \{\!|\ z : B\text{-}pair\ b1\ b2\ |\ CE\text{-}val\ (V\text{-}var\ z)\ ==\ CE\text{-}val\ (V\text{-}pair\ v1\ v2)\ |\!\} \wedge$
atom z $\sharp$ $(v1,\ v2) \wedge$ *atom z* $\sharp\ \Gamma \wedge\ \Theta\ ;\ \mathcal{B}\ ;\ \Gamma \vdash v1 \Rightarrow \{\!|\ z1 : b1\ |\ c1\ |\!\} \wedge\ \Theta\ ;\ \mathcal{B}\ ;\ \Gamma \vdash v2 \Rightarrow \{\!|\ z2 : b2\ |\ c2\ |\!\}$
using *infer-v-elims*(*3*)[*OF V-pair*(*3*)] **by** *metis*
moreover obtain $z'::x$ **where** z':*atom* $z' \sharp (v1,\ v2) \wedge$ *atom* $z' \sharp \Gamma'$ **using** *obtain-fresh fresh-prod2* **by** *metis*
moreover hence $\tau = \{\!|\ z' : B\text{-}pair\ b1\ b2\ |\ CE\text{-}val\ (V\text{-}var\ z')\ ==\ CE\text{-}val\ (V\text{-}pair\ v1\ v2)\ |\!\}$ **using** $*$ **by** *force*
ultimately show *?case* **using** *infer-v-pairI V-pair* **by** *metis*
next
case (*V-consp s dc b v*)
from *V-consp*(*2*) *V-consp*(*1*) *V-consp*(*3*) *V-consp*(*4*) **show** *?case*
proof(*nominal-induct V-consp s dc b v* τ *avoiding*: Γ' *rule*: *infer-v.strong-induct*)
case (*infer-v-conspI bv dclist* Θ *tc* $\mathcal{B}$ Γ *tv z*)
show *?case* **proof**
show ‹*AF-typedef-poly s bv dclist* $\in$ *set* Θ› **using** *infer-v-conspI* **by** *auto*
show ‹(*dc*, *tc*) $\in$ *set dclist*› **using** *infer-v-conspI* **by** *auto*
show ‹ $\Theta\ ;\ \mathcal{B}\ ;\ \Gamma' \vdash v \Rightarrow tv$› **using** *infer-v-conspI* **by** *metis*
show ‹$\Theta\ ;\ \mathcal{B}\ ;\ \Gamma'\ \vdash tv \lesssim tc[bv::=b]_{\tau b}$› **using** *infer-v-conspI subtype-weakening* **by** *metis*
show ‹*atom z* $\sharp$ $(\Theta,\ \mathcal{B},\ \Gamma',\ v,\ b)$› **using** *infer-v-conspI* **by** *auto*
show ‹*atom bv* $\sharp$ $(\Theta,\ \mathcal{B},\ \Gamma',\ v,\ b)$› **using** *infer-v-conspI* **by** *auto*
show ‹ $\Theta\ ;\ \mathcal{B}\ \vdash_{wf} b$ › **using** *infer-v-conspI* **by** *auto*
qed
qed
next
case (*V-cons s dc v*)

obtain *dclist x b c z' c' z* **where**
$*{:}\tau = (\{\!|\ z : B\text{-}id\ s\ |\ CE\text{-}val\ (V\text{-}var\ z)\ ==\ CE\text{-}val\ (V\text{-}cons\ s\ dc\ v)\ |\!\}) \wedge$
AF-typedef s dclist $\in$ *set* $\Theta \wedge$ $(dc, \{\!|\ x : b\ |\ c\ |\!\}) \in$ *set dclist* $\wedge\ \Theta\ ;\ \mathcal{B}\ ;\ \Gamma \vdash v \Rightarrow \{\!|\ z' : b\ |\ c'\ |\!\} \wedge$
$\Theta\ ;\ \mathcal{B}\ ;\ \Gamma\ \vdash \{\!|\ z' : b\ |\ c'\ |\!\} \lesssim \{\!|\ x : b\ |\ c\ |\!\} \wedge$ *atom z* $\sharp$ *v* $\wedge$ *atom z* $\sharp\ \Gamma$
using *infer-v-elims*(*4*)[*OF V-cons*(*2*)] **by** *metis*
moreover obtain $z''::x$ **where** *zdash*:*atom* $z'' \sharp v \wedge$ *atom* $z'' \sharp \Gamma'$ **using** *obtain-fresh fresh-prod2* **by** *metis*
moreover hence $t{:}\tau = (\{\!|\ z'' : B\text{-}id\ s\ |\ CE\text{-}val\ (V\text{-}var\ z'')\ ==\ CE\text{-}val\ (V\text{-}cons\ s\ dc\ v)\ |\!\})$ **proof** –
have *atom* $z'' \sharp$ *AE-val* (*V-cons s dc v*) **using** *zdash e.fresh v.fresh Un-commute b.supp*(*3*) *fresh-def*
sup-bot.right-neutral supp-b-empty v.supp(*4*) **by** *metis*
moreover have *atom z* $\sharp$ *AE-val* (*V-cons s dc v*) **using** $*$ *e.fresh v.fresh Un-commute b.supp*(*3*) *fresh-def*

sup-bot.right-neutral supp-b-empty v.supp(4) **by** *metis*
ultimately show *?thesis* **using** *type-e-eq[of z'' CE-val (V-cons s dc v) z B-id s] ∗* **by** *simp*
qed
moreover have Θ ; $\mathcal{B}$; $\Gamma' \vdash v \Rightarrow \{\!|\ z' : b\ |\ c'\ |\!\}$ **using** *∗ V-cons* **by** *meson*
moreover have Θ ; $\mathcal{B}$; $\Gamma' \vdash \{\!|\ z' : b\ |\ c'\ |\!\} \lesssim \{\!|\ x : b\ |\ c\ |\!\}$ **using** *∗ subtype-weakening V-cons* **by** *meson*

ultimately have Θ ; $\mathcal{B}$; $\Gamma' \vdash$ *V-cons s dc v* $\Rightarrow$ ($\{\!|\ z'' : B\text{-}id\ s\ |\ CE\text{-}val\ (V\text{-}var\ z'')\ ==\ CE\text{-}val\ (V\text{-}cons\ s\ dc\ v)\ |\!\}$)
using *infer-v-consI* **by** *metis*
thus *?case* **using** *t* **by** *auto*
qed

lemma *check-v-g-weakening*:
fixes *e::e* **and** $\Gamma'::\Gamma$
assumes Θ; $\mathcal{B}$; $\Gamma \vdash v \Leftarrow \tau$ **and** *setG* $\Gamma \subseteq$ *setG* Γ' **and** Θ ; $\mathcal{B} \vdash_{wf} \Gamma'$
shows Θ; $\mathcal{B}$; $\Gamma' \vdash v \Leftarrow \tau$
using *subtype-weakening infer-v-g-weakening check-v-elims check-v-subtypeI assms* **by** *metis*

lemma *infer-e-g-weakening*:
fixes *e::e* **and** $\Gamma'::\Gamma$
assumes Θ ; Φ ; $\mathcal{B}$; Γ ; $\Delta \vdash e \Rightarrow \tau$ **and** *setG* $\Gamma \subseteq$ *setG* Γ' **and** Θ ; $\mathcal{B} \vdash_{wf} \Gamma'$
shows Θ ; Φ ; $\mathcal{B}$; Γ'; $\Delta \vdash e \Rightarrow \tau$
using *assms* **proof**(*nominal-induct τ avoiding*: Γ' *rule*: *infer-e.strong-induct*)
case (*infer-e-valI* Θ $\mathcal{B}$ Γ Δ' Φ *v* τ)
then show *?case* **using** *infer-v-g-weakening wf-weakening infer-e.intros* **by** *metis*
next
case (*infer-e-plusI* Θ $\mathcal{B}$ Γ Δ Φ *v1 z1 c1 v2 z2 c2 z3*)

obtain *z'::x* **where** *z'*: *atom z' ♯ v1* $\wedge$ *atom z' ♯ v2* $\wedge$ *atom z' ♯* Γ' **using** *obtain-fresh fresh-prod3* **by** *metis*
moreover hence *∗*:$\{\!|\ z3 : B\text{-}int\ |\ CE\text{-}val\ (V\text{-}var\ z3)\ ==\ CE\text{-}op\ Plus\ [v1]^{ce}\ [v2]^{ce}\ |\!\}$ = ($\{\!|\ z' : B\text{-}int\ |\ CE\text{-}val\ (V\text{-}var\ z')\ ==\ CE\text{-}op\ Plus\ [v1]^{ce}\ [v2]^{ce}\ |\!\}$)
using *infer-e-plusI type-e-eq ce.fresh fresh-e-opp* **by** *auto*

have Θ ; Φ ; $\mathcal{B}$; Γ' ; Δ $\vdash$ *AE-op Plus v1 v2* $\Rightarrow$ $\{\!|\ z' : B\text{-}int\ |\ CE\text{-}val\ (V\text{-}var\ z')\ ==\ CE\text{-}op\ Plus\ [v1]^{ce}\ [v2]^{ce}\ |\!\}$ **proof**
show ‹ Θ ; $\mathcal{B}$; $\Gamma' \vdash_{wf} \Delta$ › **using** *wf-weakening infer-e-plusI* **by** *auto*
show ‹ $\Theta \vdash_{wf} \Phi$ › **using** *infer-e-plusI* **by** *auto*
show ‹ Θ ; $\mathcal{B}$; $\Gamma' \vdash v1 \Rightarrow \{\!|\ z1 : B\text{-}int\ |\ c1\ |\!\}$› **using** *infer-v-g-weakening infer-e-plusI* **by** *auto*
show ‹ Θ ; $\mathcal{B}$; $\Gamma' \vdash v2 \Rightarrow \{\!|\ z2 : B\text{-}int\ |\ c2\ |\!\}$› **using** *infer-v-g-weakening infer-e-plusI* **by** *auto*
show ‹*atom z' ♯ AE-op Plus v1 v2*› **using** *z'* **by** *auto*
show ‹*atom z' ♯* Γ'› **using** *z'* **by** *auto*
qed
thus *?case* **using** *∗* **by** *metis*

next
case (*infer-e-leqI* Θ $\mathcal{B}$ Γ Δ Φ *v1 z1 c1 v2 z2 c2 z3*)
obtain *z'::x* **where** *z'*: *atom z' ♯ v1* $\wedge$ *atom z' ♯ v2* $\wedge$ *atom z' ♯* Γ' **using** *obtain-fresh fresh-prod3* **by** *metis*

moreover hence *∗*:$\{\!|\ z3 : B\text{-}bool\ |\ CE\text{-}val\ (V\text{-}var\ z3)\ ==\ CE\text{-}op\ LEq\ [v1]^{ce}\ [v2]^{ce}\ |\!\}$ = ($\{\!|\ z' :$

B-bool | *CE-val* (*V-var* z') == *CE-op LEq* $[v1]^{ce}$ $[v2]^{ce}$ ⦈)
using *infer-e-leqI type-e-eq ce.fresh fresh-e-opp* **by** *auto*

have Θ ; Φ ; $\mathcal{B}$; Γ' ; Δ $\vdash$ *AE-op LEq v1 v2* $\Rightarrow$ ⦇ z' : *B-bool* | *CE-val* (*V-var* z') == *CE-op LEq* $[v1]^{ce}$ $[v2]^{ce}$ ⦈ **proof**
show ‹ Θ ; $\mathcal{B}$; Γ' $\vdash_{wf}$ Δ › **using** *wf-weakening infer-e-leqI* **by** *auto*
show ‹ Θ $\vdash_{wf}$ Φ › **using** *infer-e-leqI* **by** *auto*
show ‹ Θ ; $\mathcal{B}$; Γ' $\vdash$ *v1* $\Rightarrow$ ⦇ *z1* : *B-int* | *c1* ⦈› **using** *infer-v-g-weakening infer-e-leqI* **by** *auto*
show ‹ Θ ; $\mathcal{B}$; Γ' $\vdash$ *v2* $\Rightarrow$ ⦇ *z2* : *B-int* | *c2* ⦈› **using** *infer-v-g-weakening infer-e-leqI* **by** *auto*
show ‹*atom* z' $\sharp$ *AE-op LEq v1 v2*› **using** z' **by** *auto*
show ‹*atom* z' $\sharp$ Γ'› **using** z' **by** *auto*
qed
thus *?case* **using** $*$ **by** *metis*
next
case (*infer-e-appI* Θ $\mathcal{B}$ Γ Δ Φ f x b c τ' s' v τ)

hence $*$:Θ ; Φ ; $\mathcal{B}$; Γ ; Δ $\vdash$ *AE-app f v* $\Rightarrow$ τ **using** *Typing.infer-e-appI* **by** *auto*

obtain x'::x **where** x':*atom* x' $\sharp$ (s', c, τ', Γ') $\wedge$ (*AF-fundef f* (*AF-fun-typ-none* (*AF-fun-typ x b c* τ' s'))) = (*AF-fundef f* (*AF-fun-typ-none* (*AF-fun-typ* x' *b* ((x' $\leftrightarrow$ x) $\cdot$ c) ((x' $\leftrightarrow$ x) $\cdot$ τ') ((x' $\leftrightarrow$ x) $\cdot$ s'))))
using *obtain-fresh-fun-def*[*of* s' c τ' Γ' f x b] **by** *metis*

hence $**$: ⦇ x : b | c ⦈ = ⦇ x' : b | (x' $\leftrightarrow$ x) $\cdot$ c ⦈ **using** *fresh-PairD*(*1*) *fresh-PairD*(*2*) *type-eq-flip* **by** *blast*
have *atom* x' $\sharp$ Γ **using** x' *infer-e-appI fresh-weakening fresh-Pair* **by** *metis*

show *?case* **proof**
show ‹ Θ ; $\mathcal{B}$; Γ' $\vdash_{wf}$ Δ › **using** *wf-weakening infer-e-appI* **by** *auto*
show ‹ Θ $\vdash_{wf}$ Φ › **using** *wf-weakening infer-e-appI* **by** *auto*
have ‹*Some* (*AF-fundef f* (*AF-fun-typ-none* (*AF-fun-typ x b c* τ' s'))) = *lookup-fun* Φ f› **using** *wf-weakening infer-e-appI* **by** *auto*
thus ‹*Some* (*AF-fundef f* (*AF-fun-typ-none* (*AF-fun-typ* x' *b* ((x' $\leftrightarrow$ x) $\cdot$ c) ((x' $\leftrightarrow$ x) $\cdot$ τ') ((x' $\leftrightarrow$ x) $\cdot$ s')))) = *lookup-fun* Φ f› **using** x' **by** *metis*
show Θ ; $\mathcal{B}$; Γ' $\vdash$ v $\Leftarrow$ ⦇ x' : b | (x' $\leftrightarrow$ x) $\cdot$ c ⦈ **using** *check-v-g-weakening* $**$ *infer-e-appI* **by** *metis*
show *atom* x' $\sharp$ Γ' **using** x' *fresh-Pair* **by** *metis*
have *atom* x $\sharp$ (v, τ) $\wedge$ *atom* x' $\sharp$ (v, τ) **using** x' *infer-e-fresh*[*OF* $*$] *e.fresh*(*2*) *fresh-Pair infer-e-appI* ‹*atom* x' $\sharp$ Γ› **by** *metis*
thus ((x' $\leftrightarrow$ x) $\cdot$ τ')[x'::=v]$_v$ = τ **using** *infer-e-appI*(*7*) *infer-e-appI subst-tv-flip subst-defs* **by** *auto*
qed
next
case (*infer-e-appPI* Θ $\mathcal{B}$ Γ Δ Φ b' f bv x b c τ' s' v τ)

hence $*$:Θ ; Φ ; $\mathcal{B}$; Γ ; Δ $\vdash$ *AE-appP f b' v* $\Rightarrow$ τ **using** *Typing.infer-e-appPI* **by** *auto*

obtain x'::x **where** x':*atom* x' $\sharp$ (s', c, τ', (Γ',v,τ)) $\wedge$ (*AF-fundef f* (*AF-fun-typ-some bv* (*AF-fun-typ x b c* τ' s'))) = (*AF-fundef f* (*AF-fun-typ-some bv* (*AF-fun-typ* x' *b* ((x' $\leftrightarrow$ x) $\cdot$ c) ((x' $\leftrightarrow$ x) $\cdot$ τ') ((x' $\leftrightarrow$ x) $\cdot$ s'))))
using *obtain-fresh-fun-def*[*of* s' c τ' (Γ',v,τ) f x b]
by (*metis fun-def.eq-iff fun-typ-q.eq-iff*(*2*))

hence $**$: $\{\!|\ x : b\ |\ c\ |\!\} = \{\!|\ x' : b\ |\ (x' \leftrightarrow x) \cdot c\ |\!\}$
using *fresh-PairD(1) fresh-PairD(2) type-eq-flip* **by** *blast*
have *atom* $x' \sharp \Gamma$ **using** x' *infer-e-appPI fresh-weakening fresh-Pair* **by** *metis*

show *?case* **proof**(*rule Typing.infer-e-appPI*[**where** $x{=}x'$ **and** $bv{=}bv$ **and** $b{=}b$ **and** $c{=}(x' \leftrightarrow x) \cdot c$ **and** $\tau'{=}(x' \leftrightarrow x) \cdot \tau'$**and** $s'{=}((x' \leftrightarrow x) \cdot s')$])
show $\langle\ \Theta\ ;\ \mathcal{B}\ ;\ \Gamma'\ \vdash_{wf}\ \Delta\ \rangle$ **using** *wf-weakening infer-e-appPI* **by** *auto*
show $\langle\ \Theta\ \vdash_{wf}\ \Phi\ \rangle$ **using** *wf-weakening infer-e-appPI* **by** *auto*
show $\Theta\ ;\ \mathcal{B}\ \vdash_{wf}\ b'$ **using** *infer-e-appPI* **by** *auto*
show *Some* (*AF-fundef f* (*AF-fun-typ-some bv* (*AF-fun-typ* x' b $((x' \leftrightarrow x) \cdot c)$ $((x' \leftrightarrow x) \cdot \tau')$ $((x' \leftrightarrow x) \cdot s')$))) = *lookup-fun* Φ f **using** x' *infer-e-appPI* **by** *argo*
show $\Theta\ ;\ \mathcal{B}\ ;\ \Gamma' \vdash v \Leftarrow \{\!|\ x' : b[bv::=b']_b\ |\ ((x' \leftrightarrow x) \cdot c)[bv::=b']_b\ |\!\}$ **using** $**$
τ*.eq-iff check-v-g-weakening infer-e-appPI.hyps infer-e-appPI.prems(1) infer-e-appPI.prems subst-defs subst-tb.simps* **by** *metis*
show *atom* $x' \sharp \Gamma'$ **using** x' *fresh-prodN* **by** *metis*

have *atom* $x \sharp (v, \tau) \wedge$ *atom* $x' \sharp (v, \tau)$ **using** x' *infer-e-fresh*[*OF* $*$] *e.fresh*(*2*) *fresh-Pair infer-e-appPI* ‹*atom* $x' \sharp \Gamma$› *e.fresh* **by** *metis*
moreover then have $((x' \leftrightarrow x) \cdot \tau')[bv::=b']_{\tau b} = (x' \leftrightarrow x) \cdot (\tau'[bv::=b']_{\tau b})$ **using** *subst-b-x-flip*
by (*metis subst-b-*τ*-def*)
ultimately show $((x' \leftrightarrow x) \cdot \tau')[bv::=b']_b[x'::=v]_v = \tau$
using *infer-e-appPI subst-tv-flip subst-defs* **by** *metis*

show *atom* $bv \sharp$ $(\Theta, \Phi, \mathcal{B}, \Gamma', \Delta, b', v, \tau)$ **using** *infer-e-appPI fresh-prodN* **by** *metis*
qed

next
case (*infer-e-fstI* Θ $\mathcal{B}$ Γ Δ Φ v z'' *b1 b2 c z*)

obtain $z'::x$ **where** $*$: *atom* $z' \sharp \Gamma' \wedge$ *atom* $z' \sharp v \wedge$ *atom* $z' \sharp c$ **using** *obtain-fresh-z fresh-Pair* **by** *metis*
hence $**$:$\{\!|\ z : b1\ |\ CE\text{-}val\ (V\text{-}var\ z)\ ==\ CE\text{-}fst\ [v]^{ce}\ |\!\} = \{\!|\ z' : b1\ |\ CE\text{-}val\ (V\text{-}var\ z')\ ==\ CE\text{-}fst\ [v]^{ce}\ |\!\}$
using *type-e-eq infer-e-fstI v.fresh e.fresh ce.fresh obtain-fresh-z fresh-Pair* **by** *metis*

have $\Theta\ ;\ \Phi\ ;\ \mathcal{B}\ ;\ \Gamma'\ ;\ \Delta\ \vdash$ *AE-fst* $v \Rightarrow \{\!|\ z' : b1\ |\ CE\text{-}val\ (V\text{-}var\ z')\ ==\ CE\text{-}fst\ [v]^{ce}\ |\!\}$ **proof**
show $\langle\ \Theta\ ;\ \mathcal{B}\ ;\ \Gamma'\ \vdash_{wf}\ \Delta\ \rangle$ **using** *wf-weakening infer-e-fstI* **by** *auto*
show $\langle\ \Theta\ \vdash_{wf}\ \Phi\ \rangle$ **using** *wf-weakening infer-e-fstI* **by** *auto*
show $\Theta\ ;\ \mathcal{B}\ ;\ \Gamma' \vdash v \Rightarrow \{\!|\ z'' : B\text{-}pair\ b1\ b2\ |\ c\ |\!\}$ **using** *infer-v-g-weakening infer-e-fstI* **by** *metis*
show *atom* $z' \sharp$ *AE-fst* v **using** $*$ $**$ *e.supp* **by** *auto*
show *atom* $z' \sharp \Gamma'$ **using** $*$ **by** *auto*
qed
thus *?case* **using** $*$ $**$ **by** *metis*
next
case (*infer-e-sndI* Θ $\mathcal{B}$ Γ Δ Φ v z'' *b1 b2 c z*)

obtain $z'::x$ **where** $*$: *atom* $z' \sharp \Gamma' \wedge$ *atom* $z' \sharp v \wedge$ *atom* $z' \sharp c$ **using** *obtain-fresh-z fresh-Pair* **by** *metis*
hence $**$:$\{\!|\ z : b2\ |\ CE\text{-}val\ (V\text{-}var\ z)\ ==\ CE\text{-}snd\ [v]^{ce}\ |\!\} = \{\!|\ z' : b2\ |\ CE\text{-}val\ (V\text{-}var\ z')\ ==\ CE\text{-}snd\ [v]^{ce}\ |\!\}$
using *type-e-eq infer-e-sndI e.fresh ce.fresh obtain-fresh-z fresh-Pair* **by** *metis*

have $\Theta\ ;\ \Phi\ ;\ \mathcal{B}\ ;\ \Gamma'\ ;\ \Delta\ \vdash$ *AE-snd v* $\Rightarrow \{\!|\ z' : b2\ |\ $ *CE-val* (*V-var z'*) $==$ *CE-snd* $[v]^{ce}\ |\!\}$ **proof**
show $\langle\ \Theta\ ;\ \mathcal{B}\ ;\Gamma'\ \vdash_{wf}\ \Delta\ \rangle$ **using** *wf-weakening infer-e-sndI* **by** *auto*
show $\langle\ \Theta\ \vdash_{wf}\ \Phi\ \rangle$ **using** *wf-weakening infer-e-sndI* **by** *auto*
show $\Theta\ ;\ \mathcal{B}\ ;\ \Gamma'\ \vdash v \Rightarrow \{\!|\ z'' : B\text{-}pair\ b1\ b2\ |\ c\ |\!\}$ **using** *infer-v-g-weakening infer-e-sndI* **by** *metis*
show *atom z'* $\sharp$ *AE-snd v* **using** $*$ *e.supp* **by** *auto*
show *atom z'* $\sharp\ \Gamma'$ **using** $*$ **by** *auto*
qed
thus *?case* **using** $**$ **by** *metis*
next
case (*infer-e-lenI* $\Theta\ \ \mathcal{B}\ \Gamma\ \Delta\ \Phi\ v\ z''\ c\ z$)

obtain *z'*::*x* **where** $*$: *atom z'* $\sharp\ \Gamma' \wedge$ *atom z'* $\sharp\ v \wedge$ *atom z'* $\sharp\ c$ **using** *obtain-fresh-z fresh-Pair* **by** *metis*
hence $**$:$\{\!|\ z : B\text{-}int\ |\ $ *CE-val* (*V-var z*) $==$ *CE-len* $[v]^{ce}\ |\!\} = \{\!|\ z' : B\text{-}int\ |\ $ *CE-val* (*V-var z'*) $==$ *CE-len* $[v]^{ce}\ |\!\}$
using *type-e-eq infer-e-lenI e.fresh ce.fresh obtain-fresh-z fresh-Pair* **by** *metis*

have $\Theta\ ;\ \Phi\ ;\ \mathcal{B}\ ;\ \Gamma'\ ;\ \Delta\ \vdash$ *AE-len v* $\Rightarrow \{\!|\ z' : B\text{-}int\ |\ $ *CE-val* (*V-var z'*) $==$ *CE-len* $[v]^{ce}\ |\!\}$ **proof**
show $\langle\ \Theta\ ;\ \mathcal{B}\ ;\ \Gamma' \vdash_{wf}\ \Delta\ \rangle$ **using** *wf-weakening infer-e-lenI* **by** *auto*
show $\langle\ \Theta\ \vdash_{wf}\ \Phi\ \rangle$ **using** *wf-weakening infer-e-lenI* **by** *auto*
show $\Theta\ ;\ \mathcal{B}\ ;\ \Gamma' \vdash v \Rightarrow \{\!|\ z'' : B\text{-}bitvec\ |\ c\ |\!\}$ **using** *infer-v-g-weakening infer-e-lenI* **by** *metis*
show *atom z'* $\sharp$ *AE-len v* **using** $*$ *e.supp* **by** *auto*
show *atom z'* $\sharp\ \Gamma'$ **using** $*$ **by** *auto*
qed
thus *?case* **using** $*$ $**$ **by** *metis*
next
case (*infer-e-mvarI* $\Theta\ \Gamma\ \Phi\ \Delta\ u\ \tau$)
then show *?case* **using** *wf-weakening infer-e.intros* **by** *metis*
next
case (*infer-e-concatI* $\Theta\ \ \mathcal{B}\ \Gamma\ \Delta\ \Phi$ *v1 z1 c1 v2 z2 c2 z3*)

obtain *z'*::*x* **where** $*$: *atom z'* $\sharp\ \Gamma' \wedge$ *atom z'* $\sharp$ *v1* $\wedge$ *atom z'* $\sharp$ *v2* **using** *obtain-fresh-z fresh-Pair* **by** *metis*
hence $**$:$\{\!|\ z3 : B\text{-}bitvec\ |\ $ *CE-val* (*V-var z3*) $==$ *CE-concat* $[v1]^{ce}\ [v2]^{ce}\ |\!\} = \{\!|\ z' : B\text{-}bitvec\ |$ *CE-val* (*V-var z'*) $==$ *CE-concat* $[v1]^{ce}\ [v2]^{ce}\ |\!\}$
using *type-e-eq infer-e-concatI e.fresh ce.fresh obtain-fresh-z fresh-Pair* **by** *metis*

have $\Theta\ ;\ \Phi\ ;\ \mathcal{B}\ ;\ \Gamma'\ ;\ \Delta\ \vdash$ *AE-concat v1 v2* $\Rightarrow \{\!|\ z' : B\text{-}bitvec\ |\ $ *CE-val* (*V-var z'*) $==$ *CE-concat* $[v1]^{ce}\ [v2]^{ce}\ |\!\}$ **proof**
show $\langle\ \Theta\ ;\ \mathcal{B}\ ;\ \Gamma' \vdash_{wf}\ \Delta\ \rangle$ **using** *wf-weakening infer-e-concatI* **by** *auto*
show $\langle\ \Theta\ \vdash_{wf}\ \Phi\ \rangle$ **using** *wf-weakening infer-e-concatI* **by** *auto*
show $\Theta\ ;\ \mathcal{B}\ ;\ \Gamma' \vdash$ *v1* $\Rightarrow \{\!|\ z1 : B\text{-}bitvec\ |\ c1\ |\!\}$ **using** *infer-v-g-weakening infer-e-concatI* **by** *metis*
show $\Theta\ ;\ \mathcal{B}\ ;\ \Gamma' \vdash$ *v2* $\Rightarrow \{\!|\ z2 : B\text{-}bitvec\ |\ c2\ |\!\}$ **using** *infer-v-g-weakening infer-e-concatI* **by** *metis*
show *atom z'* $\sharp$ *AE-concat v1 v2* **using** $*$ *e.supp* **by** *auto*
show *atom z'* $\sharp\ \Gamma'$ **using** $*$ **by** *auto*
qed
thus *?case* **using** $*$ $**$ **by** *metis*
next
case (*infer-e-splitI* $\Theta\ \mathcal{B}\ \Gamma\ \Delta\ \Phi$ *v1 z1 c1 v2 z2 z3*)

show *?case* **proof**
show Θ ; $\mathcal{B}$; $\Gamma' \vdash_{wf} \Delta$ **using** *infer-e-splitI wf-weakening* **by** *auto*
show $\Theta \vdash_{wf} \Phi$ **using** *infer-e-splitI wf-weakening* **by** *auto*
show Θ ; $\mathcal{B}$; $\Gamma' \vdash v1 \Rightarrow \{\!|\ z1 : B\text{-}bitvec \mid c1\ |\!\}$ **using** *infer-v-g-weakening infer-e-splitI* **by** *metis*
show Θ ; $\mathcal{B}$; $\Gamma' \vdash v2 \Leftarrow \{\!|\ z2 : B\text{-}int \mid [\ [\ leq\ [\ [\ L\text{-}num\ 0\ ]^v\ ]^{ce}\ [\ [\ z2\ ]^v\ ]^{ce}\ ]^{ce} == [\ [\ L\text{-}true\ ]^v\ ]^{ce}$
$AND\ [\ leq\ [\ [\ z2\ ]^v\ ]^{ce}\ [|\ [\ v1\ ]^{ce}\ |]^{ce}\ ]^{ce} == [\ [\ L\text{-}true\ ]^v\ ]^{ce}\ |\!\}$
using *check-v-g-weakening infer-e-splitI* **by** *metis*
show *atom z1* $\sharp$ *AE-split v1 v2* **using** *infer-e-splitI* **by** *auto*
show *atom z1* $\sharp$ Γ' **using** *infer-e-splitI* **by** *auto*
show *atom z2* $\sharp$ *AE-split v1 v2* **using** *infer-e-splitI* **by** *auto*
show *atom z2* $\sharp$ Γ' **using** *infer-e-splitI* **by** *auto*
show *atom z3* $\sharp$ *AE-split v1 v2* **using** *infer-e-splitI* **by** *auto*
show *atom z3* $\sharp$ Γ' **using** *infer-e-splitI* **by** *auto*
qed
qed

Special cases proved explicitly, other cases at the end with method +

lemma *infer-e-d-weakening*:
fixes *e*::*e*
assumes Θ ; Φ ; $\mathcal{B}$; Γ ; $\Delta \vdash e \Rightarrow \tau$ **and** *setD* $\Delta \subseteq$ *setD* Δ' **and** *wfD* Θ $\mathcal{B}$ Γ Δ'
shows Θ; Φ ; $\mathcal{B}$; Γ ; $\Delta' \vdash e \Rightarrow \tau$
using *assms* **by**(*nominal-induct* τ *avoiding*: Δ' *rule*: *infer-e.strong-induct*,*auto simp add*:*infer-e.intros*)

lemma *wfG-x-fresh-in-v-simple*:
fixes *x*::*x* **and** *v* :: *v*
assumes Θ ; $\mathcal{B}$; $\Gamma \vdash v \Rightarrow \tau$ **and** *atom x* $\sharp$ Γ
shows *atom x* $\sharp$ *v*
using *wfV-x-fresh infer-v-wf assms* **by** *metis*

lemma *check-s-g-weakening*:
fixes *v*::*v* **and** *s*::*s* **and** *cs*::*branch-s* **and** *x*::*x* **and** *c*::*c* **and** *b*::*b* **and** Γ'::Γ **and** Θ::Θ **and** *css*::*branch-list*
shows *check-s* Θ Φ $\mathcal{B}$ Γ Δ *s t* $\Longrightarrow$ *setG* $\Gamma \subseteq$ *setG* $\Gamma' \Longrightarrow$ Θ ; $\mathcal{B} \vdash_{wf} \Gamma' \Longrightarrow$ *check-s* Θ Φ $\mathcal{B}$ Γ' Δ *s t* **and**
check-branch-s Θ Φ $\mathcal{B}$ Γ Δ *tid cons const v cs t* $\Longrightarrow$ *setG* $\Gamma \subseteq$ *setG* $\Gamma' \Longrightarrow$ Θ ; $\mathcal{B} \vdash_{wf} \Gamma' \Longrightarrow$ *check-branch-s* Θ Φ $\mathcal{B}$ Γ' Δ *tid cons const v cs t* **and**
check-branch-list Θ Φ $\mathcal{B}$ Γ Δ *tid dclist v css t* $\Longrightarrow$ *setG* $\Gamma \subseteq$ *setG* $\Gamma' \Longrightarrow$ Θ ; $\mathcal{B} \vdash_{wf} \Gamma' \Longrightarrow$ *check-branch-list* Θ Φ $\mathcal{B}$ Γ' Δ *tid dclist v css t*
proof(*nominal-induct t* **and** *t* **and** *t avoiding*: Γ' *rule*: *check-s-check-branch-s-check-branch-list.strong-induct*)
case (*check-valI* Θ $\mathcal{B}$ Γ Δ' Φ *v* τ' τ)
then show *?case* **using** *Typing.check-valI infer-v-g-weakening wf-weakening subtype-weakening* **by** *metis*
next
case (*check-letI x* Θ Φ $\mathcal{B}$ Γ Δ *e* τ *z s b c*)
hence *xf*:*atom x* $\sharp$ Γ' **by** *metis*
show *?case* **proof**
show *atom x* $\sharp$ (Θ, Φ, $\mathcal{B}$, Γ', Δ, *e*, τ) **using** *check-letI* **using** *fresh-prod4 xf* **by** *metis*
show Θ ; Φ ; $\mathcal{B}$; Γ' ; $\Delta \vdash e \Rightarrow \{\!|\ z : b \mid c\ |\!\}$ **using** *infer-e-g-weakening check-letI* **by** *metis*
show *atom z* $\sharp$ (*x*, Θ, Φ, $\mathcal{B}$, Γ', Δ, *e*, τ, *s*)
by(*unfold fresh-prodN*,*auto simp add*: *check-letI fresh-prodN*)
have *setG* ((x, b, $c[z::=V\text{-}var\ x]_v$) $\#_\Gamma$ Γ) $\subseteq$ *setG* ((x, b, $c[z::=V\text{-}var\ x]_v$) $\#_\Gamma$ Γ') **using** *check-letI setG.simps*
by (*metis Un-commute Un-empty-right Un-insert-right insert-mono*)

moreover hence Θ ; $\mathcal{B}$ $\vdash_{wf}$ $((x,\ b,\ c[z::=V\text{-}var\ x]_v)$ $\#_\Gamma$ $\Gamma')$ **using** *check-letI wfG-cons-weakening check-s-wf* **by** *metis*
ultimately show Θ ; Φ ; $\mathcal{B}$; $(x,\ b,\ c[z::=V\text{-}var\ x]_v)$ $\#_\Gamma$ Γ' ; Δ $\vdash$ $s \Leftarrow \tau$ **using** *check-letI* **by** *metis*
qed
next
case (*check-let2I x* Θ Φ $\mathcal{B}$ *G* Δ *t s1* τ *s2*)
show *?case* **proof**
show *atom x* $\sharp$ $(\Theta,\ \Phi,\ \mathcal{B},\ \Gamma',\ \Delta,\ t,\ s1,\ \tau)$ **using** *check-let2I* **using** *fresh-prod4* **by** *auto*
show Θ ; Φ ; $\mathcal{B}$; Γ' ; Δ $\vdash$ *s1* $\Leftarrow$ *t* **using** *check-let2I* **by** *metis*
have *setG* $((x,\ b\text{-}of\ t,\ c\text{-}of\ t\ x)$ $\#_\Gamma$ $G) \subseteq$ *setG* $((x,\ b\text{-}of\ t,\ c\text{-}of\ t\ x\ )$ $\#_\Gamma$ $\Gamma')$ **using** *check-let2I* **by** *auto*
moreover hence Θ ; $\mathcal{B}$ $\vdash_{wf}$ $((x,\ b\text{-}of\ t,\ c\text{-}of\ t\ x)$ $\#_\Gamma$ $\Gamma')$ **using** *check-let2I wfG-cons-weakening check-s-wf* **by** *metis*
ultimately show Θ ; Φ ; $\mathcal{B}$; $(x,\ b\text{-}of\ t,\ c\text{-}of\ t\ x)$ $\#_\Gamma$ Γ' ; Δ $\vdash$ *s2* $\Leftarrow$ τ **using** *check-let2I* **by** *metis*
qed
next
case (*check-branch-list-consI* Θ Φ $\mathcal{B}$ Γ Δ *tid dclist' v cs* τ *css dclist*)
thus *?case* **using** *Typing.check-branch-list-consI* **by** *metis*
next
case (*check-branch-list-finalI* Θ Φ $\mathcal{B}$ Γ Δ *tid dclist' v cs* τ *dclist*)
thus *?case* **using** *Typing.check-branch-list-finalI* **by** *metis*
next
case (*check-branch-s-branchI* Θ $\mathcal{B}$ Γ Δ τ *const x* Φ *tid cons v s*)
show *?case* **proof**
show Θ ; $\mathcal{B}$; Γ' $\vdash_{wf}$ Δ **using** *wf-weakening2(6) check-branch-s-branchI* **by** *metis*
show $\vdash_{wf}$ Θ **using** *check-branch-s-branchI* **by** *auto*
show Θ ; $\mathcal{B}$; Γ' $\vdash_{wf}$ τ **using** *check-branch-s-branchI wfT-weakening* ‹*wfG* Θ $\mathcal{B}$ Γ'› **by** *presburger*
show Θ ; $\{||\}$; *GNil* $\vdash_{wf}$ *const* **using** *check-branch-s-branchI* **by** *auto*
show *atom x* $\sharp$ $(\Theta,\ \Phi,\ \mathcal{B},\ \Gamma',\ \Delta,\ tid,\ cons,\ const,\ v,\ \tau)$ **using** *check-branch-s-branchI* **by** *auto*
have *setG* $((x,\ b\text{-}of\ const,\ CE\text{-}val\ v\ ==\ CE\text{-}val(V\text{-}cons\ tid\ cons\ (V\text{-}var\ x))\ \ AND\ \ c\text{-}of\ const\ x)$ $\#_\Gamma$ $\Gamma) \subseteq$ *setG* $((x,\ b\text{-}of\ const,\ CE\text{-}val\ v\ ==\ CE\text{-}val\ (V\text{-}cons\ tid\ cons\ (V\text{-}var\ x))\ AND\ \ c\text{-}of\ const\ x)$ $\#_\Gamma$ $\Gamma')$
using *check-branch-s-branchI* **by** *auto*
moreover hence Θ ; $\mathcal{B}$ $\vdash_{wf}$ $((x,\ b\text{-}of\ const,\ CE\text{-}val\ v\ ==\ CE\text{-}val\ (V\text{-}cons\ tid\ cons\ (V\text{-}var\ x))\ AND\ \ c\text{-}of\ const\ x\ )$ $\#_\Gamma$ $\Gamma')$
using *check-branch-s-branchI wfG-cons-weakening check-s-wf* **by** *metis*
ultimately show Θ ; Φ ; $\mathcal{B}$; $(x,\ b\text{-}of\ const,\ CE\text{-}val\ v\ ==\ CE\text{-}val\ (V\text{-}cons\ tid\ cons\ (V\text{-}var\ x))\ AND\ \ c\text{-}of\ const\ x\ )$ $\#_\Gamma$ Γ' ; Δ $\vdash$ $s \Leftarrow \tau$
using *check-branch-s-branchI* **using** *fresh-dom-free* **by** *auto*
qed
next
case (*check-ifI z* Θ Φ $\mathcal{B}$ Γ Δ *v s1 s2* τ)
show *?case* **proof**
show ‹*atom z* $\sharp$ $(\Theta,\ \Phi,\ \mathcal{B},\ \Gamma',\ \Delta,\ v,\ s1,\ s2,\ \tau)$› **using** *fresh-prodN check-ifI* **by** *auto*
show ‹Θ ; $\mathcal{B}$; Γ' $\vdash$ $v \Leftarrow \{\!|\ z : B\text{-}bool\ |\ TRUE\ |\!\}$› **using** *check-v-g-weakening check-ifI* **by** *auto*
show ‹ Θ ; Φ ; $\mathcal{B}$; Γ' ; Δ $\vdash$ *s1* $\Leftarrow$ $\{\!|\ z : b\text{-}of\ \tau\ |\ CE\text{-}val\ v\ ==\ CE\text{-}val\ (V\text{-}lit\ L\text{-}true)\ \ IMP\ \ c\text{-}of\ \tau\ z\ |\!\}$› **using** *check-ifI* **by** *auto*
show ‹ Θ ; Φ ; $\mathcal{B}$; Γ' ; Δ $\vdash$ *s2* $\Leftarrow$ $\{\!|\ z : b\text{-}of\ \tau\ |\ CE\text{-}val\ v\ ==\ CE\text{-}val\ (V\text{-}lit\ L\text{-}false)\ \ IMP\ \ c\text{-}of\ \tau\ z\ |\!\}$› **using** *check-ifI* **by** *auto*
qed
next

case (*check-whileI* Δ G P *s1* z *s2* τ')
then show *?case* **using** *check-s-check-branch-s-check-branch-list.intros check-v-g-weakening subtype-weakening wf-weakening*
by (*meson infer-v-g-weakening*)
next
case (*check-seqI* Δ G P *s1* z *s2* τ)
then show *?case* **using** *check-s-check-branch-s-check-branch-list.intros check-v-g-weakening subtype-weakening wf-weakening*
by (*meson infer-v-g-weakening*)
next
case (*check-varI* u Θ Φ $\mathcal{B}$ Γ Δ τ' v τ s)
thus *?case* **using** *check-v-g-weakening check-s-check-branch-s-check-branch-list.intros* **by** *auto*
next
case (*check-assignI* Θ Φ $\mathcal{B}$ Γ Δ u τ v z τ')
show *?case* **proof**
show ‹Θ $\vdash_{wf}$ Φ › **using** *check-assignI* **by** *auto*
show ‹Θ ; $\mathcal{B}$; $\Gamma' \vdash_{wf}$ Δ› **using** *check-assignI wf-weakening* **by** *auto*
show ‹$(u, \tau) \in$ *setD* Δ› **using** *check-assignI* **by** *auto*
show ‹Θ ; $\mathcal{B}$; Γ' $\vdash$ $v \Leftarrow \tau$› **using** *check-assignI check-v-g-weakening* **by** *auto*
show ‹Θ ; $\mathcal{B}$; Γ' $\vdash$ $\{\!|$ z : *B-unit* | *TRUE* $|\!\}$ $\lesssim \tau'$› **using** *subtype-weakening check-assignI* **by** *auto*
qed
next
case (*check-caseI* Δ Γ Θ *dclist cs* τ *tid* v z)

then show *?case* **using** *check-s-check-branch-s-check-branch-list.intros check-v-g-weakening subtype-weakening wf-weakening*
by (*meson infer-v-g-weakening*)
next
case (*check-assertI* x Θ Φ $\mathcal{B}$ Γ Δ c τ s)
show *?case* **proof**
show ‹*atom* $x \sharp (\Theta, \Phi, \mathcal{B}, \Gamma', \Delta, c, \tau, s)$› **using** *check-assertI* **by** *auto*

have Θ ; $\mathcal{B}$ $\vdash_{wf}$ $(x, B\text{-}bool, c) \#_\Gamma \Gamma$ **using** *check-assertI check-s-wf* **by** *metis*
hence $*$: Θ ; $\mathcal{B}$ $\vdash_{wf}$ $(x, B\text{-}bool, c) \#_\Gamma \Gamma'$ **using** *wfG-cons-weakening check-assertI* **by** *metis*
moreover have *setG* $((x, B\text{-}bool, c) \#_\Gamma \Gamma) \subseteq$ *setG* $((x, B\text{-}bool, c) \#_\Gamma \Gamma')$ **using** *check-assertI* **by** *auto*
thus ‹ Θ ; Φ ; $\mathcal{B}$; $(x, B\text{-}bool, c) \#_\Gamma \Gamma'$; Δ $\vdash$ $s \Leftarrow \tau$› **using** *check-assertI*(*11*) [*OF* - $*$] **by** *auto*

show ‹Θ ; $\mathcal{B}$; Γ' $\models$ c › **using** *check-assertI valid-weakening* **by** *metis*
show ‹ Θ ; $\mathcal{B}$; $\Gamma' \vdash_{wf}$ Δ › **using** *check-assertI wf-weakening* **by** *metis*
qed
qed

lemma *wfG-xa-fresh-in-v*:
fixes c::c **and** Γ::Γ **and** G::Γ **and** v::v **and** xa::x
assumes Θ ; $\mathcal{B}$; $\Gamma \vdash v \Rightarrow \tau$ **and** G=(Γ'@ $(x, b, c[z::=V\text{-}var\ x]_v) \#_\Gamma \Gamma$) **and** *atom xa* $\sharp$ G **and** Θ ; $\mathcal{B} \vdash_{wf} G$
shows *atom xa* $\sharp$ v
proof −
have Θ ; $\mathcal{B}$; $\Gamma \vdash_{wf} v$: *b-of* τ **using** *infer-v-wf assms* **by** *metis*
hence *supp* $v \subseteq$ *atom-dom* Γ $\cup$ *supp* $\mathcal{B}$ **using** *wfV-supp* **by** *simp*

moreover have *atom xa* $\notin$ *atom-dom G*
using *assms wfG-atoms-supp-eq*[*OF assms*(*4*)] *fresh-def* **by** *metis*
ultimately show *?thesis* **using** *fresh-def*
using *assms infer-v-wf wfG-atoms-supp-eq*
fresh-GCons fresh-append-g subsetCE
by (*metis wfG-x-fresh-in-v-simple*)
qed

lemma *fresh-z-subst-g*:
fixes *G*::Γ
assumes *atom z'* $\sharp$ (*x*,*v*) **and** ‹*atom z'* $\sharp$ *G*›
shows *atom z'* $\sharp$ $G[x::=v]_{\Gamma v}$
proof −
have *atom z'* $\sharp$ *v* **using** *assms fresh-prod2* **by** *auto*
thus *?thesis* **using** *fresh-subst-gv assms* **by** *metis*
qed

lemma *wfG-xa-fresh-in-subst-v*:
fixes *c*::*c* **and** *v*::*v* **and** *x*::*x* **and** Γ::Γ **and** *G*::Γ **and** *xa*::*x*
assumes Θ ; $\mathcal{B}$; $\Gamma \vdash v \Rightarrow \tau$ **and** *G*=(Γ'@ (*x*, *b*, *c*[*z*::=*V-var x*]$_v$) $\#_\Gamma$ Γ) **and** *atom xa* $\sharp$ *G* **and** Θ ; $\mathcal{B} \vdash_{wf}$ *G*
shows *atom xa* $\sharp$ (*subst-gv G x v*)
proof −
have *atom xa* $\sharp$ *v* **using** *wfG-xa-fresh-in-v assms* **by** *metis*
thus *?thesis* **using** *fresh-subst-gv assms* **by** *metis*
qed

12.8.1 Weakening Immutable Variable Context

declare *check-s-check-branch-s-check-branch-list.intros*[*simp*]
declare *check-s-check-branch-s-check-branch-list.intros*[*intro*]

lemma *check-s-d-weakening*:
fixes *s*::*s* **and** *v*::*v* **and** *cs*::*branch-s* **and** *css*::*branch-list*
shows Θ ; Φ ; $\mathcal{B}$; Γ ; $\Delta \vdash s \Leftarrow \tau \Longrightarrow$ *setD* $\Delta \subseteq$ *setD* $\Delta' \Longrightarrow$ *wfD* Θ $\mathcal{B}$ Γ $\Delta' \Longrightarrow \Theta$; Φ ; $\mathcal{B}$; Γ ; $\Delta' \vdash s \Leftarrow \tau$ **and**
check-branch-s Θ Φ $\mathcal{B}$ Γ Δ *tid cons const v cs* $\tau \Longrightarrow$ *setD* $\Delta \subseteq$ *setD* $\Delta' \Longrightarrow$ *wfD* Θ $\mathcal{B}$ Γ $\Delta' \Longrightarrow$ *check-branch-s* Θ Φ $\mathcal{B}$ Γ Δ' *tid cons const v cs* τ **and**
check-branch-list Θ Φ $\mathcal{B}$ Γ Δ *tid dclist v css* $\tau \Longrightarrow$ *setD* $\Delta \subseteq$ *setD* $\Delta' \Longrightarrow$ *wfD* Θ $\mathcal{B}$ Γ $\Delta' \Longrightarrow$ *check-branch-list* Θ Φ $\mathcal{B}$ Γ Δ' *tid dclist v css* τ
proof(*nominal-induct* τ **and** τ **and** τ *avoiding*: Δ' *arbitrary*: *v rule*: *check-s-check-branch-s-check-branch-list.strong-indu*
case (*check-valI* Θ $\mathcal{B}$ Γ Δ Φ *v* τ' τ)
then show *?case* **using** *check-s-check-branch-s-check-branch-list.intros* **by** *blast*
next
case (*check-letI x* Θ Φ $\mathcal{B}$ Γ Δ *e* τ *z s b c*)
show *?case* **proof**
show *atom x* $\sharp$ (Θ, Φ, $\mathcal{B}$, Γ, Δ', *e*, τ) **using** *check-letI* **by** *auto*
show *atom z* $\sharp$ (*x*, Θ, Φ, $\mathcal{B}$, Γ, Δ', *e*, τ, *s*) **using** *check-letI* **by** *auto*
show Θ ; Φ ; $\mathcal{B}$; Γ ; $\Delta' \vdash e \Rightarrow \{\!|\ z : b\ |\ c\ |\!\}$ **using** *check-letI infer-e-d-weakening* **by** *auto*
have Θ ; $\mathcal{B} \vdash_{wf}$ (*x*, *b*, *c*[*z*::=*V-var x*]$_v$) $\#_\Gamma$ Γ **using** *check-letI check-s-wf* **by** *metis*
moreover have Θ ; $\mathcal{B}$; $\Gamma \vdash_{wf} \Delta'$ **using** *check-letI check-s-wf* **by** *metis*
ultimately have Θ ; $\mathcal{B}$; (*x*, *b*, *c*[*z*::=*V-var x*]$_v$) $\#_\Gamma$ $\Gamma \vdash_{wf} \Delta'$ **using** *wf-weakening2*(*6*) *setG.simps*
by *fast*

thus Θ ; Φ ; $\mathcal{B}$; (x, b, $c[z::=V\text{-}var\ x]_v$) $\#_\Gamma$ Γ ; Δ' $\vdash$ $s \Leftarrow \tau$ **using** *check-letI* **by** *simp*
qed
next
case (*check-branch-s-branchI* Θ $\mathcal{B}$ Γ Δ τ *const* x Φ *tid cons* v s)
moreover have Θ ;$\mathcal{B} \vdash_{wf}$ (x, *b-of const*, *CE-val* v == *CE-val* (*V-cons tid cons* (*V-var* x)) *AND* *c-of const* x) $\#_\Gamma$ Γ
using *check-s-wf*[*OF check-branch-s-branchI*(*16*)] **by** *metis*
moreover hence Θ ;$\mathcal{B}$; (x, *b-of const*, *CE-val* v == *CE-val* (*V-cons tid cons* (*V-var* x)) *AND* *c-of const* x) $\#_\Gamma$ Γ $\vdash_{wf}$ Δ'
using *wf-weakening2*(*6*) *check-branch-s-branchI* **by** *fastforce*
ultimately show *?case*
using *check-s-check-branch-s-check-branch-list.intros* **by** *simp*
next
case (*check-branch-list-consI* Θ Φ $\mathcal{B}$ Γ Δ *tid dclist v cs* τ *css*)
then show *?case* **using** *check-s-check-branch-s-check-branch-list.intros* **by** *meson*
next
case (*check-branch-list-finalI* Θ Φ $\mathcal{B}$ Γ Δ *tid dclist v cs* τ)
then show *?case* **using** *check-s-check-branch-s-check-branch-list.intros* **by** *meson*
next
case (*check-ifI* z Θ Φ $\mathcal{B}$ Γ Δ v *s1 s2* τ)
show *?case* **proof**
show ‹*atom* z ♯ (Θ, Φ, $\mathcal{B}$, Γ, Δ', v, *s1*, *s2*, τ)› **using** *fresh-prodN check-ifI* **by** *auto*
show ‹Θ ; $\mathcal{B}$; Γ $\vdash$ $v \Leftarrow$ {| z : *B-bool* | *TRUE* |}› **using** *check-ifI* **by** *auto*
show ‹ Θ ; Φ ; $\mathcal{B}$; Γ ; Δ' $\vdash$ *s1* $\Leftarrow$ {| z : *b-of* τ | *CE-val* v == *CE-val* (*V-lit L-true*) *IMP* *c-of* τ z |}› **using** *check-ifI* **by** *auto*
show ‹ Θ ; Φ ; $\mathcal{B}$; Γ ; Δ' $\vdash$ *s2* $\Leftarrow$ {| z : *b-of* τ | *CE-val* v == *CE-val* (*V-lit L-false*) *IMP* *c-of* τ z |}› **using** *check-ifI* **by** *auto*
qed
next
case (*check-assertI* x Θ Φ $\mathcal{B}$ Γ Δ c τ s)
show *?case* **proof**
show *atom* x ♯ (Θ, Φ, $\mathcal{B}$, Γ, Δ', c, τ,s) **using** *fresh-prodN check-assertI* **by** *auto*
show $*$: Θ ; $\mathcal{B}$; $\Gamma \vdash_{wf}$ Δ' **using** *check-assertI* **by** *auto*
hence Θ ; $\mathcal{B}$; (x, *B-bool*, c) $\#_\Gamma$ $\Gamma \vdash_{wf}$ Δ' **using** *wf-weakening2*(*6*)[*OF* $*$, *of* (x, *B-bool*, c) $\#_\Gamma$ Γ] *check-assertI check-s-wf setG.simps* **by** *auto*
thus Θ ; Φ ; $\mathcal{B}$; (x, *B-bool*, c) $\#_\Gamma$ Γ ; Δ' $\vdash$ $s \Leftarrow \tau$
using *check-assertI*(*11*)[*OF* ‹*setD* $\Delta \subseteq$ *setD* Δ'›] **by** *simp*

show Θ ; $\mathcal{B}$; Γ $\models$ c **using** *fresh-prodN check-assertI* **by** *auto*

qed
next
case (*check-let2I* x Θ Φ $\mathcal{B}$ G Δ t *s1* τ *s2*)
show *?case* **proof**
show *atom* x ♯ (Θ, Φ, $\mathcal{B}$, G, Δ', t , *s1*, τ) **using** *check-let2I* **by** *auto*

show Θ ; Φ ; $\mathcal{B}$; G ; Δ' $\vdash$ *s1* $\Leftarrow$ t **using** *check-let2I infer-e-d-weakening* **by** *auto*
have Θ ; $\mathcal{B}$; (x, *b-of t*, *c-of t x*) $\#_\Gamma$ $G \vdash_{wf}$ Δ' **using** *check-let2I wf-weakening2*(*6*) *check-s-wf* **by** *fastforce*
thus Θ ; Φ ; $\mathcal{B}$; (x, *b-of t*, *c-of t x*) $\#_\Gamma$ G ; Δ' $\vdash$ *s2* $\Leftarrow$ τ **using** *check-let2I* **by** *simp*
qed
next

case (*check-varI* $u\ \Theta\ \Phi\ \mathcal{B}\ \Gamma\ \Delta\ \tau'\ v\ \tau\ s$)
show *?case* **proof**
show *atom* $u \sharp (\Theta, \Phi, \mathcal{B}, \Gamma, \Delta', \tau', v, \tau)$ **using** *check-varI* **by** *auto*
show $\Theta\ ;\ \mathcal{B}\ ;\ \Gamma\ \vdash v \Leftarrow \tau'$ **using** *check-varI* **by** *auto*
have *setD* $((u, \tau') \#_\Delta \Delta) \subseteq$ *setD* $((u, \tau') \#_\Delta \Delta')$ **using** *setD.simps check-varI* **by** *auto*
moreover have $u \notin$ *fst* ' *setD* Δ' **using** *check-varI*(*1*) *setD.simps fresh-DCons* **by** (*simp add*: *fresh-d-not-in*)
moreover hence $\Theta\ ;\ \mathcal{B}\ ;\ \Gamma \vdash_{wf} (u, \tau') \#_\Delta \Delta'$ **using** *wfD-cons fresh-DCons setD.simps check-varI check-v-wf* **by** *metis*
ultimately show $\Theta\ ;\ \Phi\ ;\ \mathcal{B}\ ;\ \Gamma\ ;\ (u, \tau') \#_\Delta \Delta' \vdash s \Leftarrow \tau$ **using** *check-varI* **by** *auto*
qed
next
case (*check-assignI* $\Theta\ \Phi\ \mathcal{B}\ \Gamma\ \Delta\ u\ \tau\ v\ z\ \tau'$)
moreover hence $(u, \tau) \in$ *setD* Δ' **by** *auto*
ultimately show *?case* **using** *Typing.check-assignI* **by** *simp*
next
case (*check-whileI* $\Theta\ \Phi\ \mathcal{B}\ \Gamma\ \Delta$ *s1 z s2* τ')
then show *?case* **using** *check-s-check-branch-s-check-branch-list.intros* **by** *meson*
next
case (*check-seqI* $\Theta\ \Phi\ \mathcal{B}\ \Gamma\ \Delta$ *s1 z s2* τ)
then show *?case* **using** *check-s-check-branch-s-check-branch-list.intros* **by** *meson*
next
case (*check-caseI* $\Theta\ \Phi\ \mathcal{B}\ \Gamma\ \Delta$ *tid dclist v cs* τ *z*)
then show *?case* **using** *check-s-check-branch-s-check-branch-list.intros* **by** *meson*

qed

thm *valid-ce-eq*

lemma *valid-ce-eq*:
fixes *v*::*v* **and** *ce2*::*ce*
assumes *ce1* $= ce2[x::=v]_{cev}$ **and** *wfV* Θ $\mathcal{B}$ *GNil v b* **and** *wfCE* Θ $\mathcal{B}$ $((x, b, TRUE) \#_\Gamma GNil)$ *ce2 b'* **and** *wfCE* Θ $\mathcal{B}$ *GNil ce1 b'*
shows $\langle \Theta\ ;\ \mathcal{B}\ ;\ (x, b, ([[x]^v]^{ce} == [v]^{ce})) \#_\Gamma GNil \models ce1 == ce2 \rangle$
unfolding *valid.simps* **proof**
have *wfg*: $\Theta\ ;\ \mathcal{B} \vdash_{wf} (x, b, [[x]^v]^{ce} == [v]^{ce}) \#_\Gamma GNil$
using *wfG-cons1I wfG-nilI wfX-wfY assms wf-intros*
by (*meson fresh-GNil wfC-e-eq wfG-intros2*)

show *wf*: $\langle \Theta\ ;\ \mathcal{B}\ ;\ (x, b, [[x]^v]^{ce} == [v]^{ce}) \#_\Gamma GNil \vdash_{wf} ce1 == ce2 \rangle$
apply(*rule wfC-eqI*[**where** *b*=*b'*])
using *wfg setG.simps assms wfCE-weakening* **apply** *simp*

using *wfg assms wf-replace-inside1*(*8*) *assms*
using *wfC-trueI wf-trans*(*8*) **by** *auto*

show $\langle \forall i.\ ((\Theta\ ;\ (x, b, [[x]^v]^{ce} == [v]^{ce}) \#_\Gamma GNil \vdash i) \wedge (i \models (x, b, [[x]^v]^{ce} == [v]^{ce}) \#_\Gamma GNil) \longrightarrow (i \models (ce1 == ce2))) \rangle$ **proof**(*rule+*,*goal-cases*)
fix *i*
assume *as*:$\Theta\ ;\ (x, b, [[x]^v]^{ce} == [v]^{ce}) \#_\Gamma GNil \vdash i \wedge i \models (x, b, [[x]^v]^{ce} == [v]^{ce}) \#_\Gamma GNil$

have *1*:*wfV* Θ $\mathcal{B}$ ((x, b, [[x]^{v}]^{ce} == [v]^{ce}) #$_{\Gamma}$ *GNil*) v b
using *wf-weakening assms append-g.simps setG.simps wf wfX-wfY*
by (*metis empty-subsetI*)
hence $\exists s.\ i$ ⟦ v ⟧ ~ s **using** *eval-v-exist*[*OF - 1*] *as* **by** *auto*
then obtain s **where** *iv*:i⟦ v ⟧ ~ s **..**

hence *ix*:i x = *Some* s **proof** −
have i ⊨ [[x]^{v}]^{ce} == [v]^{ce} **using** *is-satis-g.simps as* **by** *auto*
hence i ⟦ [[x]^{v}]^{ce} == [v]^{ce} ⟧ ~ *True* **using** *is-satis.simps* **by** *auto*
hence i ⟦ [[x]^{v}]^{ce} ⟧ ~ s **using**
iv eval-e-elims
by (*metis eval-c-elims*(*7*) *eval-e-uniqueness eval-e-valI*)
thus *?thesis* **using** *eval-v-elims*(*2*) *eval-e-elims*(*1*) **by** *metis*
qed

have *1*:*wfCE* Θ $\mathcal{B}$ ((x, b, [[x]^{v}]^{ce} == [v]^{ce}) #$_{\Gamma}$ *GNil*) *ce1* b'
using *wfCE-weakening assms append-g.simps setG.simps wf wfX-wfY*
by (*metis empty-subsetI*)
hence $\exists$ *s1*. i ⟦ *ce1* ⟧ ~ *s1* **using** *eval-e-exist assms as* **by** *auto*
then obtain *s1* **where** *s1*: i⟦*ce1*⟧ ~ *s1* **..**

moreover have i⟦ *ce2* ⟧ ~ *s1* **proof** −
have i⟦ *ce2*[*x*::=*v*]$_{cev}$ ⟧ ~ *s1* **using** *assms s1* **by** *auto*
moreover have *ce1* = *ce2*[*x*::=*v*]$_{cev}$ **using** *subst-v-ce-def assms subst-v-simple-commute* **by** *auto*
ultimately have $i(x \mapsto s)$ ⟦ *ce2* ⟧ ~ *s1*
using *ix subst-e-eval-v*[*of i ce1 s1 ce2*[*z*::=[*x*]^{v}]$_{v}$ *x v s*] *iv s1* **by** *auto*
moreover have $i(x \mapsto s) = i$ **using** *ix* **by** *auto*
ultimately show *?thesis* **by** *auto*
qed
ultimately show i ⟦ *ce1* == *ce2* ⟧ ~ *True* **using** *eval-c-eqI* **by** *metis*
qed
qed

lemma *check-v-top*:
fixes *v*::*v*
assumes Θ ; $\mathcal{B}$; *GNil* ⊢ v ⇐ τ **and** *ce1* = *ce2*[*z*::=*v*]$_{cev}$ **and** Θ ; $\mathcal{B}$; *GNil* ⊢$_{wf}$ {| z : *b-of* τ | *ce1* == *ce2* |}
and *supp ce1* ⊆ *supp* $\mathcal{B}$
shows Θ ; $\mathcal{B}$; *GNil* ⊢ v ⇐ {| z : *b-of* τ | *ce1* == *ce2* |}
proof −
obtain t **where** t: Θ ; $\mathcal{B}$; *GNil* ⊢ v ⇒ t ∧ Θ ; $\mathcal{B}$; *GNil* ⊢ $t \lesssim \tau$
using *assms check-v-elims* **by** *metis*

then obtain z' **and** b' **where** ∗:t = {| z' : b' | [[z']^{v}]^{ce} == [v]^{ce} |} ∧ *atom* z' ♯ v ∧ *atom* z' ♯ *GNil*
using *assms infer-v-form* **by** *metis*
have *beq*: *b-of* t = *b-of* τ **using** *subtype-eq-base2 b-of.simps t* **by** *auto*
obtain *x*::*x* **where** *xf*: ‹*atom* x ♯ (Θ, $\mathcal{B}$, *GNil*, z', [[z']^{v}]^{ce} == [v]^{ce} , z, *ce1* == *ce2*)›
using *obtain-fresh* **by** *metis*

have Θ ; $\mathcal{B}$; (x, *b-of* τ, *TRUE*) #$_{\Gamma}$ *GNil* ⊢$_{wf}$ (*ce1*[*z*::=[*x*]^{v}]$_{v}$ == *ce2*[*z*::=[*x*]^{v}]$_{v}$)

using *wfT-wfC2*[*OF assms*(3), *of x*] *subst-cv.simps*(6) *subst-v-c-def subst-v-ce-def fresh-GNil* **by** *simp*

then obtain *b2* **where** *b2*: Θ ; $\mathcal{B}$; (*x*, *b-of t*, *TRUE*) $\#_\Gamma$ *GNil* $\vdash_{wf}$ *ce1*[*z*::=[*x*]^v]$_v$: *b2* $\wedge$
Θ ; $\mathcal{B}$; (*x*, *b-of t*, *TRUE*) $\#_\Gamma$ *GNil* $\vdash_{wf}$ *ce2*[*z*::=[*x*]^v]$_v$: *b2* **using** *wfC-elims*(3) *beq* **by** *metis*

from *xf* **have** Θ ; $\mathcal{B}$; *GNil* $\vdash$ {| *z'* : *b-of t* | [[*z'*]^v]^{ce} == [*v*]^{ce} |} $\lesssim$ {| *z* : *b-of t* | *ce1* == *ce2* |}
proof
show ‹ Θ ; $\mathcal{B}$; *GNil* $\vdash_{wf}$ {| *z'* : *b-of t* | [[*z'*]^v]^{ce} == [*v*]^{ce} |} › **using** *b-of.simps assms infer-v-wf t* $*$ **by** *auto*
show ‹ Θ ; $\mathcal{B}$; *GNil* $\vdash_{wf}$ {| *z* : *b-of t* | *ce1* == *ce2* |} › **using** *beq assms* **by** *auto*
have ‹Θ ; $\mathcal{B}$; (*x*, *b-of t*, ([[*x*]^v]^{ce} == [*v*]^{ce})) $\#_\Gamma$ *GNil* $\models$ (*ce1*[*z*::=[*x*]^v]$_v$ == *ce2*[*z*::=[*x*]^v]$_v$) ›
proof(*rule valid-ce-eq*)
show ‹*ce1*[*z*::=[*x*]^v]$_v$ = *ce2*[*z*::=[*x*]^v]$_v$[*x*::=*v*]$_{cev}$› **proof** −
have *atom z* $\sharp$ *ce1* **using** *assms fresh-def x-not-in-b-set* **by** *fast*
hence *ce1*[*z*::=[*x*]^v]$_v$ = *ce1*
using *forget-subst-v* **by** *auto*
also have ... = *ce2*[*z*::=*v*]$_{cev}$ **using** *assms* **by** *auto*
also have ... = *ce2*[*z*::=[*x*]^v]$_v$[*x*::=*v*]$_{cev}$ **proof** −
have *atom x* $\sharp$ *ce2* **using** *xf fresh-prodN c.fresh* **by** *metis*
thus *?thesis* **using** *subst-v-simple-commute subst-v-ce-def* **by** *simp*
qed
finally show *?thesis* **by** *auto*
qed
show ‹ Θ ; $\mathcal{B}$; *GNil* $\vdash_{wf}$ *v* : *b-of t* › **using** *infer-v-wf t* **by** *simp*
show ‹ Θ ; $\mathcal{B}$; (*x*, *b-of t*, *TRUE*) $\#_\Gamma$ *GNil* $\vdash_{wf}$ *ce2*[*z*::=[*x*]^v]$_v$: *b2* › **using** *b2* **by** *auto*

have Θ ; $\mathcal{B}$; (*x*, *b-of t*, *TRUE*) $\#_\Gamma$ *GNil* $\vdash_{wf}$ *ce1*[*z*::=[*x*]^v]$_v$: *b2* **using** *b2* **by** *auto*
moreover have *atom x* $\sharp$ *ce1*[*z*::=[*x*]^v]$_v$
using *fresh-subst-v-if assms fresh-def*
using ‹Θ ; $\mathcal{B}$; *GNil* $\vdash_{wf}$ *v* : *b-of t*› ‹*ce1*[*z*::=[*x*]^v]$_v$ = *ce2*[*z*::=[*x*]^v]$_v$[*x*::=*v*]$_{cev}$›
fresh-GNil subst-v-ce-def wfV-x-fresh **by** *auto*
ultimately show ‹ Θ ; $\mathcal{B}$; *GNil* $\vdash_{wf}$ *ce1*[*z*::=[*x*]^v]$_v$: *b2* › **using**
wf-restrict(8) **by** *force*
qed
moreover have *v*: *v*[*z'*::=[*x*]^v]$_{vv}$ = *v*
using *forget-subst assms infer-v-wf wfV-supp x-not-in-b-set*
by (*simp add*: *local.*$*$)
ultimately show Θ ; $\mathcal{B}$; (*x*, *b-of t*, ([[*z'*]^v]^{ce} == [*v*]^{ce})[*z'*::=[*x*]^v]$_v$) $\#_\Gamma$ *GNil* $\models$ (*ce1* == *ce2*)[*z*::=[*x*]^v]$_v$
unfolding *subst-cv.simps subst-v-c-def subst-cev.simps subst-vv.simps*
using *subst-v-ce-def* **by** *simp*
qed
thus *?thesis* **using** *b-of.simps assms* $*$ *check-v-subtypeI t b-of.simps subtype-eq-base2* **by** *metis*
qed

This lemma confirms that if we assume the existence of a boolean like datatype then if and match are the same where the latter is a match for this datatype

end

declare *freshers*[*simp del*]

Chapter 13

Context Subtyping Lemmas

Lemmas allowing us to replace the type of a variable in the context with a subtype and have the judgement remain valid. Otherwise known as narrowing.

13.1 Replace Type of Variable in Context

Because the G-context is extended by the statements like let, we will need a generalised substitution lemma for statements. For this we setup a function that replaces in G for a particular x the constraint for it

nominal-function *replace-in-g-many* :: $\Gamma \Rightarrow (x*c)\ list \Rightarrow \Gamma$ **where**
  replace-in-g-many G xcs = List.foldr $(\lambda(x,c)\ G.\ G[x \longmapsto c])$ *xcs G*
by(*auto*,*simp add*: *eqvt-def replace-in-g-many-graph-aux-def*)
nominal-termination (*eqvt*) **by** *lexicographic-order*

inductive *replace-in-g-subtyped* :: $\Theta \Rightarrow \mathcal{B} \Rightarrow \Gamma \Rightarrow (x*c)\ list \Rightarrow \Gamma \Rightarrow bool$ (- ; - ⊢ - ⟨ - ⟩ ⇝ - [*100*,*50*,*50*] *50*) **where**
  replace-in-g-subtyped-nilI: Θ ; $\mathcal{B} \vdash G \langle [] \rangle \rightsquigarrow G$
| *replace-in-g-subtyped-consI*: ⟦
    Some $(b,c') = lookup\ G\ x$;
    Θ ; $\mathcal{B}$; $G \vdash_{wf} c$;
    Θ ; $\mathcal{B}$; $G[x \longmapsto c] \models c'$;
    Θ ; $\mathcal{B} \vdash G[x \longmapsto c] \langle xcs \rangle \rightsquigarrow G'$; $x \notin fst\ `\ set\ xcs$ ⟧ ⟹
    Θ ; $\mathcal{B} \vdash G \langle (x,c)\#xcs \rangle \rightsquigarrow G'$
equivariance *replace-in-g-subtyped*
nominal-inductive *replace-in-g-subtyped* **.**

inductive-cases *replace-in-g-subtyped-elims*[*elim*!]:
  Θ ; $\mathcal{B} \vdash G \langle [] \rangle \rightsquigarrow G'$
  Θ ; $\mathcal{B} \vdash ((x,b,c)\#_\Gamma \Gamma\ G) \langle acs \rangle \rightsquigarrow ((x,b,c)\#_\Gamma G')$
  Θ ; $\mathcal{B} \vdash G' \langle (x,c)\#\ acs \rangle \rightsquigarrow G$

thm *replace-in-g-def*

lemma *rigs-atom-dom-eq*:
  assumes Θ ; $\mathcal{B} \vdash G \langle xcs \rangle \rightsquigarrow G'$
  shows *atom-dom G* = *atom-dom G'*

using *assms* **proof**(*induct rule*: *replace-in-g-subtyped.induct*)
case (*replace-in-g-subtyped-nilI G*)
then show *?case* **by** *simp*
next
case (*replace-in-g-subtyped-consI b c′ G x* Θ $\mathcal{B}$ *c xcs G′*)
then show *?case* **using** *rig-dom-eq atom-dom.simps dom.simps* **by** *simp*
qed

lemma *replace-in-g-wfG*:
assumes Θ ; $\mathcal{B}$ ⊢ *G* ⟨ *xcs* ⟩ ⇝ *G′* **and** *wfG* Θ $\mathcal{B}$ *G*
shows *wfG* Θ $\mathcal{B}$ *G′*
using *assms* **proof**(*induct rule*: *replace-in-g-subtyped.induct*)
case (*replace-in-g-subtyped-nilI* Θ *G*)
then show *?case* **by** *auto*
next
case (*replace-in-g-subtyped-consI b c′ G x* Θ *c xcs G′*)
then show *?case* **using** *valid-g-wf* **by** *auto*
qed

lemma *wfD-rig-single*:
fixes Δ::Δ **and** *x*::*x* **and** *c*::*c* **and** *G*::Γ
assumes Θ ; $\mathcal{B}$; *G* ⊢$_{wf}$ Δ **and** *wfG* Θ $\mathcal{B}$ (*G*[*x*⟼*c*])
shows Θ ; $\mathcal{B}$; *G*[*x*⟼*c*] ⊢$_{wf}$ Δ
proof(*cases atom x* ∈ *atom-dom G*)
case *False*
hence (*G*[*x*⟼*c*]) = *G* **using** *assms replace-in-g-forget wfX-wfY* **by** *metis*
then show *?thesis* **using** *assms* **by** *auto*
next
case *True*
then obtain *G1 G2 b c′* **where** ∗: *G*=*G1*@(*x*,*b*,*c′*)#$_\Gamma$*G2* **using** *split-G* **by** *fastforce*
hence ∗∗: (*G*[*x*⟼*c*]) = *G1*@(*x*,*b*,*c*)#$_\Gamma$*G2* **using** *replace-in-g-inside wfD-wf assms wfD-wf* **by** *metis*

hence *wfG* Θ $\mathcal{B}$ ((*x*,*b*,*c*)#$_\Gamma$*G2*) **using** *wfG-suffix assms* **by** *auto*
hence Θ ; $\mathcal{B}$; (*x*, *b*, *TRUE*) #$_\Gamma$ *G2* ⊢$_{wf}$ *c* **using** *wfG-elim2* **by** *auto*

thus *?thesis* **using** *wf-replace-inside1 assms* ∗ ∗∗
by (*simp add*: *wf-replace-inside2*(*6*))
qed

lemma *wfD-rig*:
assumes Θ ; $\mathcal{B}$ ⊢ *G* ⟨ *xcs* ⟩ ⇝ *G′* **and** *wfD* Θ $\mathcal{B}$ *G* Δ
shows *wfD* Θ $\mathcal{B}$ *G′* Δ
using *assms* **proof**(*induct rule*: *replace-in-g-subtyped.induct*)
case (*replace-in-g-subtyped-nilI* Θ *G*)
then show *?case* **by** *auto*
next
case (*replace-in-g-subtyped-consI b c′ G x* Θ *c xcs G′*)

then show *?case* **using** *wfD-rig-single valid.simps wfC-wf* **by** *auto*
qed

lemma *replace-in-g-fresh*:
fixes *x*::*x*
assumes Θ ; $\mathcal{B} \vdash \Gamma \langle\, xcs \,\rangle \rightsquigarrow \Gamma'$ **and** *wfG* Θ $\mathcal{B}$ Γ **and** *wfG* Θ $\mathcal{B}$ Γ′ **and** *atom x* ♯ Γ
shows *atom x* ♯ Γ′
using *wfG-dom-supp assms fresh-def rigs-atom-dom-eq* **by** *metis*

lemma *replace-in-g-fresh1*:
fixes *x*::*x*
assumes Θ ; $\mathcal{B} \vdash \Gamma \langle\, xcs \,\rangle \rightsquigarrow \Gamma'$ **and** *wfG* Θ $\mathcal{B}$ Γ **and** *atom x* ♯ Γ
shows *atom x* ♯ Γ′
proof −
have *wfG* Θ $\mathcal{B}$ Γ′ **using** *replace-in-g-wfG assms* **by** *auto*
thus *?thesis* **using** *assms replace-in-g-fresh* **by** *metis*
qed

Wellscoping for an eXchange list

inductive *wsX*:: Γ ⇒ (*x*∗*c*) *list* ⇒ *bool* **where**
wsX-NilI: *wsX G* []
| *wsX-ConsI*: ⟦ *wsX G xcs* ; *atom x* ∈ *atom-dom G* ; *x* ∉ *fst* ‘ *set xcs* ⟧ ⟹ *wsX G* ((*x*,*c*)#*xcs*)
equivariance *wsX*
nominal-inductive *wsX* **.**

lemma *wsX-if1*:
assumes *wsX G xcs*
shows ((*atom* ‘ *fst* ‘ *set xcs*) ⊆ *atom-dom G*) ∧ *List.distinct* (*List.map fst xcs*)
using *assms* **by**(*induct rule*: *wsX.induct*,*force*+)

lemma *wsX-if2*:
assumes ((*atom* ‘ *fst* ‘ *set xcs*) ⊆ *atom-dom G*) ∧ *List.distinct* (*List.map fst xcs*)
shows *wsX G xcs*
using *assms* **proof**(*induct xcs*)
case *Nil*
then show *?case* **using** *wsX-NilI* **by** *fast*
next
case (*Cons a xcs*)
then obtain *x* **and** *c* **where** *xc*: *a*=(*x*,*c*) **by** *force*
have *wsX G xcs* **proof** −
have *distinct* (*map fst xcs*) **using** *Cons* **by** *force*
moreover have *atom* ‘ *fst* ‘ *set xcs* ⊆ *atom-dom G* **using** *Cons* **by** *simp*
ultimately show *?thesis* **using** *Cons* **by** *fast*
qed
moreover have *atom x* ∈ *atom-dom G* **using** *Cons xc*
by *simp*
moreover have *x* ∉ *fst* ‘ *set xcs* **using** *Cons xc*
by *simp*
ultimately show *?case* **using** *wsX-ConsI xc* **by** *blast*
qed

lemma *wsX-iff*:

wsX G xcs = (((*atom ‘ fst ‘ set xcs*) ⊆ *atom-dom G*) ∧ *List.distinct* (*List.map fst xcs*))
using *wsX-if1 wsX-if2* **by** *meson*

inductive-cases *wsX-elims*[*elim*!]:
wsX G []
wsX G ((*x*,*c*)#*xcs*)

lemma *wsX-cons*:
assumes *wsX* Γ *xcs* **and** *x* ∉ *fst ‘ set xcs*
shows *wsX* ((*x*, *b*, *c1*) #Γ Γ) ((*x*, *c2*) # *xcs*)
using *assms* **proof**(*induct* Γ)
case *GNil*
then show *?case* **using** *atom-dom.simps wsX-iff* **by** *auto*
next
case (*GCons xbc* Γ)
obtain *x′* **and** *b′* **and** *c′* **where** *xbc*: *xbc* = (*x′*,*b′*,*c′*) **using** *prod-cases3* **by** *blast*
then have *atom ‘ fst ‘ set xcs* ⊆ *atom-dom* (*xbc* #Γ Γ) ∧ *distinct* (*map fst xcs*)
using *GCons.prems*(*1*) *wsX-iff* **by** *blast*
then have *wsX* ((*x*, *b*, *c1*) #Γ *xbc* #Γ Γ) *xcs*
by (*simp add*: *Un-commute subset-Un-eq wsX-if2*)
then show *?case* **by** (*simp add*: *GCons.prems*(*2*) *wsX-ConsI*)
qed

lemma *wsX-cons2*:
assumes *wsX* Γ *xcs* **and** *x* ∉ *fst ‘ set xcs*
shows *wsX* ((*x*, *b*, *c1*) #Γ Γ) *xcs*
using *assms* **proof**(*induct* Γ)
case *GNil*
then show *?case* **using** *atom-dom.simps wsX-iff* **by** *auto*
next
case (*GCons xbc* Γ)
obtain *x′* **and** *b′* **and** *c′* **where** *xbc*: *xbc* = (*x′*,*b′*,*c′*) **using** *prod-cases3* **by** *blast*
then have *atom ‘ fst ‘ set xcs* ⊆ *atom-dom* (*xbc* #Γ Γ) ∧ *distinct* (*map fst xcs*)
using *GCons.prems*(*1*) *wsX-iff* **by** *blast* **then show** *?case* **by** (*simp add*: *Un-commute subset-Un-eq wsX-if2*)
qed

lemma *wsX-cons3*:
assumes *wsX* Γ *xcs*
shows *wsX* ((*x*, *b*, *c1*) #Γ Γ) *xcs*
using *assms* **proof**(*induct* Γ)
case *GNil*
then show *?case* **using** *atom-dom.simps wsX-iff* **by** *auto*
next
case (*GCons xbc* Γ)
obtain *x′* **and** *b′* **and** *c′* **where** *xbc*: *xbc* = (*x′*,*b′*,*c′*) **using** *prod-cases3* **by** *blast*
then have *atom ‘ fst ‘ set xcs* ⊆ *atom-dom* (*xbc* #Γ Γ) ∧ *distinct* (*map fst xcs*)
using *GCons.prems*(*1*) *wsX-iff* **by** *blast* **then show** *?case* **by** (*simp add*: *Un-commute subset-Un-eq wsX-if2*)
qed

lemma *wsX-fresh*:

assumes *wsX G xcs* **and** *atom x* $\sharp$ *G* **and** *wfG* Θ $\mathcal{B}$ *G*
shows $x \notin$ *fst ' set xcs*
proof −
have *atom x* $\notin$ *atom-dom G* **using** *assms*
using *fresh-def wfG-dom-supp* **by** *auto*
thus *?thesis* **using** *wsX-iff assms* **by** *blast*
qed

lemma *replace-in-g-dist*:
assumes $x' \neq x$
shows *replace-in-g* ((*x*, *b*,*c*) $\#_\Gamma$ *G*) *x' c''* = ((*x*, *b*,*c*) $\#_\Gamma$ (*replace-in-g G x' c''*)) **using** *replace-in-g.simps assms* **by** *presburger*

lemma *wfG-replace-inside-rig*:
fixes *c''*::*c*
assumes ‹Θ ; $\mathcal{B}$ $\vdash_{wf}$ *G*[*x'*$\longmapsto$*c''*]› ‹Θ ; $\mathcal{B}$ $\vdash_{wf}$ (*x*, *b*, *c*) $\#_\Gamma$ *G* ›
shows Θ ; $\mathcal{B}$ $\vdash_{wf}$ (*x*, *b*, *c*) $\#_\Gamma$ *G*[*x'*$\longmapsto$*c''*]
proof(*rule wfG-consI*)

have *wfG* Θ $\mathcal{B}$ *G* **using** *wfG-cons assms* **by** *auto*

show ∗:Θ ; $\mathcal{B}$ $\vdash_{wf}$ *G*[*x'*$\longmapsto$*c''*] **using** *assms* **by** *auto*
show *atom x* $\sharp$ *G*[*x'*$\longmapsto$*c''*] **using** *replace-in-g-fresh-single*[*OF* ∗] *assms wfG-elims assms* **by** *metis*
show ∗∗:Θ ; $\mathcal{B}$ $\vdash_{wf}$ *b* **using** *wfG-elim2 assms* **by** *auto*
show Θ ; $\mathcal{B}$; (*x*, *b*, *TRUE*) $\#_\Gamma$ *G*[*x'*$\longmapsto$*c''*] $\vdash_{wf}$ *c*
proof(*cases atom x'* $\notin$ *atom-dom G*)
case *True*
hence *G* = *G*[*x'*$\longmapsto$*c''*] **using** *replace-in-g-forget* ‹*wfG* Θ $\mathcal{B}$ *G*› **by** *auto*
thus *?thesis* **using** *assms wfG-wfC* **by** *auto*
next
case *False*
then obtain *G1 G2 b' c'* **where** ∗∗:*G*=*G1*@(*x'*,*b'*,*c'*)$\#_\Gamma$*G2*
using *split-G* **by** *fastforce*
hence ∗∗∗: (*G*[*x'*$\longmapsto$*c''*]) = *G1*@(*x'*,*b'*,*c''*)$\#_\Gamma$*G2*
using *replace-in-g-inside* ‹*wfG* Θ $\mathcal{B}$ *G* › **by** *metis*
hence Θ ; $\mathcal{B}$; (*x*, *b*, *TRUE*) $\#_\Gamma$ *G1*@(*x'*,*b'*,*c'*)$\#_\Gamma$*G2* $\vdash_{wf}$ *c* **using** ∗ ∗∗ *assms wfG-wfC* **by** *auto*
hence Θ ; $\mathcal{B}$; (*x*, *b*, *TRUE*) $\#_\Gamma$ *G1*@(*x'*,*b'*,*c''*)$\#_\Gamma$*G2* $\vdash_{wf}$ *c* **using** ∗ ∗∗∗ *wf-replace-inside assms*
by (*metis* ∗∗ *append-g.simps*(*2*) *wfG-elim2 wfG-suffix*)
thus *?thesis* **using** ∗∗ ∗ ∗∗∗ **by** *auto*
qed
qed

lemma *replace-in-g-valid-weakening*:
assumes Θ ; $\mathcal{B}$; Γ[*x'*$\longmapsto$*c''*] $\models$ *c'* **and** *x'* $\neq$ *x* **and** Θ ; $\mathcal{B}$ $\vdash_{wf}$ (*x*, *b*, *c*) $\#_\Gamma$ Γ[*x'*$\longmapsto$*c''*]
shows Θ ; $\mathcal{B}$; ((*x*, *b*,*c*) $\#_\Gamma$ Γ)[*x'*$\longmapsto$ *c''*] $\models$ *c'*
apply(*subst replace-in-g-dist*,*simp add*: *assms*,*rule valid-weakening*)
using *assms* **by** *auto*+

lemma *replace-in-g-subtyped-cons*:
assumes *replace-in-g-subtyped* Θ $\mathcal{B}$ *G xcs G'* **and** *wfG* Θ $\mathcal{B}$ ((*x*,*b*,*c*)$\#_\Gamma$*G*)
shows *x* $\notin$ *fst ' set xcs* $\Longrightarrow$ *replace-in-g-subtyped* Θ $\mathcal{B}$ ((*x*,*b*,*c*)$\#_\Gamma$*G*) *xcs* ((*x*,*b*,*c*)$\#_\Gamma$*G'*)
using *assms* **proof**(*induct rule*: *replace-in-g-subtyped.induct*)

case (*replace-in-g-subtyped-nilI G*)
then show *?case*
by (*simp add*: *replace-in-g-subtyped.replace-in-g-subtyped-nilI*)
next
case (*replace-in-g-subtyped-consI b' c' G x' Θ B c'' xcs' G'*)
hence Θ ; $\mathcal{B} \vdash_{wf} G[x' \longmapsto c'']$ **using** *valid.simps wfC-wf* **by** *auto*

show *?case* **proof**(*rule replace-in-g-subtyped.replace-in-g-subtyped-consI*)
show *Some* (*b'*, *c'*) = *lookup* ((*x*, *b*,*c*) #$_\Gamma$ *G*) *x'* **using** *lookup.simps*
fst-conv image-iff Γ*-set-intros surj-pair replace-in-g-subtyped-consI* **by** *force*
show *wbc*: Θ ; $\mathcal{B}$; (*x*, *b*, *c*) #$_\Gamma$ *G* $\vdash_{wf}$ *c''* **using** *wf-weakening* ‹ Θ ; $\mathcal{B}$; *G* $\vdash_{wf}$ *c''*› ‹Θ ; $\mathcal{B}$ $\vdash_{wf}$ (*x*, *b*, *c*) #$_\Gamma$ *G* › **by** *fastforce*
have *x'* ≠ *x* **using** *replace-in-g-subtyped-consI* **by** *auto*
have *wbc1*: Θ ; $\mathcal{B} \vdash_{wf}$ (*x*, *b*, *c*) #$_\Gamma$ *G*[*x'*⟼*c''*] **proof** −
have (*x*, *b*, *c*) #$_\Gamma$ *G*[*x'*⟼*c''*] = ((*x*, *b*, *c*) #$_\Gamma$ *G*)[*x'*⟼*c''*] **using** ‹*x'* ≠ *x*› **using** *replace-in-g.simps* **by** *auto*
thus *?thesis* **using** *wfG-replace-inside-rig* ‹Θ ; $\mathcal{B} \vdash_{wf}$ *G*[*x'*⟼*c''*]› ‹Θ ; $\mathcal{B}$ $\vdash_{wf}$ (*x*, *b*, *c*) #$_\Gamma$ *G* › **by** *fastforce*
qed
show ∗: Θ ; $\mathcal{B}$; *replace-in-g* ((*x*, *b*,*c*) #$_\Gamma$ *G*) *x' c''* ⊨ *c'*
proof −
have Θ ; $\mathcal{B}$; *G*[*x'*⟼*c''*] ⊨ *c'* **using** *replace-in-g-subtyped-consI* **by** *auto*
thus *?thesis* **using** *replace-in-g-valid-weakening wbc1* ‹*x'*≠*x*› **by** *auto*
qed

show *replace-in-g-subtyped* Θ $\mathcal{B}$ (*replace-in-g* ((*x*, *b*,*c*) #$_\Gamma$ *G*) *x' c''*) *xcs'* ((*x*, *b*,*c*) #$_\Gamma$ *G'*)
using *replace-in-g-subtyped-consI wbc1* **by** *auto*
show *x'* ∉ *fst* ‘ *set xcs'*
using *replace-in-g-subtyped-consI* **by** *linarith*
qed
qed

lemma *replace-in-g-split*:
fixes *G*::Γ
assumes Γ = *replace-in-g* Γ' *x c* **and** Γ' = *G'*@(*x*,*b*,*c'*)#$_\Gamma$*G* **and** *wfG* Θ $\mathcal{B}$ Γ'
shows Γ = *G'*@(*x*,*b*,*c*)#$_\Gamma$*G*
using *assms* **proof**(*induct G' arbitrary*: *G* Γ Γ' *rule*: Γ*-induct*)
case *GNil*
then show *?case* **by** *simp*
next
case (*GCons x1 b1 c1* Γ*1*)
hence *x1* ≠ *x*
using *wfG-cons-fresh2*[*of* Θ $\mathcal{B}$ *x1 b1 c1* Γ*1 x b*]
using *GCons.prems*(*2*) *GCons.prems*(*3*) *append-g.simps*(*2*) **by** *auto*
moreover hence ∗: Θ ; $\mathcal{B}$ $\vdash_{wf}$ (Γ*1* @ (*x*, *b*, *c'*) #$_\Gamma$ *G*) **using** *GCons append-g.simps wfG-elims* **by** *metis*
moreover hence *replace-in-g* (Γ*1* @ (*x*, *b*, *c'*) #$_\Gamma$ *G*) *x c* = Γ*1* @ (*x*, *b*, *c*) #$_\Gamma$ *G* **using** *GCons replace-in-g-inside*[*OF* ∗, *of c*] **by** *auto*

ultimately show *?case* **using** *replace-in-g.simps*(*2*)[*of x1 b1 c1* Γ*1* @ (*x*, *b*, *c'*) #$_\Gamma$ *G x c*] *GCons*
by (*simp add*: *GCons.prems*(*1*) *GCons.prems*(*2*))

qed

lemma *replace-in-g-subtyped-split0*:
fixes $G::\Gamma$
assumes *replace-in-g-subtyped* $\Theta\ \mathcal{B}\ \Gamma'[(x,c)]\ \Gamma$ **and** $\Gamma' = G'@(x,b,c')\#_\Gamma G$ **and** *wfG* $\Theta\ \mathcal{B}\ \Gamma'$
shows $\Gamma = G'@(x,b,c)\#_\Gamma G$
proof −
have $\Gamma =$ *replace-in-g* $\Gamma'\ x\ c$ **using** *assms replace-in-g-subtyped.simps*
by (*metis Pair-inject list.distinct*(*1*) *list.inject*)
thus *?thesis* **using** *assms replace-in-g-split* **by** *blast*
qed

lemma *replace-in-g-subtyped-split*:
assumes *Some* $(b,\ c') =$ *lookup* $G\ x$ **and** $\Theta\ ;\ \mathcal{B}\ ;$ *replace-in-g* $G\ x\ c\ \models c'$ **and** *wfG* $\Theta\ \mathcal{B}\ G$
shows $\exists\ \Gamma\ \Gamma'.\ G = \Gamma'@(x,b,c')\#_\Gamma\Gamma \wedge \Theta\ ;\ \mathcal{B}\ ;\ \Gamma'@(x,b,c)\#_\Gamma\Gamma \models c'$
proof −
obtain Γ **and** Γ' **where** $G = \Gamma'@(x,b,c')\#_\Gamma\Gamma$ **using** *assms lookup-split* **by** *blast*
moreover hence *replace-in-g* $G\ x\ c = \Gamma'@(x,b,c)\#_\Gamma\Gamma$ **using** *replace-in-g-split assms* **by** *blast*
ultimately show *?thesis* **by** (*metis assms*(*2*))
qed

13.2 Validity and Subtyping

lemma *wfC-replace-in-g*:
fixes $c::c$ **and** $c0::c$
assumes $\Theta\ ;\ \mathcal{B}\ ;\ \Gamma'@(x,b,c0')\#_\Gamma\Gamma \vdash_{wf} c$ **and** $\Theta\ ;\ \mathcal{B}\ ;\ (x,b,TRUE)\#_\Gamma\Gamma \vdash_{wf} c0$
shows $\Theta\ ;\ \mathcal{B}\ ;\ \Gamma' @ (x,\ b,\ c0)\ \#_\Gamma\ \Gamma \vdash_{wf} c$
using *wf-replace-inside1*(*2*) *assms* **by** *auto*

lemma *ctx-subtype-valid*:
assumes $\Theta\ ;\ \mathcal{B}\ ;\ \Gamma'@(x,b,c0')\#_\Gamma\Gamma \models c$ **and**
$\Theta\ ;\ \mathcal{B}\ ;\ \Gamma'@(x,b,c0)\#_\Gamma\Gamma \models c0'$
shows $\Theta\ ;\ \mathcal{B}\ ;\ \Gamma'@(x,b,c0)\#_\Gamma\Gamma \models c$
proof(*rule validI*)
show $\Theta\ ;\ \mathcal{B}\ ;\ \Gamma' @ (x,\ b,\ c0)\ \#_\Gamma\ \Gamma \vdash_{wf} c$ **proof** −
have $\Theta\ ;\ \mathcal{B}\ ;\ \Gamma'@(x,b,c0')\#_\Gamma\Gamma \vdash_{wf} c$ **using** *valid.simps assms* **by** *auto*
moreover have $\Theta\ ;\ \mathcal{B}\ ;\ (x,b,TRUE)\#_\Gamma\Gamma \vdash_{wf} c0$ **proof** −
have *wfG* $\Theta\ \mathcal{B}\ (\Gamma'@(x,b,c0)\#_\Gamma\Gamma)$ **using** *assms valid.simps wfC-wf* **by** *auto*
hence *wfG* $\Theta\ \mathcal{B}\ ((x,b,c0)\#_\Gamma\Gamma)$ **using** *wfG-suffix* **by** *auto*
thus *?thesis* **using** *wfG-wfC* **by** *auto*
qed
ultimately show *?thesis* **using** *assms wfC-replace-in-g* **by** *auto*
qed

show $\forall i.$ *wfI* $\Theta\ (\Gamma' @ (x,\ b,\ c0)\ \#_\Gamma\ \Gamma)\ i \wedge$ *is-satis-g* $i\ (\Gamma' @ (x,\ b,\ c0)\ \#_\Gamma\ \Gamma) \longrightarrow$ *is-satis* $i\ c$
proof(*rule*,*rule*)
fix i
assume $*$: *wfI* $\Theta\ (\Gamma' @ (x,\ b,\ c0)\ \#_\Gamma\ \Gamma)\ i \wedge$ *is-satis-g* $i\ (\Gamma' @ (x,\ b,\ c0)\ \#_\Gamma\ \Gamma)$

hence *is-satis-g* $i\ (\Gamma'@(x,\ b,\ c0)\ \#_\Gamma\ \Gamma) \wedge$ *wfI* $\Theta\ (\Gamma'@(x,\ b,\ c0)\ \#_\Gamma\ \Gamma)\ i$ **using** *is-satis-g-append wfI-suffix* **by** *metis*

moreover hence *is-satis i c0'* **using** *valid.simps assms* **by** *presburger*

moreover have *is-satis-g i* Γ' **using** *is-satis-g-append* $*$ **by** *simp*
ultimately have *is-satis-g i* $(\Gamma' @ (x, b, c0') \#_\Gamma \Gamma)$ **using** *is-satis-g-append* **by** *simp*
moreover have *wfI* Θ $(\Gamma' @ (x, b, c0') \#_\Gamma \Gamma)$ *i* **using** *wfI-def wfI-suffix* $*$ *wfI-def wfI-replace-inside* **by** *metis*
ultimately show *is-satis i c* **using** *assms valid.simps* **by** *metis*
qed
qed

lemma *ctx-subtype-subtype*:
fixes Γ::Γ
shows Θ ; $\mathcal{B}$; $G \vdash t1 \lesssim t2 \Longrightarrow G = \Gamma'@(x,b0,c0')\#_\Gamma\Gamma \Longrightarrow \Theta$; $\mathcal{B}$; $\Gamma'@(x,b0,c0)\#_\Gamma\Gamma \models c0' \Longrightarrow \Theta$; $\mathcal{B}$; $\Gamma'@(x,b0,c0)\#_\Gamma\Gamma \vdash t1 \lesssim t2$
proof(*nominal-induct avoiding*: *c0 rule*: *subtype.strong-induct*)

case (*subtype-baseI* x' Θ $\mathcal{B}$ Γ'' z c z' c' b)
let $?\Gamma c0 = \Gamma'@(x,b0,c0)\#_\Gamma\Gamma$
have *wb1*: *wfG* Θ $\mathcal{B}$ $?\Gamma c0$ **using** *valid.simps wfC-wf subtype-baseI* **by** *metis*
show *?case* **proof**
show $\langle$ Θ ; $\mathcal{B}$; Γ' @ $(x, b0, c0)$ $\#_\Gamma$ Γ $\vdash_{wf}$ $\{\!|$ z : b | c $|\!\}$ $\rangle$ **using** *wfT-replace-inside2*[*OF - wb1*] *subtype-baseI* **by** *metis*
show $\langle$ Θ ; $\mathcal{B}$; Γ' @ $(x, b0, c0)$ $\#_\Gamma$ Γ $\vdash_{wf}$ $\{\!|$ z' : b | c' $|\!\}$ $\rangle$ **using** *wfT-replace-inside2*[*OF - wb1*] *subtype-baseI* **by** *metis*
have *atom* $x' \sharp \Gamma'$ @ $(x, b0, c0)$ $\#_\Gamma$ Γ **using** *fresh-prodN subtype-baseI fresh-replace-inside wb1 subtype-wf wfX-wfY* **by** *metis*
thus $\langle$*atom* $x' \sharp (\Theta, \mathcal{B}, \Gamma'$ @ $(x, b0, c0)$ $\#_\Gamma$ Γ, z, c , z' , c' $)\rangle$ **using** *subtype-baseI fresh-prodN* **by** *metis*
have Θ ; $\mathcal{B}$; $((x', b, c[z::=V\text{-}var\ x']_v)$ $\#_\Gamma$ $\Gamma')$ @ $(x, b0, c0)$ $\#_\Gamma$ Γ $\models c'[z'::=V\text{-}var\ x']_v$ **proof**(*rule ctx-subtype-valid*)
show *1*: $\langle\Theta$; $\mathcal{B}$; $((x', b, c[z::=V\text{-}var\ x']_v)$ $\#_\Gamma$ $\Gamma')$ @ $(x, b0, c0')$ $\#_\Gamma$ Γ $\models c'[z'::=V\text{-}var\ x']_v$ $\rangle$
using *subtype-baseI append-g.simps subst-defs* **by** *metis*
have $*$:Θ ; $\mathcal{B} \vdash_{wf} ((x', b, c[z::=V\text{-}var\ x']_v)$ $\#_\Gamma$ $\Gamma')$ @ $(x, b0, c0)$ $\#_\Gamma$ Γ **proof**(*rule wfG-replace-inside2*)
show Θ ; $\mathcal{B} \vdash_{wf} ((x', b, c[z::=V\text{-}var\ x']_v)$ $\#_\Gamma$ $\Gamma')$ @ $(x, b0, c0')$ $\#_\Gamma$ Γ
using $*$ *valid-wf-all wfC-wf 1 append-g.simps* **by** *metis*
show Θ ; $\mathcal{B} \vdash_{wf}$ $(x, b0, c0)$ $\#_\Gamma$ Γ **using** *wfG-suffix wb1* **by** *auto*
qed
moreover have *setG* $(\Gamma'$ @ $(x, b0, c0)$ $\#_\Gamma$ $\Gamma) \subseteq$ *setG* $(((x', b, c[z::=V\text{-}var\ x']_v)$ $\#_\Gamma$ $\Gamma')$ @ $(x, b0, c0)$ $\#_\Gamma$ $\Gamma)$ **using** *setG.simps append-g.simps* **by** *auto*
ultimately show $\langle\Theta$; $\mathcal{B}$; $((x', b, c[z::=V\text{-}var\ x']_v)$ $\#_\Gamma$ $\Gamma')$ @ $(x, b0, c0)$ $\#_\Gamma$ Γ $\models c0'$ $\rangle$ **using** *valid-weakening subtype-baseI* $*$ **by** *blast*
qed
thus $\langle\Theta$; $\mathcal{B}$; $(x', b, c[z::=V\text{-}var\ x']_v)$ $\#_\Gamma$ Γ' @ $(x, b0, c0)$ $\#_\Gamma$ Γ $\models c'[z'::=V\text{-}var\ x']_v$ $\rangle$ **using** *append-g.simps subst-defs* **by** *simp*
qed
qed

lemma *ctx-subtype-subtype-rig*:
assumes *replace-in-g-subtyped* Θ $\mathcal{B}$ Γ' $[(x,c0)]$ Γ **and** Θ ; $\mathcal{B}$; $\Gamma' \vdash t1 \lesssim t2$
shows Θ ; $\mathcal{B}$; $\Gamma \vdash t1 \lesssim t2$
proof −

have *wf*: *wfG* Θ $\mathcal{B}$ Γ' **using** *subtype-g-wf assms* **by** *auto*
obtain *b* **and** *c0'* **where** *Some* (*b*, *c0'*) = *lookup* Γ' *x* $\wedge$ (Θ ; $\mathcal{B}$; *replace-in-g* Γ' *x c0* $\models$ *c0'*) **using**
replace-in-g-subtyped.simps[*of* Θ $\mathcal{B}$ Γ' [(*x*, *c0*)] Γ] *assms*(*1*)

by (*metis fst-conv list.inject list.set-intros*(*1*) *list.simps*(*15*) *not-Cons-self2 old.prod.exhaust prod.inject set-ConsD surj-pair*)
moreover then obtain *G* **and** *G'* **where** $*$: $\Gamma' = G'@(x,b,c0')\#_\Gamma G \wedge \Theta$; $\mathcal{B}$; $G'@(x,b,c0)\#_\Gamma G \models c0'$
using *replace-in-g-subtyped-split*[*of b c0'* Γ' *x* Θ $\mathcal{B}$ *c0*] *wf* **by** *metis*

ultimately show *?thesis* **using** *ctx-subtype-subtype*
assms(*1*) *assms*(*2*) *replace-in-g-subtyped-split0 subtype-g-wf*
by (*metis* (*no-types*, *lifting*) *local.wf replace-in-g-split*)
qed

We now prove versions of the *ctx-subtype* lemmas above using *replace-in-g*. First we do case where the replace is just for a single variable (indicated by suffix rig) and then the general case for multiple replacements (indicated by suffix rigs)

lemma *ctx-subtype-subtype-rigs*:
assumes *replace-in-g-subtyped* Θ $\mathcal{B}$ Γ' *xcs* Γ **and** Θ ; $\mathcal{B}$; $\Gamma' \vdash$ *t1* $\lesssim$ *t2*
shows Θ ; $\mathcal{B}$; $\Gamma \vdash$ *t1* $\lesssim$ *t2*
using *assms* **proof**(*induct xcs arbitrary*: Γ Γ')
case *Nil*
moreover have $\Gamma' = \Gamma$ **using** *replace-in-g-subtyped-nilI*
using *calculation*(*1*) **by** *blast*
ultimately show *?case* **by** *auto*
next
case (*Cons a xcs*)

then obtain *x* **and** *c* **where** *a*=(*x*,*c*) **by** *fastforce*
then obtain *b* **and** *c'* **where** *bc*: *Some* (*b*, *c'*) = *lookup* Γ' *x* $\wedge$
replace-in-g-subtyped Θ $\mathcal{B}$ (*replace-in-g* Γ' *x c*) *xcs* Γ $\wedge$ Θ ; $\mathcal{B}$; $\Gamma' \vdash_{wf}$ *c* $\wedge$
x $\notin$ *fst* ' *set xcs* $\wedge$ Θ ; $\mathcal{B}$; (*replace-in-g* Γ' *x c*) $\models$ *c'* **using** *replace-in-g-subtyped-elims*(*3*)[*of* Θ $\mathcal{B}$ Γ' *x c xcs* Γ] *Cons*
by (*metis valid.simps*)

hence $*$: *replace-in-g-subtyped* Θ $\mathcal{B}$ Γ' [(*x*,*c*)] (*replace-in-g* Γ' *x c*) **using** *replace-in-g-subtyped-consI*
by (*meson image-iff list.distinct*(*1*) *list.set-cases replace-in-g-subtyped-nilI*)

hence Θ ; $\mathcal{B}$; (*replace-in-g* Γ' *x c*) $\vdash$ *t1* $\lesssim$ *t2*
using *ctx-subtype-subtype-rig* $*$ *assms Cons.prems*(*2*) **by** *auto*

moreover have *replace-in-g-subtyped* Θ $\mathcal{B}$ (*replace-in-g* Γ' *x c*) *xcs* Γ **using** *Cons*
using *bc* **by** *blast*

ultimately show *?case* **using** *Cons* **by** *blast*
qed

lemma *replace-in-g-inside-valid*:
assumes *replace-in-g-subtyped* Θ $\mathcal{B}$ Γ' [(*x*,*c0*)] Γ **and** *wfG* Θ $\mathcal{B}$ Γ'
shows $\exists$ *b c0' G G'*. $\Gamma' = G'$ @ (*x*,*b*,*c0'*)$\#_\Gamma G \wedge$ $\Gamma = G'$ @ (*x*,*b*,*c0*)$\#_\Gamma G \wedge \Theta$; $\mathcal{B}$; G'@ (*x*,*b*,*c0*)$\#_\Gamma G$ $\models$ *c0'*

proof −
obtain *b* **and** *c0'* **where** *bc*: *Some* (*b*, *c0'*) = *lookup* Γ' *x* ∧ Θ ; $\mathcal{B}$; *replace-in-g* Γ' *x c0* ⊨ *c0'*
using
replace-in-g-subtyped.simps[*of* Θ $\mathcal{B}$ Γ' [(*x*, *c0*)] Γ] *assms*(*1*)
by (*metis fst-conv list.inject list.set-intros*(*1*) *list.simps*(*15*) *not-Cons-self2 old.prod.exhaust prod.inject set-ConsD surj-pair*)
then obtain *G* **and** *G'* **where** ∗: Γ' = *G'*@(*x*,*b*,*c0'*)#Γ *G* ∧ Θ ; $\mathcal{B}$; *G'*@(*x*,*b*,*c0*)#Γ *G* ⊨ *c0'* **using** *replace-in-g-subtyped-split*[*of b c0'* Γ' *x* Θ $\mathcal{B}$ *c0*] *assms*
by *metis*
thus *?thesis* **using** *replace-in-g-inside bc*
using *assms*(*1*) *assms*(*2*) **by** *blast*
qed

lemma *replace-in-g-valid*:
assumes Θ ; $\mathcal{B}$ ⊢ *G* ⟨ *xcs* ⟩ ⇝ *G'* **and** Θ ; $\mathcal{B}$; *G* ⊨ *c*
shows ‹Θ ; $\mathcal{B}$; *G'* ⊨ *c* ›
using *assms* **proof**(*induct rule*: *replace-in-g-subtyped.inducts*)
case (*replace-in-g-subtyped-nilI* Θ $\mathcal{B}$ *G*)
then show *?case* **by** *auto*
next
case (*replace-in-g-subtyped-consI b c1 G x* Θ $\mathcal{B}$ *c2 xcs G'*)
hence Θ ; $\mathcal{B}$; *G*[*x*⟼*c2*] ⊨ *c*
by (*metis ctx-subtype-valid replace-in-g-split replace-in-g-subtyped-split valid-g-wf*)
then show *?case* **using** *replace-in-g-subtyped-consI* **by** *auto*
qed

13.3 Literals

13.4 Values

lemma *lookup-inside-unique-b*[*simp*]:
assumes Θ ; *B* ⊢$_{wf}$ (Γ'@(*x*,*b0*,*c0*)#Γ) **and** Θ ; *B* ⊢$_{wf}$ (Γ'@(*x*,*b0*,*c0'*)#Γ)
and *Some* (*b*, *c*) = *lookup* (Γ' @ (*x*, *b0*, *c0'*) #Γ Γ) *y* **and** *Some* (*b0*,*c0*) = *lookup* (Γ'@((*x*,*b0*,*c0*))#Γ) *x* **and** *x*=*y*
shows *b* = *b0*
by (*metis assms*(*2*) *assms*(*3*) *assms*(*5*) *lookup-inside-wf old.prod.exhaust option.inject prod.inject*)

I think using rule induction for values and expressions is only going to save us from doing the elimination step

lemma *ctx-subtype-v*:
fixes *v*::*v*
assumes
Θ ; $\mathcal{B}$; Γ'@((*x*,*b0*,*c0'*)#Γ) ⊢ *v* ⇒ *t1* **and** Θ ; $\mathcal{B}$; Γ'@(*x*,*b0*,*c0*)#Γ ⊨ *c0'*
shows ∃ *t2*. Θ ; $\mathcal{B}$; Γ'@((*x*,*b0*,*c0*)#Γ) ⊢ *v* ⇒ *t2* ∧ Θ ; $\mathcal{B}$; Γ'@((*x*,*b0*,*c0*)#Γ) ⊢ *t2* ≲ *t1*
using *assms* **proof**(*nominal-induct v arbitrary*: *t1* *rule*: *v.strong-induct*)
case (*V-lit l*)
have ⊢ *l* ⇒ *t1* **using** *V-lit infer-v-elims* **by** *force*
hence Θ ; $\mathcal{B}$; Γ' @ (*x*, *b0*, *c0*) #Γ Γ ⊢ *V-lit l* ⇒ *t1*
using *infer-v-litI* *V-lit valid.simps wfC-wf* **by** *metis*
moreover hence Θ ; $\mathcal{B}$; Γ'@((*x*,*b0*,*c0*)#Γ) ⊢ *t1* ≲ *t1* **using** *infer-v-wf*
by (*meson subtype-reflI2*)

ultimately show *?case* **using** $*$ **by** *metis*
next
case (*V-var y*)
have *wfg0*: *wfG* Θ $\mathcal{B}$ (Γ' @ (x, *b0*, *c0'*) $\#_\Gamma$ Γ) **using** *infer-v-wf V-var* **by** *fast*
hence *wfg1*: *wfG* Θ $\mathcal{B}$ (Γ'@(x,*b0*,*c0*)$\#_\Gamma\Gamma$) **using** *V-var wfG-inside-valid2* **by** *metis*

obtain z **and** b **and** c **where** *zb*: *t1* = ({| z : b | *C-eq* (*CE-val* (*V-var z*)) (*CE-val* (*V-var y*)) |}) $\wedge$ *atom* z $\sharp$ y $\wedge$ *atom* z $\sharp$ (Γ' @ (x, *b0*, *c0'*) $\#_\Gamma$ Γ) $\wedge$ *Some* (b, c) = *lookup* (Γ' @ (x, *b0*, *c0'*) $\#_\Gamma$ Γ) y
using *infer-v-elims*(*1*)[*OF V-var*(*1*)] **by** *metis*
hence *lu1*: *Some* (b, c) = *lookup* (Γ' @ (x, *b0*, *c0'*) $\#_\Gamma$ Γ) y **by** *auto*

show *?case* **proof**(*cases* x = y)
case *True*
have *lu*: *Some* (*b0*,*c0*) = *lookup* (Γ'@((*x*,*b0*,*c0*))$\#_\Gamma\Gamma$) x **using** *lookup-inside-wf wfg1* **by** *metis*
moreover hence *b0* = b **using** *lu1 True lookup-inside-unique-b*
using ‹*wfG* Θ $\mathcal{B}$ (Γ' @ (x, *b0*, *c0'*) $\#_\Gamma$ Γ)› *wfg1* **by** *metis*
moreover have *atom* z $\sharp$ x **using** *True zb* **by** *simp*
moreover have *atom* z $\sharp$ Γ'@((*x*,*b0*,*c0*)$\#_\Gamma\Gamma$) **using** *zb fresh-replace-inside wfg0 wfg1* **by** *metis*
ultimately have Θ ; $\mathcal{B}$; Γ'@((*x*,*b0*,*c0*)$\#_\Gamma\Gamma$) $\vdash$ (*V-var x*) $\Rightarrow$ ({| z : b | *C-eq* (*CE-val* (*V-var z*)) (*CE-val* (*V-var x*)) |})
using *infer-v-varI wfg1* **by** *metis*
thus *?thesis*
using *True infer-v-t-wf subtype-reflI2 zb* **by** *metis*
next
case *False*
then obtain *b1* **and** *c1* **where** *bc*: *Some* (*b1*,*c1*) = *lookup* (Γ'@((*x*,*b0*,*c0'*)$\#_\Gamma\Gamma$)) y
using *infer-v-elims V-var* **by** *meson*

have Θ ; $\mathcal{B}$; Γ'@((*x*,*b0*,*c0*)$\#_\Gamma\Gamma$) $\vdash$ (*V-var y*) $\Rightarrow$ ({| z : *b1* | *C-eq* (*CE-val* (*V-var z*)) (*CE-val* (*V-var y*)) |}) **proof**
show Θ ; $\mathcal{B}$ $\vdash_{wf}$ Γ' @ (x, *b0*, *c0*) $\#_\Gamma$ Γ **using** *wfg1* **by** *auto*
show *Some* (*b1*, *c1*) = *lookup* (Γ' @ (x, *b0*, *c0*) $\#_\Gamma$ Γ) y **using** *lookup-inside2 False bc* **by** *blast*
show *atom* z $\sharp$ y **using** *zb* **by** *auto*
show *atom* z $\sharp$ Γ' @ (x, *b0*, *c0*) $\#_\Gamma$ Γ **using** *fresh-replace-inside wfg0 wfg1 zb* **by** *metis*
qed

thus *?thesis*
using *subtype-reflI2 infer-v-t-wf*
by (*metis Pair-inject V-var.prems*(*1*) *bc infer-v-elims*(*1*) *option.inject type-eq-subst-eq2*(*2*) *zb*)
qed
next
case (*V-pair v1 v2*)

then obtain *tv1* **and** *tv2* **and** z **where** *tt1*: Θ ; $\mathcal{B}$; Γ' @ (x, *b0*, *c0'*) $\#_\Gamma$ Γ $\vdash$ *v1* $\Rightarrow$ *tv1* $\wedge$ Θ ; $\mathcal{B}$; Γ' @ (x, *b0*, *c0'*) $\#_\Gamma$ Γ $\vdash$ *v2* $\Rightarrow$ *tv2* $\wedge$ *t1* = ({| z : *B-pair* (*b-of tv1*) (*b-of tv2*) | *CE-val* (*V-var z*) == *CE-val* (*V-pair v1 v2*) |}) $\wedge$ *atom* z $\sharp$ (*v1*,*v2*)
using *infer-v-pair2E* **by** *presburger*
obtain *tv1'* **where** *t1*: Θ ; $\mathcal{B}$; Γ' @ (x, *b0*, *c0*) $\#_\Gamma$ Γ $\vdash$ *v1* $\Rightarrow$ *tv1'* $\wedge$ Θ ; $\mathcal{B}$; Γ' @ (x, *b0*, *c0*) $\#_\Gamma$ Γ $\vdash$ *tv1'* $\lesssim$ *tv1* **using** *tt1 V-pair* **by** *fast*
moreover obtain *tv2'* **where** *t2*: Θ ; $\mathcal{B}$; Γ' @ (x, *b0*, *c0*) $\#_\Gamma$ Γ $\vdash$ *v2* $\Rightarrow$ *tv2'* $\wedge$ Θ ; $\mathcal{B}$; Γ' @ (x, *b0*, *c0*) $\#_\Gamma$ Γ $\vdash$ *tv2'* $\lesssim$ *tv2* **using** *tt1 V-pair* **by** *fast*

ultimately obtain t' **and** z' **where** *tt2*: Θ ; $\mathcal{B}$; Γ' @ $(x, b0, c0)$ $\#_\Gamma$ Γ $\vdash$ *V-pair v1 v2* $\Rightarrow$ $t' \wedge$
$t' = (\{\!|\ z' : \textit{B-pair}\ (\textit{b-of tv1'})\ (\textit{b-of tv2'})\ |\ \textit{CE-val}\ (\textit{V-var}\ z')\ ==\ \textit{CE-val}\ (\textit{V-pair v1 v2})\ |\!\}) \wedge \textit{atom}\ z' \sharp (v1, v2)$
using *infer-v-pair2I-zbc t1 t2* **by** *metis*

hence *t1* = t' **proof** −
have $t' = (\{\!|\ z' : \textit{B-pair}\ (\textit{b-of tv1'})\ (\textit{b-of tv2'})\ |\ \textit{CE-val}\ (\textit{V-var}\ z')\ ==\ \textit{CE-val}\ (\textit{V-pair v1 v2})\ |\!\})$ **using** *tt2* **by** *auto*
moreover have *t1* = $(\{\!|\ z : \textit{B-pair}\ (\textit{b-of tv1})\ (\textit{b-of tv2})\ |\ \textit{CE-val}\ (\textit{V-var}\ z)\ ==\ \textit{CE-val}\ (\textit{V-pair v1 v2})\ |\!\})$ **using** *tt1* **by** *auto*
moreover have *b-of tv1* = *b-of tv1'* $\wedge$ *b-of tv2* = *b-of tv2'*
using *t1 t2* **by** (*metis subtype-eq-base2*)
moreover have *atom z* $\sharp$ *CE-val* (*V-pair v1 v2*) $\wedge$ *atom z'* $\sharp$ *CE-val* (*V-pair v1 v2*) **using** *tt1 tt2 ce.fresh v.fresh* **by** *force*
ultimately show *?thesis* **using** *type-e-eq* **by** *presburger*
qed

moreover have *wfT* Θ $\mathcal{B}$ (Γ' @ $(x, b0, c0)\#_\Gamma\Gamma$) t' **using** *t1 infer-v-t-wf tt2* **by** *metis*
ultimately have Θ ; $\mathcal{B}$; Γ' @ $(x, b0, c0)\#_\Gamma\Gamma \vdash t' \lesssim t1$ **using** *subtype-reflI*
using *subtype-reflI2* **by** *blast*

then show *?case* **using** *tt2* **by** *meson*

next
case (*V-consp s dc b' v'*)

obtain *z*::*x* **where** *zf*: *atom z* $\sharp$ (Θ, $\mathcal{B}$, Γ' @ $(x, b0, c0)$ $\#_\Gamma$ Γ, *v'*, *b'*, *V-consp s dc b' v'*) **using** *obtain-fresh* **by** *metis*

from *V-consp(2) V-consp(1) V-consp(3) zf* **have** *t2*:Θ ; $\mathcal{B}$; Γ' @ $(x, b0, c0)$ $\#_\Gamma$ $\Gamma \vdash$ *V-consp s dc b' v'* $\Rightarrow$ $\{\!|\ z : \textit{B-app}\ s\ b'\ |\ [\,[\,z\,]^v\,]^{ce}\ ==\ [\,\textit{V-consp}\ s\ dc\ b'\ v'\,]^{ce}\ |\!\}$
proof(*nominal-induct* Γ' @ $(x, b0, c0')$ $\#_\Gamma$ Γ *V-consp s dc b' v' t1 avoiding*: *c0 arbitrary*: *t1 rule*: *infer-v.strong-induct*)
case (*infer-v-conspI bv dclist* Θ *tc* $\mathcal{B}$ *tv zz*)
obtain *tv2* **where** $*$: Θ ; $\mathcal{B}$; Γ' @ $(x, b0, c0)$ $\#_\Gamma$ $\Gamma \vdash v' \Rightarrow tv2 \wedge \Theta$; $\mathcal{B}$; Γ' @ $(x, b0, c0)$ $\#_\Gamma$ $\Gamma \vdash tv2 \lesssim tv$
using *infer-v-conspI(17) infer-v-conspI* **by** *metis*
thm *ctx-subtype-subtype infer-v-conspI(18)*
show *?case* **proof**
show ‹*AF-typedef-poly s bv dclist* $\in$ *set* Θ› **using** *infer-v-conspI* **by** *auto*
show ‹(*dc*, *tc*) $\in$ *set dclist*› **using** *infer-v-conspI* **by** *auto*
show ‹ Θ ; $\mathcal{B}$ $\vdash_{wf}$ *b'* › **using** *infer-v-conspI* **by** *auto*
show *iv*: ‹ Θ ; $\mathcal{B}$; Γ' @ $(x, b0, c0)$ $\#_\Gamma$ $\Gamma \vdash v' \Rightarrow tv2$› **using** $*$ **by** *auto*

have Θ ; $\mathcal{B}$; Γ' @ $(x, b0, c0)$ $\#_\Gamma$ Γ $\vdash tv \lesssim tc[bv::=b']_{\tau b}$
using *infer-v-conspI infer-v-conspI(18) ctx-subtype-subtype* **by** *metis*

thus ‹Θ ; $\mathcal{B}$; Γ' @ $(x, b0, c0)$ $\#_\Gamma$ Γ $\vdash tv2 \lesssim tc[bv::=b']_{\tau b}$› **using** $*$ *subtype-trans* **by** *metis*

show ‹*atom z* $\sharp$ (Θ, $\mathcal{B}$, Γ' @ $(x, b0, c0)$ $\#_\Gamma$ Γ, *v'*, *b'*)› **using** *fresh-prodN infer-v-conspI* **by** *metis*

have *atom bv* $\sharp$ Γ' @ $(x, b0, c0)$ $\#_\Gamma$ Γ **unfolding** *fresh-append-g fresh-GCons fresh-prod3*

fresh-append-g
using *fresh-append-g fresh-GCons fresh-prod3 fresh-append-g* ‹ *atom bv* ♯ Γ' @ (x, *b0*, *c0'*) $\#_\Gamma$ Γ› *infer-v-conspI* **by** *metis*

thus ‹*atom bv* ♯ (Θ, $\mathcal{B}$, Γ' @ (x, *b0*, *c0*) $\#_\Gamma$ Γ, *v'*, *b'*)› **using** *infer-v-conspI fresh-prodN* **by** *metis*

qed
qed

let *?t2* = {| z : *B-app s b'* | [[z]^v]^{ce} == [*V-consp s dc b' v'*]^{ce} |}

obtain *z1* **and** *b1* **where** *t1*:*t1* = {| *z1* : *b1* | [[*z1*]^v]^{ce} == [*V-consp s dc b' v'*]^{ce} |} ∧ *atom z1* ♯ *V-consp s dc b' v'*
using *V-consp*(*2*) *infer-v-form* **by** *metis*
moreover then have *b1*:*b1* = *B-app s b'* **using** *infer-v-form-consp V-consp b-of.simps* **by** *metis*

let *?t1* = {| *z1* : *B-app s b'* | [[*z1*]^v]^{ce} == [*V-consp s dc b' v'*]^{ce} |}

have *?t1* = *?t2* **using** *type-e-eq zf t1 ce.fresh fresh-prodN* **by** *metis*

moreover have Θ ; $\mathcal{B}$; Γ' @ (x, *b0*, *c0*) $\#_\Gamma$ Γ $\vdash_{wf}$ {| z : *B-app s b'* | [[z]^v]^{ce} == [*V-consp s dc b' v'*]^{ce} |}
using *t2* **using** *infer-v-wf* **by** *auto*
ultimately have Θ ; $\mathcal{B}$; Γ' @ (x, *b0*, *c0*) $\#_\Gamma$ Γ ⊢ *?t2* $\lesssim$ *?t1* **using** *subtype-reflI* **by** *metis*

moreover have *?t1* = *t1* **using** *t1 b1* **by** *auto*
ultimately show *?case* **using** *t2* **by** *metis*

next
case (*V-cons s dc v'*)

obtain *xa* **and** *b* **and** *c* **and** *z'* **and** *c'* **and** *z* **and** *dclist* **where** *tt*:
t1 = ({| z : *B-id s* | *CE-val* (*V-var z*) == *CE-val* (*V-cons s dc v'*) |}) ∧
AF-typedef s dclist ∈ *set* Θ ∧
(*dc*, {| *xa* : *b* | *c* |}) ∈ *set dclist* ∧ *atom z* ♯ Γ' @ (x, *b0*, *c0'*) $\#_\Gamma$ Γ ∧
Θ ; $\mathcal{B}$; Γ' @ (x, *b0*, *c0'*) $\#_\Gamma$ Γ ⊢ *v'* ⇒ {| *z'* : *b* | *c'* |} ∧ Θ ; $\mathcal{B}$; Γ' @ (x, *b0*, *c0'*) $\#_\Gamma$ Γ ⊢ {| *z'* : *b* | *c'* |} $\lesssim$ {| *xa* : *b* | *c* |} ∧ *atom z* ♯ *v'*
using *infer-v-elims*(*4*)[*OF V-cons*(*2*)] **by** *metis*

hence Θ ; $\mathcal{B}$; Γ' @ (x, *b0*, *c0'*) $\#_\Gamma$ Γ ⊢ *v'* ⇒ {| *z'* : *b* | *c'* |} **by** *linarith*
then obtain *t2* **where** ∗: Θ ; $\mathcal{B}$; Γ' @ (x, *b0*, *c0*) $\#_\Gamma$ Γ ⊢ *v'* ⇒ *t2* ∧ Θ ; $\mathcal{B}$; Γ' @ ((x,*b0*,*c0*)$\#_\Gamma\Gamma$) ⊢ *t2* $\lesssim$ {| *z'* : *b* | *c'* |}
using *V-cons* **by** *presburger*
obtain *z3* **and** *b3* **and** *c3* **where** *t2*: *t2* = ({| *z3* : *b3* |*c3* |}) **using** *obtain-fresh-z* **by** *meson*
hence *beq*: *b* = *b3* **using** *subtype-eq-base* ∗ **by** *blast*

have Θ; $\mathcal{B}$; Γ' @ (x, *b0*, *c0*) $\#_\Gamma$ Γ ⊢ {| *z'* : *b* | *c'* |} $\lesssim$ {| *xa* : *b* | *c* |} **using** *tt ctx-subtype-subtype V-cons* **by** *metis*

hence *tsub*: Θ ; $\mathcal{B}$; Γ' @ (x, *b0*, *c0*) $\#_\Gamma$ Γ ⊢ *t2* $\lesssim$ {| *xa* : *b* | *c* |}
using *subtype-trans* ∗ **by** *blast*

have *wfTh* Θ **using** *tt infer-v-wf* **by** *auto*
moreover have *AF-typedef s dclist* $\in$ *set* $\Theta \wedge$ (*dc*, $\{\!|$ *xa* : *b* | *c* $|\!\}$) $\in$ *set dclist* **using** *tt* **by** *auto*
moreover have Θ ; $\mathcal{B}$; Γ' @ (*x*, *b0*, *c0*) $\#_\Gamma$ Γ $\vdash$ *v'* $\Rightarrow$ $\{\!|$ *z3* : *b* | *c3* $|\!\}$ **using** $*$ *t2 beq* **by** *blast*
moreover have Θ ; $\mathcal{B}$; Γ' @ (*x*, *b0*, *c0*) $\#_\Gamma$ Γ $\vdash$ $\{\!|$ *z3* : *b* | *c3* $|\!\}$ $\lesssim$ $\{\!|$ *xa* : *b* | *c* $|\!\}$ **using** *t2 tsub beq* **by** *blast*
moreover have *atom z* $\sharp$ *v'* **using** *tt* **by** *auto*
moreover have *atom z* $\sharp$ Γ' @ (*x*, *b0*, *c0*) $\#_\Gamma$ Γ **using** *fresh-replace-inside tt infer-v-wf* $*$ **by** *metis*
ultimately have Θ ; $\mathcal{B}$; Γ' @ (*x*, *b0*, *c0*) $\#_\Gamma$ Γ $\vdash$ *V-cons s dc v'* $\Rightarrow$
$\{\!|$ *z* : *B-id s* | *CE-val* (*V-var z*) == *CE-val* (*V-cons s dc v'*) $|\!\}$
using *infer-v-consI* **by** *metis*

hence $**$: Θ ; $\mathcal{B}$; Γ' @ (*x*, *b0*, *c0*) $\#_\Gamma$ Γ $\vdash$ *V-cons s dc v'* $\Rightarrow$ *t1*
using *tt* **by** *argo*

moreover have Θ ; $\mathcal{B}$; Γ' @ (*x*, *b0*, *c0*) $\#_\Gamma$ Γ $\vdash$ *t1* $\lesssim$ *t1* **proof** −
have *wfT* Θ $\mathcal{B}$ (Γ' @ (*x*, *b0*, *c0*) $\#_\Gamma$ Γ) *t1* **using** $**$ *infer-v-wf* **by** *metis*
thus *?thesis* **using** *subtype-reflI2* **by** *presburger*
qed
ultimately show *?case* **by** *metis*
qed

lemma *ctx-subtype-v-eq*:
fixes *v*::*v*
assumes
Θ ; $\mathcal{B}$; Γ'@((x,*b0*,*c0'*)$\#_\Gamma\Gamma$) $\vdash$ *v* $\Rightarrow$ *t1* **and**
Θ ; $\mathcal{B}$; Γ'@(x,*b0*,*c0*)$\#_\Gamma\Gamma$ $\models$ *c0'*
shows Θ ; $\mathcal{B}$; Γ'@((x,*b0*,*c0*)$\#_\Gamma\Gamma$) $\vdash$ *v* $\Rightarrow$ *t1*
proof −
obtain *t1'* **where** Θ ; $\mathcal{B}$; Γ'@((x,*b0*,*c0*)$\#_\Gamma\Gamma$) $\vdash$ *v* $\Rightarrow$ *t1'* **using** *ctx-subtype-v assms* **by** *metis*
moreover have *replace-in-g* (Γ'@((x,*b0*,*c0'*)$\#_\Gamma\Gamma$)) *x c0* = Γ'@((x,*b0*,*c0*)$\#_\Gamma\Gamma$) **using** *replace-in-g-inside infer-v-wf assms* **by** *metis*
ultimately show *?thesis* **using** *infer-v-uniqueness-rig assms* **by** *metis*
qed

lemma *ctx-subtype-check-v-eq*:
assumes Θ ; $\mathcal{B}$; Γ'@((x,*b0*,*c0'*)$\#_\Gamma\Gamma$) $\vdash$ *v* $\Leftarrow$ *t1* **and** Θ ; $\mathcal{B}$; Γ'@(x,*b0*,*c0*)$\#_\Gamma\Gamma$ $\models$ *c0'*
shows Θ ; $\mathcal{B}$; Γ'@((x,*b0*,*c0*)$\#_\Gamma\Gamma$) $\vdash$ *v* $\Leftarrow$ *t1*
proof −
obtain *t2* **where** *t2*: Θ ; $\mathcal{B}$; Γ'@((x,*b0*,*c0'*)$\#_\Gamma\Gamma$) $\vdash$ *v* $\Rightarrow$ *t2* $\wedge$ Θ ; $\mathcal{B}$; Γ'@((x,*b0*,*c0'*)$\#_\Gamma\Gamma$) $\vdash$ *t2* $\lesssim$ *t1*
using *check-v-elims assms* **by** *blast*
hence *t3*: Θ ; $\mathcal{B}$; Γ'@((x,*b0*,*c0*)$\#_\Gamma\Gamma$) $\vdash$ *v* $\Rightarrow$ *t2*
using *assms ctx-subtype-v-eq* **by** *blast*

have Θ ; $\mathcal{B}$; Γ'@((x,*b0*,*c0*)$\#_\Gamma\Gamma$) $\vdash$ *v* $\Rightarrow$ *t2* **using** *t3* **by** *auto*
moreover have Θ ; $\mathcal{B}$; Γ'@((x,*b0*,*c0*)$\#_\Gamma\Gamma$) $\vdash$ *t2* $\lesssim$ *t1* **proof** −

have Θ ; $\mathcal{B}$; Γ'@((x,*b0*,*c0'*)$\#_\Gamma\Gamma$) $\vdash$ *t2* $\lesssim$ *t1* **using** *t2* **by** *auto*
thus *?thesis* **using** *subtype-trans*
using *assms*(*2*) *ctx-subtype-subtype* **by** *blast*
qed

ultimately show *?thesis* **using** *check-v.intros* **by** *presburger*
qed

Basically the same as *ctx-subtype-v-eq* but in a different form

lemma *ctx-subtype-v-rig-eq*:
fixes $v::v$
assumes *replace-in-g-subtyped* $\Theta\ \mathcal{B}\ \Gamma'\ [(x,c0)]\ \Gamma$ **and**
$\Theta\ ;\ \mathcal{B}\ ;\ \Gamma'\ \vdash\ v \Rightarrow t1$
shows $\Theta\ ;\ \mathcal{B}\ ;\ \Gamma \vdash v \Rightarrow t1$
proof −
obtain b **and** $c0'$ **and** G **and** G' **where** $\Gamma' = G'\ @\ (x,b,c0')\#_\Gamma G \wedge\ \Gamma = G'\ @\ (x,b,c0)\#_\Gamma G \wedge\ \Theta\ ;\ \mathcal{B}\ ;\ G'@\ (x,b,c0)\#_\Gamma G\ \models c0'$
using *assms replace-in-g-inside-valid infer-v-wf* **by** *metis*
thus *?thesis* **using** *ctx-subtype-v-eq*[*of* $\Theta\ \mathcal{B}\ G'\ x\ b\ c0'\ G\ v\ t1\ c0$] *assms* **by** *simp*
qed

lemma *ctx-subtype-v-rigs-eq*:
fixes $v::v$
assumes *replace-in-g-subtyped* $\Theta\ \mathcal{B}\ \Gamma'\ xcs\ \Gamma$ **and**
$\Theta\ ;\ \mathcal{B}\ ;\ \Gamma'\ \vdash\ v \Rightarrow t1$
shows $\Theta\ ;\ \mathcal{B}\ ;\ \Gamma \vdash v \Rightarrow t1$
using *assms* **proof**(*induct xcs arbitrary*: $\Gamma\ \Gamma'\ t1$)
case *Nil*
then show *?case* **by** *auto*
next
case (*Cons a xcs*)
then obtain x **and** c **where** $a=(x,c)$ **by** *fastforce*

then obtain b **and** c' **where** *bc*: *Some* $(b,\ c') =$ *lookup* $\Gamma'\ x \wedge$
replace-in-g-subtyped $\Theta\ \mathcal{B}$ (*replace-in-g* $\Gamma'\ x\ c$) $xcs\ \Gamma \wedge\ \Theta\ ;\ \mathcal{B}\ ;\ \Gamma'\ \vdash_{wf}\ c \wedge$
$x \notin$ *fst* ‘ *set xcs* $\wedge\quad \Theta\ ;\ \mathcal{B}\ ;$ (*replace-in-g* $\Gamma'\ x\ c$) $\models c'$
using *replace-in-g-subtyped-elims*(*3*)[*of* $\Theta\ \mathcal{B}\ \Gamma'\ x\ c\ xcs\ \Gamma$] *Cons* **by** (*metis valid.simps*)

hence $*$: *replace-in-g-subtyped* $\Theta\ \mathcal{B}\ \Gamma'\ [(x,c)]$ (*replace-in-g* $\Gamma'\ x\ c$) **using** *replace-in-g-subtyped-consI*
by (*meson image-iff list.distinct*(*1*) *list.set-cases replace-in-g-subtyped-nilI*)
hence *t2*: $\Theta\ ;\ \mathcal{B}\ ;$ (*replace-in-g* $\Gamma'\ x\ c) \vdash v \Rightarrow t1$ **using** *ctx-subtype-v-rig-eq*[*OF* $*$ *Cons*(*3*)] **by** *blast*
moreover have $**$: *replace-in-g-subtyped* $\Theta\ \mathcal{B}$ (*replace-in-g* $\Gamma'\ x\ c$) $xcs\ \Gamma$ **using** *bc* **by** *auto*
ultimately have *t2'*: $\Theta\ ;\ \mathcal{B}\ ;\ \Gamma \vdash v \Rightarrow t1$ **using** *Cons* **by** *blast*
thus *?case* **by** *blast*
qed

lemma *ctx-subtype-check-v-rigs-eq*:
assumes *replace-in-g-subtyped* $\Theta\ \mathcal{B}\ \Gamma'\ xcs\ \Gamma$ **and**
$\Theta\ ;\ \mathcal{B}\ ;\ \Gamma'\ \vdash\ v \Leftarrow t1$
shows $\Theta\ ;\ \mathcal{B}\ ;\ \Gamma \vdash v \Leftarrow t1$
proof −
obtain *t2* **where** $\Theta\ ;\ \mathcal{B}\ ;\ \Gamma'\ \vdash v \Rightarrow t2 \wedge\ \Theta\ ;\ \mathcal{B}\ ;\ \Gamma' \vdash t2 \lesssim t1$ **using** *check-v-elims assms* **by** *fast*
hence $\Theta\ ;\ \mathcal{B}\ ;\ \Gamma\ \vdash v \Rightarrow t2 \wedge\ \Theta\ ;\ \mathcal{B}\ ;\ \Gamma \vdash t2 \lesssim t1$ **using** *ctx-subtype-v-rigs-eq ctx-subtype-subtype-rigs*

using *assms*(*1*) **by** *blast*
thus *?thesis*

using *check-v-subtypeI* **by** *blast*
qed

13.5 Expressions

lemma *valid-wfC*:
fixes *c0*::*c*
assumes Θ ; $\mathcal{B}$; $\Gamma'@(x,b0,c0)\#_\Gamma\Gamma \models c0'$
shows Θ ; $\mathcal{B}$; $(x,\ b0,\ TRUE)\ \#_\Gamma\ \Gamma\ \vdash_{wf}\ c0$
using *wfG-elim2 valid.simps wfG-suffix*
using *assms valid-g-wf* **by** *metis*

lemma *ctx-subtype-e-eq*:
fixes *G*::Γ
assumes
Θ ; Φ ; $\mathcal{B}$; *G* ; $\Delta \vdash e \Rightarrow t1$ **and** $G = \Gamma'@((x,b0,c0')\#_\Gamma\Gamma)$
Θ ; $\mathcal{B}$; $\Gamma'@(x,b0,c0)\#_\Gamma\Gamma \models c0'$
shows Θ ; Φ ; $\mathcal{B}$; $\Gamma'@((x,b0,c0)\#_\Gamma\Gamma)$; $\Delta \vdash e \Rightarrow t1$
using *assms* **proof**(*nominal-induct t1 avoiding*: *c0 rule*: *infer-e.strong-induct*)
case (*infer-e-valI* Θ $\mathcal{B}$ Γ'' Δ Φ *v* τ)
show *?case* **proof**
show ‹ Θ ; $\mathcal{B}$; Γ' @ $(x,\ b0,\ c0)$ $\#_\Gamma$ Γ $\vdash_{wf}$ Δ › **using** *wf-replace-inside2*(*6*) *valid-wfC infer-e-valI* **by** *auto*
show ‹ Θ $\vdash_{wf}$ Φ › **using** *infer-e-valI* **by** *auto*
show ‹ Θ ; $\mathcal{B}$; Γ' @ $(x,\ b0,\ c0)$ $\#_\Gamma$ Γ $\vdash v \Rightarrow \tau$› **using** *infer-e-valI ctx-subtype-v-eq* **by** *auto*
qed
next
case (*infer-e-plusI* Θ $\mathcal{B}$ Γ'' Δ Φ *v1 z1 c1 v2 z2 c2 z3*)
show *?case* **proof**
show ‹ Θ ; $\mathcal{B}$; Γ' @ $(x,\ b0,\ c0)$ $\#_\Gamma$ Γ $\vdash_{wf}$ Δ › **using** *wf-replace-inside2*(*6*) *valid-wfC infer-e-plusI* **by** *auto*
show ‹ Θ $\vdash_{wf}$ Φ › **using** *infer-e-plusI* **by** *auto*
show ∗:‹ Θ ; $\mathcal{B}$; Γ' @ $(x,\ b0,\ c0)$ $\#_\Gamma$ Γ $\vdash v1 \Rightarrow \{\!|\ z1 : B\text{-}int\ |\ c1\ |\!\}$› **using** *infer-e-plusI ctx-subtype-v-eq* **by** *auto*
show ‹ Θ ; $\mathcal{B}$; Γ' @ $(x,\ b0,\ c0)$ $\#_\Gamma$ Γ $\vdash v2 \Rightarrow \{\!|\ z2 : B\text{-}int\ |\ c2\ |\!\}$› **using** *infer-e-plusI ctx-subtype-v-eq* **by** *auto*
show ‹*atom z3* ♯ *AE-op Plus v1 v2*› **using** *infer-e-plusI* **by** *auto*
show ‹*atom z3* ♯ Γ' @ $(x,\ b0,\ c0)$ $\#_\Gamma$ Γ› **using** ∗ *infer-e-plusI fresh-replace-inside infer-v-wf* **by** *metis*
qed
next
case (*infer-e-leqI* Θ $\mathcal{B}$ Γ'' Δ Φ *v1 z1 c1 v2 z2 c2 z3*)
show *?case* **proof**
show ‹ Θ ; $\mathcal{B}$; Γ' @ $(x,\ b0,\ c0)$ $\#_\Gamma$ Γ $\vdash_{wf}$ Δ › **using** *wf-replace-inside2*(*6*) *valid-wfC infer-e-leqI* **by** *auto*
show ‹ Θ $\vdash_{wf}$ Φ › **using** *infer-e-leqI* **by** *auto*
show ∗:‹ Θ ; $\mathcal{B}$; Γ' @ $(x,\ b0,\ c0)$ $\#_\Gamma$ Γ $\vdash v1 \Rightarrow \{\!|\ z1 : B\text{-}int\ |\ c1\ |\!\}$› **using** *infer-e-leqI ctx-subtype-v-eq* **by** *auto*
show ‹ Θ ; $\mathcal{B}$; Γ' @ $(x,\ b0,\ c0)$ $\#_\Gamma$ Γ $\vdash v2 \Rightarrow \{\!|\ z2 : B\text{-}int\ |\ c2\ |\!\}$› **using** *infer-e-leqI ctx-subtype-v-eq* **by** *auto*
show ‹*atom z3* ♯ *AE-op LEq v1 v2*› **using** *infer-e-leqI* **by** *auto*
show ‹*atom z3* ♯ Γ' @ $(x,\ b0,\ c0)$ $\#_\Gamma$ Γ› **using** ∗ *infer-e-leqI fresh-replace-inside infer-v-wf* **by**

metis
qed
next
case (*infer-e-appI* Θ $\mathcal{B}$ Γ'' Δ Φ f x' b c τ' s' v τ)
show *?case* **proof**
show ‹ Θ ; $\mathcal{B}$; Γ' @ (x, *b0*, *c0*) $\#_\Gamma$ Γ $\vdash_{wf}$ Δ › **using** *wf-replace-inside2*(6) *valid-wfC infer-e-appI* **by** *auto*
show ‹ Θ $\vdash_{wf}$ Φ › **using** *infer-e-appI* **by** *auto*
show ‹*Some* (*AF-fundef f* (*AF-fun-typ-none* (*AF-fun-typ x' b c τ' s'*))) = *lookup-fun* Φ f› **using** *infer-e-appI* **by** *auto*
show ‹Θ ; $\mathcal{B}$; Γ' @ (x, *b0*, *c0*) $\#_\Gamma$ Γ $\vdash$ v ⇐ {| x' : b | c |}› **using** *infer-e-appI ctx-subtype-check-v-eq* **by** *auto*
thus ‹*atom x'* ♯ Γ' @ (x, *b0*, *c0*) $\#_\Gamma$ Γ› **using** *infer-e-appI fresh-replace-inside*[*of* Θ $\mathcal{B}$ Γ' *x b0 c0' Γ c0 x'*] *infer-v-wf* **by** *auto*
show ‹τ'[x'::=v]$_v$ = τ› **using** *infer-e-appI* **by** *auto*
qed
next
case (*infer-e-appPI* Θ $\mathcal{B}$ $\Gamma 1$ Δ Φ b' f *bv x1* b c τ' s' v τ)
show *?case* **proof**
show ‹ Θ ; $\mathcal{B}$; Γ' @ (x, *b0*, *c0*) $\#_\Gamma$ Γ $\vdash_{wf}$ Δ › **using** *wf-replace-inside2*(6) *valid-wfC infer-e-appPI* **by** *auto*
show ‹ Θ $\vdash_{wf}$ Φ › **using** *infer-e-appPI* **by** *auto*
show ‹ Θ ; $\mathcal{B}$ $\vdash_{wf}$ b' › **using** *infer-e-appPI* **by** *auto*
show ‹*Some* (*AF-fundef f* (*AF-fun-typ-some bv* (*AF-fun-typ x1 b c τ' s'*))) = *lookup-fun* Φ f› **using** *infer-e-appPI* **by** *auto*
show ‹Θ ; $\mathcal{B}$; Γ' @ (x, *b0*, *c0*) $\#_\Gamma$ Γ $\vdash$ v ⇐ {| *x1* : b[*bv*::=b']$_b$ | c[*bv*::=b']$_b$ |}› **using** *infer-e-appPI ctx-subtype-check-v-eq subst-defs* **by** *auto*
thus ‹*atom x1* ♯ Γ' @ (x, *b0*, *c0*) $\#_\Gamma$ Γ› **using** *fresh-replace-inside*[*of* Θ $\mathcal{B}$ Γ' *x b0 c0' Γ c0 x1*] *infer-v-wf infer-e-appPI* **by** *auto*
show ‹τ'[*bv*::=b']$_b$[*x1*::=v]$_v$ = τ› **using** *infer-e-appPI* **by** *auto*
have *atom bv* ♯ Γ' @ (x, *b0*, *c0'*) $\#_\Gamma$ Γ **using** *infer-e-appPI* **by** *metis*
hence *atom bv* ♯ Γ' @ (x, *b0*, *c0*) $\#_\Gamma$ Γ
unfolding *fresh-append-g fresh-GCons fresh-prod3* **using** ‹*atom bv* ♯ *c0* › *fresh-append-g* **by** *metis*
thus ‹*atom bv* ♯ (Θ, Φ, $\mathcal{B}$, Γ' @ (x, *b0*, *c0*) $\#_\Gamma$ Γ, Δ, b', v, τ)› **using** *infer-e-appPI* **by** *auto*
qed
next
case (*infer-e-fstI* Θ $\mathcal{B}$ Γ'' Δ Φ v z' *b1 b2* c z)
show *?case* **proof**
show ‹ Θ ; $\mathcal{B}$; Γ' @ (x, *b0*, *c0*) $\#_\Gamma$ Γ $\vdash_{wf}$ Δ › **using** *wf-replace-inside2*(6) *valid-wfC infer-e-fstI* **by** *auto*
show ‹ Θ $\vdash_{wf}$ Φ › **using** *infer-e-fstI* **by** *auto*
show ‹ Θ ; $\mathcal{B}$; Γ' @ (x, *b0*, *c0*) $\#_\Gamma$ Γ $\vdash$ v ⇒ {| z' : *B-pair b1 b2* | c |}› **using** *infer-e-fstI ctx-subtype-v-eq* **by** *auto*
thus ‹*atom z* ♯ Γ' @ (x, *b0*, *c0*) $\#_\Gamma$ Γ› **using** *infer-e-fstI fresh-replace-inside*[*of* Θ $\mathcal{B}$ Γ' *x b0 c0'* Γ *c0 z*] *infer-v-wf* **by** *auto*
show ‹*atom z* ♯ *AE-fst v*› **using** *infer-e-fstI* **by** *auto*
qed
next
case (*infer-e-sndI* Θ $\mathcal{B}$ Γ'' Δ Φ v z' *b1 b2* c z)
show *?case* **proof**
show ‹ Θ ; $\mathcal{B}$; Γ' @ (x, *b0*, *c0*) $\#_\Gamma$ Γ $\vdash_{wf}$ Δ › **using** *wf-replace-inside2*(6) *valid-wfC infer-e-sndI* **by** *auto*

show ‹ $\Theta \vdash_{wf} \Phi$ › **using** *infer-e-sndI* **by** *auto*
show ‹ Θ ; $\mathcal{B}$; Γ' @ (x, *b0*, *c0*) $\#_\Gamma$ Γ $\vdash$ v $\Rightarrow$ $\{\!|$ z' : *B-pair b1 b2* | c $|\!\}$› **using** *infer-e-sndI ctx-subtype-v-eq* **by** *auto*
thus ‹*atom z* ♯ Γ' @ (x, *b0*, *c0*) $\#_\Gamma$ Γ› **using** *infer-e-sndI fresh-replace-inside*[*of* Θ $\mathcal{B}$ Γ' *x b0 c0'* Γ *c0 z*] *infer-v-wf* **by** *auto*
show ‹*atom z* ♯ *AE-snd v*› **using** *infer-e-sndI* **by** *auto*
qed
next
case (*infer-e-lenI* Θ $\mathcal{B}$ Γ'' Δ Φ v z' c z)
show *?case* **proof**
show ‹ Θ ; $\mathcal{B}$; Γ' @ (x, *b0*, *c0*) $\#_\Gamma$ Γ $\vdash_{wf}$ Δ › **using** *wf-replace-inside2*(*6*) *valid-wfC infer-e-lenI* **by** *auto*
show ‹ $\Theta \vdash_{wf} \Phi$ › **using** *infer-e-lenI* **by** *auto*
show ‹ Θ ; $\mathcal{B}$; Γ' @ (x, *b0*, *c0*) $\#_\Gamma$ Γ $\vdash$ v $\Rightarrow$ $\{\!|$ z': *B-bitvec* | c $|\!\}$› **using** *infer-e-lenI ctx-subtype-v-eq* **by** *auto*
thus ‹*atom z* ♯ Γ' @ (x, *b0*, *c0*) $\#_\Gamma$ Γ› **using** *infer-e-lenI fresh-replace-inside*[*of* Θ $\mathcal{B}$ Γ' *x b0 c0'* Γ *c0 z*] *infer-v-wf* **by** *auto*
show ‹*atom z* ♯ *AE-len v*› **using** *infer-e-lenI* **by** *auto*
qed
next
case (*infer-e-mvarI* Θ $\mathcal{B}$ Γ'' Φ Δ u τ)
show *?case* **proof**
show Θ ; $\mathcal{B}$; Γ' @ (x, *b0*, *c0*) $\#_\Gamma$ Γ $\vdash_{wf}$ Δ **using** *wf-replace-inside2*(*6*) *valid-wfC infer-e-mvarI* **by** *auto*
thus Θ ; $\mathcal{B}$ $\vdash_{wf}$ Γ' @ (x, *b0*, *c0*) $\#_\Gamma$ Γ **using** *infer-e-mvarI fresh-replace-inside wfD-wf* **by** *blast*
show $\Theta \vdash_{wf} \Phi$ **using** *infer-e-mvarI* **by** *auto*
show (u, τ) $\in$ *setD* Δ **using** *infer-e-mvarI* **by** *auto*
qed
next
case (*infer-e-concatI* Θ $\mathcal{B}$ Γ'' Δ Φ *v1 z1 c1 v2 z2 c2 z3*)
show *?case* **proof**
show ‹ Θ ; $\mathcal{B}$; Γ' @ (x, *b0*, *c0*) $\#_\Gamma$ Γ $\vdash_{wf}$ Δ › **using** *wf-replace-inside2*(*6*) *valid-wfC infer-e-concatI* **by** *auto*
thus ‹*atom z3* ♯ Γ' @ (x, *b0*, *c0*) $\#_\Gamma$ Γ› **using** *infer-e-concatI fresh-replace-inside*[*of* Θ $\mathcal{B}$ Γ' *x b0 c0'* Γ *c0 z3*] *infer-v-wf wfX-wfY* **by** *metis*
show ‹ $\Theta \vdash_{wf} \Phi$ › **using** *infer-e-concatI* **by** *auto*
show ‹ Θ ; $\mathcal{B}$; Γ' @ (x, *b0*, *c0*) $\#_\Gamma$ Γ $\vdash$ *v1* $\Rightarrow$ $\{\!|$ *z1* : *B-bitvec* | *c1* $|\!\}$› **using** *infer-e-concatI ctx-subtype-v-eq* **by** *auto*
show ‹ Θ ; $\mathcal{B}$; Γ' @ (x, *b0*, *c0*) $\#_\Gamma$ Γ $\vdash$ *v2* $\Rightarrow$ $\{\!|$ *z2* : *B-bitvec* | *c2* $|\!\}$› **using** *infer-e-concatI ctx-subtype-v-eq* **by** *auto*
show ‹*atom z3* ♯ *AE-concat v1 v2*› **using** *infer-e-concatI* **by** *auto*
qed
next
case (*infer-e-splitI* Θ $\mathcal{B}$ Γ'' Δ Φ *v1 z1 c1 v2 z2 z3*)
show *?case* **proof**
show ∗:‹ Θ ; $\mathcal{B}$; Γ' @ (x, *b0*, *c0*) $\#_\Gamma$ Γ $\vdash_{wf}$ Δ › **using** *wf-replace-inside2*(*6*) *valid-wfC infer-e-splitI* **by** *auto*
show ‹ Θ $\vdash_{wf}$ Φ › **using** *infer-e-splitI* **by** *auto*
show ‹ Θ ; $\mathcal{B}$; Γ' @ (x, *b0*, *c0*) $\#_\Gamma$ Γ $\vdash$ *v1* $\Rightarrow$ $\{\!|$ *z1* : *B-bitvec* | *c1* $|\!\}$› **using** *infer-e-splitI ctx-subtype-v-eq* **by** *auto*
show ‹Θ ; $\mathcal{B}$; Γ' @
(x, *b0*, *c0*) $\#_\Gamma$

$\Gamma \vdash v2 \Leftarrow \{\!|\, z2 : B\text{-}int \mid [\, leq\, [\, [\, L\text{-}num\ 0\,]^v\,]^{ce}\, [\, [\, z2\,]^v\,]^{ce}\,]^{ce} == [\, [\, L\text{-}true\,]^v\,]^{ce}\ AND$
$[\, leq\, [\, [\, z2\,]^v\,]^{ce}\, [\!|\, [\, v1\,]^{ce}\, |\!]^{ce}\,]^{ce} == [\, [\, L\text{-}true\,]^v\,]^{ce}\ \ |\!|\}\rangle$
using *infer-e-splitI ctx-subtype-check-v-eq* **by** *auto*

show ‹*atom z1* ♯ Γ' @ $(x,\ b0,\ c0)$ $\#_\Gamma$ Γ› **using** *fresh-replace-inside*[*of* Θ $\mathcal{B}$ Γ' *x b0 c0'* Γ *c0 z1*] *infer-e-splitI infer-v-wf wfX-wfY* ∗ **by** *metis*
show ‹*atom z2* ♯ Γ' @ $(x,\ b0,\ c0)$ $\#_\Gamma$ Γ› **using** *fresh-replace-inside*[*of* Θ $\mathcal{B}$ Γ' *x b0 c0'* Γ *c0*] *infer-e-splitI infer-v-wf wfX-wfY* ∗ **by** *metis*
show ‹*atom z3* ♯ Γ' @ $(x,\ b0,\ c0)$ $\#_\Gamma$ Γ› **using** *fresh-replace-inside*[*of* Θ $\mathcal{B}$ Γ' *x b0 c0'* Γ *c0*] *infer-e-splitI infer-v-wf wfX-wfY* ∗ **by** *metis*
show ‹*atom z1* ♯ *AE-split v1 v2*› **using** *infer-e-splitI* **by** *auto*
show ‹*atom z2* ♯ *AE-split v1 v2*› **using** *infer-e-splitI* **by** *auto*
show ‹*atom z3* ♯ *AE-split v1 v2*› **using** *infer-e-splitI* **by** *auto*
qed
qed

lemma *ctx-subtype-e-rig-eq*:
assumes *replace-in-g-subtyped* Θ $\mathcal{B}$ Γ' $[(x,c0)]$ Γ **and**
Θ ; Φ ; $\mathcal{B}$; Γ' ; $\Delta \vdash e \Rightarrow t1$
shows Θ ; Φ ; $\mathcal{B}$; Γ ; $\Delta \vdash e \Rightarrow t1$
proof −
obtain *b* **and** *c0'* **and** *G* **and** *G'* **where** $\Gamma' = G'$ @ $(x,b,c0')\#_\Gamma G \wedge\ \Gamma = G'$ @ $(x,b,c0)\#_\Gamma G \wedge\ \Theta$; $\mathcal{B}$; G'@ $(x,b,c0)\#_\Gamma G\ \models c0'$
using *assms replace-in-g-inside-valid infer-e-wf* **by** *meson*
thus *?thesis*
using *assms ctx-subtype-e-eq* **by** *presburger*
qed

lemma *ctx-subtype-e-rigs-eq*:
assumes *replace-in-g-subtyped* Θ $\mathcal{B}$ Γ' *xcs* Γ **and**
Θ ; Φ ; $\mathcal{B}$; Γ'; $\Delta \vdash e \Rightarrow t1$
shows Θ ; Φ ; $\mathcal{B}$; Γ ; $\Delta \vdash e \Rightarrow t1$
using *assms* **proof**(*induct xcs arbitrary*: Γ Γ' *t1*)
case *Nil*
moreover have $\Gamma' = \Gamma$ **using** *replace-in-g-subtyped-nilI*
using *calculation*(*1*) **by** *blast*
moreover have $\Theta;\mathcal{B};\Gamma \vdash t1 \lesssim t1$ **using** *subtype-reflI2 Nil infer-e-t-wf* **by** *blast*
ultimately show *?case* **by** *blast*
next
case (*Cons a xcs*)

then obtain *x* **and** *c* **where** *a*=(*x*,*c*) **by** *fastforce*
then obtain *b* **and** *c'* **where** *bc*: *Some* (*b*, *c'*) = *lookup* Γ' *x* ∧
replace-in-g-subtyped Θ $\mathcal{B}$ (*replace-in-g* Γ' *x c*) *xcs* Γ ∧ Θ ; $\mathcal{B}$; Γ' $\vdash_{wf}$ *c* ∧
x ∉ *fst* ‘ *set xcs* ∧ Θ ; $\mathcal{B}$; (*replace-in-g* Γ' *x c*) $\models c'$ **using** *replace-in-g-subtyped-elims*(*3*)[*of* Θ $\mathcal{B}$ Γ' *x c xcs* Γ] *Cons*
by (*metis valid.simps*)

hence ∗: *replace-in-g-subtyped* Θ $\mathcal{B}$ Γ' $[(x,c)]$ (*replace-in-g* Γ' *x c*) **using** *replace-in-g-subtyped-consI*
by (*meson image-iff list.distinct*(*1*) *list.set-cases replace-in-g-subtyped-nilI*)
hence *t2*: Θ ; Φ ; $\mathcal{B}$; (*replace-in-g* Γ' *x c*) ; $\Delta \vdash e \Rightarrow t1$ **using** *ctx-subtype-e-rig-eq*[*OF* ∗ *Cons*(*3*)]

by *blast*
moreover have $**$: *replace-in-g-subtyped* $\Theta\ \mathcal{B}$ (*replace-in-g* $\Gamma'\ x\ c$) *xcs* Γ **using** *bc* **by** *auto*
ultimately have *t2'*: $\Theta\ ;\ \Phi\ ;\ \mathcal{B}\ ;\ \Gamma\ ;\ \Delta \vdash e \Rightarrow t1$ **using** *Cons* **by** *blast*
thus *?case* **by** *blast*
qed

13.6 Statements

lemma *ctx-subtype-s-rigs*:
fixes *c0*::*c* **and** *s*::*s* **and** *G'*::Γ **and** *xcs* :: (*x*∗*c*) *list* **and** *css*::*branch-list*
shows
check-s $\Theta\ \Phi\ \mathcal{B}\ G\ \Delta\ \ s\ \ t1 \Longrightarrow$ *wsX G xcs* $\Longrightarrow$ *replace-in-g-subtyped* $\Theta\ \mathcal{B}$ *G xcs G'* $\Longrightarrow$ *check-s* $\Theta\ \Phi\ \mathcal{B}\ G'\ \Delta\ \ s\ \ t1$ **and**
check-branch-s $\Theta\ \Phi\ \mathcal{B}\ G\ \Delta$ *tid cons const v cs t1* $\Longrightarrow$ *wsX G xcs* $\Longrightarrow$ *replace-in-g-subtyped* $\Theta\ \mathcal{B}$ *G xcs G'* $\Longrightarrow$ *check-branch-s* $\Theta\ \Phi\ \mathcal{B}\ G'\ \Delta$ *tid cons const v cs t1*
check-branch-list $\Theta\ \Phi\ \mathcal{B}\ G\ \Delta$ *tid dclist v css t1* $\Longrightarrow$ *wsX G xcs* $\Longrightarrow$ *replace-in-g-subtyped* $\Theta\ \mathcal{B}$ *G xcs G'* $\Longrightarrow$ *check-branch-list* $\Theta\ \Phi\ \mathcal{B}\ G'\ \Delta$ *tid dclist v css t1*
proof(*induction arbitrary*: *xcs G'* **and** *xcs G'* **and** *xcs G' rule*: *check-s-check-branch-s-check-branch-list.inducts*)
case (*check-valI* $\Theta\ \mathcal{B}\ \Gamma\ \Delta\ \Phi\ v\ \tau'\ \tau$)
hence ∗:$\Theta\ ;\ \mathcal{B}\ ;\ G' \vdash v \Rightarrow \tau' \wedge\ \Theta\ ;\ \mathcal{B}\ ;\ G' \vdash \tau' \lesssim \tau$ **using** *ctx-subtype-v-rigs-eq ctx-subtype-subtype-rigs*

by (*meson check-v.simps*)
show *?case* **proof**
show ‹$\Theta\ ;\ \mathcal{B}\ ;\ G' \vdash_{wf} \Delta$› **using** *check-valI wfD-rig* **by** *auto*
show ‹$\Theta \vdash_{wf} \Phi$ › **using** *check-valI* **by** *auto*
show ‹$\Theta\ ;\ \mathcal{B}\ ;\ G' \vdash v \Rightarrow \tau'$› **using** ∗ **by** *auto*
show ‹$\Theta\ ;\ \mathcal{B}\ ;\ G' \vdash \tau' \lesssim \tau$› **using** ∗ **by** *auto*
qed
next
case (*check-letI x* $\Theta\ \Phi\ \mathcal{B}\ \Gamma\ \Delta\ e\ \tau\ z'\ s\ b'\ c'$)
thm *replace-in-g-wfG*
show *?case* **proof**
have *wfG*: $\Theta\ ;\ \mathcal{B} \vdash_{wf} \Gamma \wedge \Theta\ ;\ \mathcal{B} \vdash_{wf} G'$ **using** *infer-e-wf check-letI replace-in-g-wfG* **using** *infer-e-wf*(*2*) **by** (*auto simp add*: *freshers*)
hence *atom x* ♯ *G'* **using** *check-letI replace-in-g-fresh replace-in-g-wfG* **by** *auto*
thus *atom x* ♯ (Θ, Φ, $\mathcal{B}$, *G'*, Δ, *e*, τ) **using** *check-letI* **by** *auto*
have *atom z'* ♯ *G'* **apply**(*rule replace-in-g-fresh*[*OF check-letI*(*7*)])
using *replace-in-g-wfG check-letI fresh-prodN infer-e-wf* **by** *metis*+
thus *atom z'* ♯ (*x*, Θ, Φ, $\mathcal{B}$, *G'*, Δ, *e*, τ, *s*) **using** *check-letI fresh-prodN* **by** *metis*

show $\Theta\ ;\ \Phi\ ;\ \mathcal{B}\ ;\ G'\ ;\ \Delta\ \vdash e \Rightarrow \{\!|\ z' : b'\ |\ c'\ |\!\}$
using *check-letI ctx-subtype-e-rigs-eq* **by** *blast*
show $\Theta\ ;\ \Phi\ ;\ \mathcal{B}\ ;\ (x,\ b',\ c'[z'::=V\text{-}var\ x]_v)\ \#_\Gamma\ G'\ ;\ \Delta\ \vdash s \Leftarrow \tau$
proof(*rule check-letI*(*5*))
have *vld*: Θ;$\mathcal{B}$;((x, b', $c'[z'::=V\text{-}var\ x]_v$) $\#_\Gamma$ Γ) $\models$ $c'[z'::=V\text{-}var\ x]_{cv}$ **proof** −
have *wfG* $\Theta\ \mathcal{B}$ ((x, b', $c'[z'::=V\text{-}var\ x]_v$) $\#_\Gamma$ Γ) **using** *check-letI check-s-wf* **by** *metis*
hence *wfC* $\Theta\ \mathcal{B}$ ((x, b', $c'[z'::=V\text{-}var\ x]_v$) $\#_\Gamma$ Γ) ($c'[z'::=V\text{-}var\ x]_{cv}$) **using** *wfC-refl subst-defs* **by** *auto*
thus *?thesis* **using** *valid-reflI*[*of* $\Theta\ \mathcal{B}\ x\ b'\ c'[z'::=V\text{-}var\ x]_v\ \Gamma\ \ c'[z'::=V\text{-}var\ x]_v$] *subst-defs* **by** *auto*
qed
have *xf*: $x \notin$ *fst* ‘ *set xcs* **proof** −

```
      have  atom ‘ fst ‘ set xcs ⊆ atom-dom Γ using check-letI wsX-iff by meson
      moreover have wfG Θ B Γ using infer-e-wf check-letI by metis
      ultimately show ?thesis using fresh-def  check-letI  wfG-dom-supp
        using wsX-fresh by auto
    qed
     show replace-in-g-subtyped Θ B ((x, b', c'[z'::=V-var x]_v) #_Γ Γ) ((x, c'[z'::=V-var x]_v) # xcs)
((x, b', c'[z'::=V-var x]_v) #_Γ G') proof −

      have Some (b', c'[z'::=V-var x]_v) =  lookup ((x, b', c'[z'::=V-var x]_v) #_Γ Γ) x by auto

       moreover have Θ ; B ; replace-in-g ((x, b', c'[z'::=V-var x]_v) #_Γ Γ) x  (c'[z'::=V-var x]_v) ⊨
c'[z'::=V-var x]_v proof −
       have replace-in-g ((x, b', c'[z'::=V-var x]_v) #_Γ Γ) x  (c'[z'::=V-var x]_v) = ((x, b', c'[z'::=V-var
x]_v) #_Γ Γ)
         using replace-in-g.simps by presburger
        thus ?thesis  using vld subst-defs by auto
       qed

        moreover have  replace-in-g-subtyped Θ B (replace-in-g ((x, b', c'[z'::=V-var x]_v) #_Γ Γ) x
(c'[z'::=V-var x]_v)) xcs ( ((x, b', c'[z'::=V-var x]_v) #_Γ G')) proof −
        have wfG Θ B ( ((x, b', c'[z'::=V-var x]_v) #_Γ Γ))  using check-letI check-s-wf  by metis
       hence replace-in-g-subtyped Θ B ( ((x, b', c'[z'::=V-var x]_v) #_Γ Γ)) xcs ( ((x, b', c'[z'::=V-var
x]_v) #_Γ G'))
         using check-letI replace-in-g-subtyped-cons xf by meson
       moreover have replace-in-g ((x, b', c'[z'::=V-var x]_v) #_Γ Γ) x (c'[z'::=V-var x]_v) = ( ((x, b',
c'[z'::=V-var x]_v) #_Γ Γ))
         using replace-in-g.simps by presburger
        ultimately show ?thesis by argo
       qed
        moreover  have Θ ; B ; (x, b', c'[z'::=V-var x]_v) #_Γ Γ  ⊢_wf  c'[z'::=V-var x]_v  using vld
subst-defs by auto
      ultimately show ?thesis using  replace-in-g-subtyped-consI xf replace-in-g.simps(2) by metis
    qed

    show  wsX ((x, b', c'[z'::=V-var x]_v) #_Γ Γ) ((x, c'[z'::=V-var x]_v) # xcs)
      using check-letI xf subst-defs  by (simp add: wsX-cons)
   qed
  qed

next
  case (check-branch-list-consI Θ Φ B Γ Δ tid dclist v cs τ css)
  then show ?case using Typing.check-branch-list-consI by auto
next
  case (check-branch-list-finalI Θ Φ B Γ Δ tid dclist v cs τ)
  then show ?case using Typing.check-branch-list-finalI by auto
next
   case (check-branch-s-branchI Θ B Γ Δ τ const x Φ tid cons v s)

  have wfcons: wfG Θ B ((x, b-of const, CE-val v  ==  CE-val (V-cons tid cons (V-var x))  AND c-of
const x) #_Γ Γ) using check-s-wf check-branch-s-branchI
    by meson
  hence wf: wfG Θ B Γ using  wfG-cons by metis
```

moreover have *atom x ♯ (const, G′,v)* **proof** −
have *atom x ♯ G′* **using** *check-branch-s-branchI wf replace-in-g-fresh wfG-dom-supp replace-in-g-wfG* **by** *simp*
thus *?thesis* **using** *check-branch-s-branchI fresh-prodN* **by** *simp*
qed

moreover have *st*: Θ ; Φ ; $\mathcal{B}$; (*x*, *b-of const*, *CE-val v == CE-val*(*V-cons tid cons* (*V-var x*)) *AND c-of const x*) $\#_\Gamma$ *G′* ; Δ ⊢ *s* ⇐ τ **proof** −
have *wsX* ((*x*, *b-of const*, *CE-val v == CE-val*(*V-cons tid cons* (*V-var x*)) *AND c-of const x*) $\#_\Gamma$ Γ) *xcs* **using** *check-branch-s-branchI wsX-cons2 wsX-fresh wf* **by** *force*
moreover have *replace-in-g-subtyped* Θ $\mathcal{B}$ ((*x*, *b-of const*, *CE-val v == CE-val* (*V-cons tid cons* (*V-var x*)) *AND c-of const x*) $\#_\Gamma$ Γ) *xcs* ((*x*, *b-of const*, *CE-val v == CE-val*(*V-cons tid cons* (*V-var x*)) *AND c-of const x*) $\#_\Gamma$ *G′*)
using *replace-in-g-subtyped-cons wsX-fresh wf check-branch-s-branchI wfcons* **by** *auto*
thus *?thesis* **using** *check-branch-s-branchI calculation* **by** *meson*
qed
moreover have *wft*: *wfT* Θ $\mathcal{B}$ *G′* τ **using**
check-branch-s-branchI ctx-subtype-subtype-rigs subtype-reflI2 subtype-wf **by** *metis*
moreover have *wfD* Θ $\mathcal{B}$ *G′* Δ **using** *check-branch-s-branchI wfD-rig* **by** *presburger*
ultimately show *?case* **using**
Typing.check-branch-s-branchI
using *check-branch-s-branchI.hyps* **by** *simp*

next
case (*check-ifI z* Θ Φ $\mathcal{B}$ Γ Δ *v s1 s2* τ)
hence *wf*:*wfG* Θ $\mathcal{B}$ Γ **using** *check-s-wf* **by** *presburger*
show *?case* **proof**(*rule check-s-check-branch-s-check-branch-list.check-ifI*)
show ‹*atom z ♯* (Θ, Φ, $\mathcal{B}$, *G′*, Δ, *v*, *s1*, *s2*, τ)› **using** *fresh-prodN replace-in-g-fresh1 wf check-ifI* **by** *auto*
show ‹Θ ; $\mathcal{B}$; *G′* ⊢ *v* ⇐ {| *z* : *B-bool* | *TRUE* |}› **using** *ctx-subtype-check-v-rigs-eq check-ifI* **by** *presburger*
show ‹ Θ ; Φ ; $\mathcal{B}$; *G′* ; Δ ⊢ *s1* ⇐ {| *z* : *b-of* τ | *CE-val v == CE-val* (*V-lit L-true*) *IMP c-of* τ *z* |}› **using** *check-ifI* **by** *auto*
show ‹ Θ ; Φ ; $\mathcal{B}$; *G′* ; Δ ⊢ *s2* ⇐ {| *z* : *b-of* τ | *CE-val v == CE-val* (*V-lit L-false*) *IMP c-of* τ *z* |}› **using** *check-ifI* **by** *auto*
qed
next

case (*check-let2I x P* Φ $\mathcal{B}$ *G* Δ *t s1* τ *s2*)
show *?case* **proof**
have *wfG P* $\mathcal{B}$ *G* **using** *check-let2I check-s-wf* **by** *metis*
show ∗: *P* ; Φ ; $\mathcal{B}$; *G′* ; Δ ⊢ *s1* ⇐*t* **using** *check-let2I* **by** *blast*
show *atom x ♯* (*P*, Φ, $\mathcal{B}$, *G′*, Δ, *t*, *s1*, τ) **proof** −
have *wfG P* $\mathcal{B}$ *G′* **using** *check-s-wf* ∗ **by** *blast*
hence *atom-dom G* = *atom-dom G′* **using** *check-let2I rigs-atom-dom-eq* **by** *presburger*
moreover have *atom x ♯ G* **using** *check-let2I* **by** *auto*
moreover have *wfG P* $\mathcal{B}$ *G* **using** *check-s-wf* ∗ *replace-in-g-wfG check-let2I* **by** *simp*
ultimately have *atom x ♯ G′* **using** *wfG-dom-supp fresh-def* ‹*wfG P* $\mathcal{B}$ *G′*› **by** *metis*
thus *?thesis* **using** *check-let2I* **by** *auto*
qed
show *P* ; Φ ; $\mathcal{B}$;(*x*, *b-of t*, *c-of t x*) $\#_\Gamma$ *G′* ; Δ ⊢ *s2* ⇐ τ **proof** −

have *wsX* $((x,\ b\text{-}of\ t,\ c\text{-}of\ t\ x)\ \#_\Gamma\ G)$ *xcs* **using** *check-let2I wsX-cons2 wsX-fresh* ‹*wfG P* $\mathcal{B}$ *G*› **by** *simp*
moreover have *replace-in-g-subtyped P* $\mathcal{B}$ $((x,\ b\text{-}of\ t,\ c\text{-}of\ t\ x)\ \#_\Gamma\ G)$ *xcs* $((x,\ b\text{-}of\ t,\ c\text{-}of\ t\ x)\ \#_\Gamma\ G')$ **proof**(*rule replace-in-g-subtyped-cons*)
show *replace-in-g-subtyped P* $\mathcal{B}$ *G xcs G'* **using** *check-let2I* **by** *auto*
have *atom x* $\sharp$ *G* **using** *check-let2I* **by** *auto*
moreover have *wfT P* $\mathcal{B}$ *G t* **using** *check-let2I check-s-wf* **by** *metis*

moreover have *atom x* $\sharp$ *t* **using** *check-let2I check-s-wf wfT-supp* **by** *auto*
ultimately show *wfG P* $\mathcal{B}$ $((x,\ b\text{-}of\ t,\ c\text{-}of\ t\ x)\ \#_\Gamma\ G)$ **using** *wfT-wf-cons b-of-c-of-eq*[*of x t*] **by** *auto*
show *x* $\notin$ *fst* ‘ *set xcs* **using** *check-let2I wsX-fresh* ‹*wfG P* $\mathcal{B}$ *G*› **by** *simp*
qed
ultimately show *?thesis* **using** *check-let2I* **by** *presburger*
qed
qed
next
case (*check-varI u* Θ Φ $\mathcal{B}$ Γ Δ τ' *v* τ *s*)
show *?case* **proof**
have *atom u* $\sharp$ *G'* **unfolding** *fresh-def*
apply(*rule u-not-in-g* , *rule replace-in-g-wfG*)
using *check-v-wf check-varI* **by** *simp+*
thus ‹*atom u* $\sharp$ $(\Theta,\ \Phi,\ \mathcal{B},\ G',\ \Delta,\ \tau',\ v,\ \tau)$› **unfolding** *fresh-prodN* **using** *check-varI* **by** *simp*
show ‹Θ ; $\mathcal{B}$; *G'* $\vdash$ *v* $\Leftarrow$ τ'› **using** *ctx-subtype-check-v-rigs-eq check-varI* **by** *auto*
show ‹ Θ ; Φ ; $\mathcal{B}$; *G'* ; $(u,\ \tau')$ $\#_\Delta$ Δ $\vdash$ *s* $\Leftarrow$ τ› **using** *check-varI* **by** *auto*
qed
next
case (*check-assignI P* Φ $\mathcal{B}$ *G* Δ *u* τ *v z* τ')
show *?case* **proof**
show ‹*P* $\vdash_{wf}$ Φ › **using** *check-assignI* **by** *auto*
show ‹*P* ; $\mathcal{B}$; *G'* $\vdash_{wf}$ Δ › **using** *check-assignI wfD-rig* **by** *auto*
show ‹$(u,\ \tau)$ $\in$ *setD* Δ› **using** *check-assignI* **by** *auto*
show ‹*P* ; $\mathcal{B}$; *G'* $\vdash$ *v* $\Leftarrow$ τ› **using** *ctx-subtype-check-v-rigs-eq check-assignI* **by** *auto*
show ‹*P* ; $\mathcal{B}$; *G'* $\vdash$ $\{\!|\ z : B\text{-}unit\ |\ TRUE\ |\!\}$ $\lesssim$ τ'› **using** *ctx-subtype-subtype-rigs check-assignI* **by** *auto*
qed
next
case (*check-whileI* Δ *G P s1 z s2* τ')
then show *?case* **using** *Typing.check-whileI*
by (*meson ctx-subtype-subtype-rigs*)
next
case (*check-seqI* Δ *G P s1 z s2* τ)
then show *?case*
using *check-s-check-branch-s-check-branch-list.check-seqI* **by** *blast*
next
case (*check-caseI* Θ Φ $\mathcal{B}$ Γ Δ *tid dclist v cs* τ *z*)
show *?case* **proof**
show Θ ; Φ ; $\mathcal{B}$; *G'* ; Δ ; *tid* ; *dclist* ; *v* $\vdash$ *cs* $\Leftarrow$ τ **using** *check-caseI ctx-subtype-check-v-rigs-eq* **by** *auto*
show *AF-typedef tid dclist* $\in$ *set* Θ **using** *check-caseI* **by** *auto*
show Θ ; $\mathcal{B}$; *G'* $\vdash$ *v* $\Leftarrow$ $\{\!|\ z : B\text{-}id\ tid\ |\ TRUE\ |\!\}$ **using** *check-caseI ctx-subtype-check-v-rigs-eq* **by** *auto*

show $\vdash_{wf} \Theta$ **using** *check-caseI* **by** *auto*
qed
next
case (*check-assertI* $x\ \Theta\ \Phi\ \mathcal{B}\ \Gamma\ \Delta\ c\ \tau\ s$)
show *?case* **proof**
have *wfG*: $\Theta\ ;\ \mathcal{B} \vdash_{wf} \Gamma \wedge \Theta\ ;\ \mathcal{B} \vdash_{wf} G'$ **using** *check-s-wf check-assertI replace-in-g-wfG wfX-wfY* **by** *metis*
hence *atom* $x \sharp G'$ **using** *check-assertI replace-in-g-fresh replace-in-g-wfG* **by** *auto*
thus ‹*atom* $x \sharp (\Theta, \Phi, \mathcal{B}, G', \Delta, c, \tau, s)$› **using** *check-assertI fresh-prodN* **by** *auto*
show ‹ $\Theta\ ;\ \Phi\ ;\ \mathcal{B}\ ;\ (x,\ B\text{-}bool,\ c) \#_\Gamma\ G'\ ;\ \Delta\ \vdash s \Leftarrow \tau$ › **proof**(*rule check-assertI*(5))
show *wsX* $((x,\ B\text{-}bool,\ c) \#_\Gamma \Gamma)$ *xcs* **using** *check-assertI wsX-cons3* **by** *simp*
show $\Theta\ ;\ \mathcal{B} \vdash (x,\ B\text{-}bool,\ c) \#_\Gamma \Gamma \langle\ xcs\ \rangle \rightsquigarrow (x,\ B\text{-}bool,\ c) \#_\Gamma G'$ **proof**(*rule replace-in-g-subtyped-cons*)
show ‹ $\Theta\ ;\ \mathcal{B}\ \vdash \Gamma \langle\ xcs\ \rangle \rightsquigarrow G'$› **using** *check-assertI* **by** *auto*
show ‹ $\Theta\ ;\ \mathcal{B}\ \vdash_{wf} (x,\ B\text{-}bool,\ c) \#_\Gamma \Gamma$ › **using** *check-assertI check-s-wf* **by** *metis*
thus ‹$x \notin$ *fst* ' *set xcs*› **using** *check-assertI wsX-fresh wfG-elims wfX-wfY* **by** *metis*
qed
qed
show ‹$\Theta\ ;\ \mathcal{B}\ ;\ G'\ \models c$ › **using** *check-assertI replace-in-g-valid* **by** *auto*
show ‹ $\Theta\ ;\ \mathcal{B}\ ;\ G' \vdash_{wf} \Delta$ › **using** *check-assertI wfD-rig* **by** *auto*
qed
qed

lemma *replace-in-g-subtyped-empty*:
assumes *wfG* $\Theta\ \mathcal{B}\ (\Gamma' @ (x,\ b,\ c[z::=V\text{-}var\ x]_{cv}) \#_\Gamma \Gamma)$
shows *replace-in-g-subtyped* $\Theta\ \mathcal{B}$ (*replace-in-g* $(\Gamma' @ (x,\ b,\ c[z::=V\text{-}var\ x]_{cv}) \#_\Gamma \Gamma)\ x\ (c'[z'::=V\text{-}var\ x]_{cv})$) [] $(\Gamma' @ (x,\ b,\ c'[z'::=V\text{-}var\ x]_{cv}) \#_\Gamma \Gamma)$
proof −
have *replace-in-g* $(\Gamma' @ (x,\ b,\ c[z::=V\text{-}var\ x]_{cv}) \#_\Gamma \Gamma)\ x\ (c'[z'::=V\text{-}var\ x]_{cv}) = (\Gamma' @ (x,\ b,\ c'[z'::=V\text{-}var\ x]_{cv}) \#_\Gamma \Gamma)$
using *assms* **proof**(*induct* Γ' *rule*: Γ-*induct*)
case *GNil*
then show *?case* **using** *replace-in-g.simps* **by** *auto*
next
case (*GCons x1 b1 c1* $\Gamma 1$)
have $x \notin$ *fst* ' *setG* $((x1,b1,c1)\#_\Gamma \Gamma 1)$ **using** *GCons wfG-inside-fresh atom-dom.simps setG.simps append-g.simps* **by** *fast*
hence $x1 \neq x$ **using** *assms wfG-inside-fresh GCons* **by** *force*
hence $((x1,b1,c1) \#_\Gamma (\Gamma 1 @ (x,\ b,\ c[z::=V\text{-}var\ x]_{cv}) \#_\Gamma \Gamma))[x \longmapsto c'[z'::=V\text{-}var\ x]_{cv}] = (x1,b1,c1) \#_\Gamma (\Gamma 1 @ (x,\ b,\ c'[z'::=V\text{-}var\ x]_{cv}) \#_\Gamma \Gamma)$
using *replace-in-g.simps GCons wfG-elims append-g.simps* **by** *metis*
thus *?case* **using** *append-g.simps* **by** *simp*
qed
thus *?thesis* **using** *replace-in-g-subtyped-nilI* **by** *presburger*
qed

lemma *ctx-subtype-s*:
fixes *s*::*s*
assumes $\Theta\ ;\ \Phi\ ;\ \mathcal{B}\ ;\ \Gamma' @ ((x,b,c[z::=V\text{-}var\ x]_{cv}) \#_\Gamma \Gamma)\ ;\ \Delta \vdash s \Leftarrow \tau$ **and**
$\Theta\ ;\ \mathcal{B}\ ;\ \Gamma \vdash \{\!|\ z' : b \mid c'\ |\!\} \lesssim \{\!|\ z : b \mid c\ |\!\}$ **and**
atom $x \sharp (z,z',c,c')$

shows Θ ; Φ ; $\mathcal{B}$; $\Gamma'@(x,b,c'[z'::=V\text{-}var\ x]_{cv})\#_\Gamma\Gamma$; $\Delta \vdash s \Leftarrow \tau$
proof −

have *wf*: *wfG* Θ $\mathcal{B}$ $(\Gamma'@((x,b,c[z::=V\text{-}var\ x]_{cv})\#_\Gamma\Gamma))$ **using** *check-s-wf assms* **by** *meson*
hence $*$:$x \notin$ *fst* ‘ *setG* Γ' **using** *wfG-inside-fresh* **by** *force*
have *wfG* Θ $\mathcal{B}$ $((x,b,c[z::=V\text{-}var\ x]_{cv})\#_\Gamma\Gamma)$ **using** *wf wfG-suffix* **by** *metis*
hence *xfg*: *atom* $x \sharp \Gamma$ **using** *wfG-elims* **by** *metis*
have $x \neq z'$ **using** *assms fresh-at-base fresh-prod4* **by** *metis*
hence *a2*: *atom* $x \sharp c'$ **using** *assms fresh-prod4* **by** *metis*

have *atom* $x \sharp (z', c', z, c, \Gamma)$ **proof** −
have $x \neq z$ **using** *assms* **using** *assms fresh-at-base fresh-prod4* **by** *metis*
hence *a1* : *atom* $x \sharp c$ **using** *assms subtype-wf subtype-wf assms wfT-fresh-c xfg* **by** *meson*
thus *?thesis* **using** *a1 a2* ‹*atom* $x \sharp (z,z',c,c')$› *fresh-prod4 fresh-Pair xfg* **by** *simp*
qed
hence *wc1*: Θ ; $\mathcal{B}$; $(x,\ b,\ c'[z'::=V\text{-}var\ x]_v)\ \#_\Gamma\ \Gamma\ \models c[z::=V\text{-}var\ x]_v$
using *subtype-valid assms fresh-prodN* **by** *metis*

have *vld*: Θ;$\mathcal{B}$; $(\Gamma'@(x,\ b,\ c'[z'::=V\text{-}var\ x]_{cv})\ \#_\Gamma\ \Gamma) \models c[z::=V\text{-}var\ x]_{cv}$ **proof** −

have *setG* $((x,\ b,\ c'[z'::=V\text{-}var\ x]_{cv})\ \#_\Gamma\ \Gamma) \subseteq$ *setG* $(\Gamma'@(x,\ b,\ c'[z'::=V\text{-}var\ x]_{cv})\ \#_\Gamma\ \Gamma)$ **by** *auto*
moreover have *wfG* Θ $\mathcal{B}$ $(\Gamma'@(x,\ b,\ c'[z'::=V\text{-}var\ x]_{cv})\ \#_\Gamma\ \Gamma)$ **proof** −
have $*$:*wfT* Θ $\mathcal{B}$ Γ $(\{\!|\ z' : b \mid c'\ |\!\})$ **using** *subtype-wf assms* **by** *meson*
moreover have *atom* $x \sharp (c',\Gamma)$ **using** *xfg a2* **by** *simp*
ultimately have *wfG* Θ $\mathcal{B}$ $((x,\ b,\ c'[z'::=V\text{-}var\ x]_{cv})\ \#_\Gamma\ \Gamma)$ **using** *wfT-wf-cons-flip freshers* **by** *blast*
thus *?thesis* **using** *wfG-replace-inside2 check-s-wf assms* **by** *metis*
qed
ultimately show *?thesis* **using** *wc1 valid-weakening subst-defs* **by** *metis*
qed
hence *wbc*: Θ ; $\mathcal{B}$; Γ' @ $(x,\ b,\ c'[z'::=V\text{-}var\ x]_{cv})\ \#_\Gamma\ \Gamma\ \vdash_{wf} c[z::=V\text{-}var\ x]_{cv}$ **using** *valid.simps* **by** *auto*
have *wbc1*: Θ ; $\mathcal{B}$; $(x,\ b,\ c'[z'::=V\text{-}var\ x]_{cv})\ \#_\Gamma\ \Gamma\ \vdash_{wf} c[z::=V\text{-}var\ x]_{cv}$ **using** *wc1 valid.simps subst-defs* **by** *auto*
have *wsX* $(\Gamma'@((x,b,c[z::=V\text{-}var\ x]_{cv})\#_\Gamma\Gamma))$ $[(x,\ c'[z'::=V\text{-}var\ x]_{cv})]$ **proof**
show *wsX* $(\Gamma'$ @ $(x,\ b,\ c[z::=V\text{-}var\ x]_{cv})\ \#_\Gamma\ \Gamma)$ [] **using** *wsX-NilI* **by** *auto*
show *atom* $x \in$ *atom-dom* $(\Gamma'$ @ $(x,\ b,\ c[z::=V\text{-}var\ x]_{cv})\ \#_\Gamma\ \Gamma)$ **by** *simp*
show $x \notin$ *fst* ‘ *set* [] **by** *auto*
qed
moreover have *replace-in-g-subtyped* Θ $\mathcal{B}$ $(\Gamma'@((x,b,c[z::=V\text{-}var\ x]_{cv})\#_\Gamma\Gamma))$ $[(x,\ c'[z'::=V\text{-}var\ x]_{cv})]$ $(\Gamma'@(x,b,c'[z'::=V\text{-}var\ x]_{cv})\#_\Gamma\Gamma)$ **proof**
show *Some* $(b,\ c[z::=V\text{-}var\ x]_{cv})$ = *lookup* $(\Gamma'$ @ $(x,\ b,\ c[z::=V\text{-}var\ x]_{cv})\ \#_\Gamma\ \Gamma)\ x$ **using** *lookup-inside*$*$ **by** *auto*
show Θ ; $\mathcal{B}$; *replace-in-g* $(\Gamma'$ @ $(x,\ b,\ c[z::=V\text{-}var\ x]_{cv})\ \#_\Gamma\ \Gamma)\ x\ (c'[z'::=V\text{-}var\ x]_{cv})\ \models c[z::=V\text{-}var\ x]_{cv}$ **using** *vld replace-in-g-split wf* **by** *metis*
show *replace-in-g-subtyped* Θ $\mathcal{B}$ (*replace-in-g* $(\Gamma'$ @ $(x,\ b,\ c[z::=V\text{-}var\ x]_{cv})\ \#_\Gamma\ \Gamma)\ x\ (c'[z'::=V\text{-}var\ x]_{cv}))$ [] $(\Gamma'$ @ $(x,\ b,\ c'[z'::=V\text{-}var\ x]_{cv})\ \#_\Gamma\ \Gamma)$
using *replace-in-g-subtyped-empty wf* **by** *presburger*
show $x \notin$ *fst* ‘ *set* [] **by** *auto*
show Θ ; $\mathcal{B}$; Γ' @ $(x,\ b,\ c[z::=V\text{-}var\ x]_{cv})\ \#_\Gamma\ \Gamma\ \vdash_{wf} c'[z'::=V\text{-}var\ x]_{cv}$
proof(*rule wf-weakening*)
show ‹ Θ ; $\mathcal{B}$; $(x,\ b,\ c[z::=V\text{-}var\ x]_{cv})\ \#_\Gamma\ \Gamma\ \vdash_{wf} c'[z'::=[\ x\ ]^v]_{cv}$ › **using** *wfC-cons-switch*[*OF*

wbc1] *wf-weakening*(*6*) *check-s-wf assms setG.simps* **by** *metis*
show ‹ Θ ; $\mathcal{B}$ $\vdash_{wf}$ Γ' @ (x, b, c[z::=[x]^v]$_{cv}$) $\#_\Gamma$ Γ › **using** *wfC-cons-switch*[*OF wbc1*] *wf-weakening*(*6*) *check-s-wf assms setG.simps* **by** *metis*
show ‹*setG* ((x, b, c[z::=*V-var* x]$_{cv}$) $\#_\Gamma$ Γ) $\subseteq$ *setG* (Γ' @ (x, b, c[z::=[x]^v]$_{cv}$) $\#_\Gamma$ Γ)› **using** *append-g.simps setG.simps* **by** *auto*
qed
qed
ultimately show *?thesis* **using** *ctx-subtype-s-rigs*(*1*)[*OF assms*(*1*)] **by** *presburger*
qed

end

Chapter 14

Immutable Variable Substitution Lemmas

Lemmas that show that types are preserved, in some way, under immutable variable substitution

14.1 Misc

lemma *subst-top-eq*:
$\{\!|\ z : b\ |\ TRUE\ |\!\} = \{\!|\ z : b\ |\ TRUE\ |\!\}[x::=v]_{\tau v}$
proof −
obtain $z'::x$ **and** c' **where** *zeq*: $\{\!|\ z : b\ |\ TRUE\ |\!\} = \{\!|\ z' : b\ |\ c'\ |\!\} \wedge atom\ z' \sharp (x,v)$ **using** *obtain-fresh-z2 b-of.simps* **by** *metis*
hence $\{\!|\ z' : b\ |\ TRUE\ |\!\}[x::=v]_{\tau v} = \{\!|\ z' : b\ |\ TRUE\ |\!\}$ **using** *subst-tv.simps subst-cv.simps* **by** *metis*
moreover have $c' = C\text{-}true$ **using** τ*.eq-iff Abs1-eq-iff*(*3*) *c.fresh flip-fresh-fresh* **by** (*metis zeq*)
ultimately show *?thesis* **using** *zeq* **by** *metis*
qed

lemma *wfD-subst*:
fixes $\tau_1::\tau$ **and** $v::v$ **and** $\Delta::\Delta$ **and** $\Theta::\Theta$ **and** $\Gamma::\Gamma$
assumes $\Theta\ ;\ \mathcal{B}\ ;\ \Gamma \vdash v \Rightarrow \tau_1$ **and** *wfD* $\Theta\ \ \mathcal{B}\ (\Gamma'@((x,b_1,c0[z0::=[x]^v]_{cv})\ \#_\Gamma\ \Gamma))\ \ \Delta$ **and** *b-of* $\tau_1=b_1$
shows $\Theta\ ;\ \mathcal{B}\ ;\ \Gamma'[x::=v]_{\Gamma v}\ @\ \Gamma\ \vdash_{wf}\ \Delta[x::=v]_{\Delta v}$
proof −
have $\Theta\ ;\ \mathcal{B}\ ;\ \Gamma \vdash_{wf} v : b_1$ **using** *infer-v-v-wf assms* **by** *auto*
moreover have $(\Gamma'@((x,b_1,c0[z0::=[x]^v]_{cv})\#_\Gamma\Gamma))[x::=v]_{\Gamma v} = \Gamma'[x::=v]_{\Gamma v}\ @\ \Gamma$ **using** *subst-g-inside wfD-wf assms* **by** *metis*
ultimately show *?thesis* **using** *wf-subst assms* **by** *metis*
qed

lemma *subst-v-c-of*:
assumes $atom\ xa \sharp (v,x)$
shows $c\text{-}of\ t[x::=v]_{\tau v}\ xa = (c\text{-}of\ t\ xa)[x::=v]_{cv}$
using *assms* **proof**(*nominal-induct t avoiding*: *x v xa rule*:τ*.strong-induct*)
case (*T-refined-type* $z'\ b'\ c'$)
then have $c\text{-}of\ \{\!|\ z' : b'\ |\ c'\ |\!\}[x::=v]_{\tau v}\ xa = c\text{-}of\ \{\!|\ z' : b'\ |\ c'[x::=v]_{cv}\ |\!\}\ xa$
using *subst-tv.simps fresh-Pair* **by** *metis*

also have ... = $c'[x::=v]_{cv}$ $[z'::=V\text{-}var\ xa]_{cv}$ **using** *c-of.simps T-refined-type* **by** *metis*
also have ... = c' $[z'::=V\text{-}var\ xa]_{cv}[x::=v]_{cv}$
using *subst-cv-commute-subst*[*of z' v x V-var xa c'*] *subst-v-c-def T-refined-type fresh-Pair fresh-at-base v.fresh fresh-x-neq* **by** *metis*
finally show *?case* **using** *c-of.simps T-refined-type* **by** *metis*
qed

14.2 Context

lemma *subst-lookup*:
assumes *Some* (*b*,*c*) = *lookup* (Γ'@((*x*,b_1,c_1)#$_\Gamma\Gamma$)) *y* **and** $x \neq y$ **and** *wfG* Θ $\mathcal{B}$ (Γ'@((*x*,b_1,c_1)#$_\Gamma\Gamma$))
shows $\exists$ *d*. *Some* (*b*,*d*) = *lookup* (($\Gamma'[x::=v]_{\Gamma v}$)@$\Gamma$) *y*
using *assms* **proof**(*induct* Γ' *rule*: Γ-*induct*)
case *GNil*
hence *Some* (*b*,*c*) = *lookup* Γ *y* **by** (*simp add*: *assms*(*1*))
then show *?case* **using** *subst-gv.simps* **by** *auto*
next
case (*GCons x1 b1 c1* Γ*1*)
show *?case* **proof**(*cases x1* = *x*)
case *True*
hence *atom x* $\sharp$ (Γ*1* @ (*x*, b_1, c_1) #$_\Gamma$ Γ) **using** *GCons wfG-elims*(*2*)
append-g.simps **by** *metis*
moreover have *atom x* $\in$ *atom-dom* (Γ*1* @ (*x*, b_1, c_1) #$_\Gamma$ Γ) **by** *simp*
ultimately show *?thesis*
using *forget-subst-gv not-GCons-self2 subst-gv.simps append-g.simps*
by (*metis GCons.prems*(*3*) *True wfG-cons-fresh2*)
next
case *False*
hence ((*x1*,*b1*,*c1*) #$_\Gamma$ Γ*1*)[*x*::=*v*]$_{\Gamma v}$ = (*x1*,*b1*,*c1*[*x*::=*v*]$_{cv}$)#$_\Gamma$$\Gamma$*1*[*x*::=*v*]$_{\Gamma v}$ **using** *subst-gv.simps* **by** *auto*
then show *?thesis* **proof**(*cases x1*=*y*)
case *True*
then show *?thesis* **using** *GCons* **using** *lookup.simps*
by (*metis* ‹((*x1*, *b1*, *c1*) #$_\Gamma$ Γ*1*)[*x*::=*v*]$_{\Gamma v}$ = (*x1*, *b1*, *c1*[*x*::=*v*]$_{cv}$) #$_\Gamma$ Γ*1*[*x*::=*v*]$_{\Gamma v}$› *append-g.simps fst-conv option.inject*)
next
case *False*
then show *?thesis* **using** *GCons* **using** *lookup.simps*
using ‹((*x1*, *b1*, *c1*) #$_\Gamma$ Γ*1*)[*x*::=*v*]$_{\Gamma v}$ = (*x1*, *b1*, *c1*[*x*::=*v*]$_{cv}$) #$_\Gamma$ Γ*1*[*x*::=*v*]$_{\Gamma v}$› *append-g.simps* Γ.*distinct* Γ.*inject wfG.simps wfG-elims* **by** *metis*

qed
qed
qed

14.3 Satisfiability

lemma *is-satis-g-i-upd2*:
assumes *eval-v i v s* **and** *is-satis* ((*i* (*x* $\mapsto$ *s*))) *c0* **and** *atom x* $\sharp$ *G* **and** *wfG* Θ $\mathcal{B}$ (*G3*@((*x*,*b*,*c0*)#$_\Gamma$ *G*)) **and** *wfV* Θ $\mathcal{B}$ *G v b* **and** *wfI* Θ (*G3*[*x*::=*v*]$_{\Gamma v}$@*G*) *i*
and *is-satis-g i* (*G3*[*x*::=*v*]$_{\Gamma v}$@*G*)

shows *is-satis-g* (*i* ($x \mapsto s$)) (*G3*@((*x*,*b*,*c0*)#$_\Gamma$ *G*))
using *assms* **proof**(*induct G3 rule*: Γ-*induct*)
case *GNil*
hence *is-satis-g* (*i*($x \mapsto s$)) *G* **using** *is-satis-g-i-upd* **by** *auto*
then show *?case* **using** *GNil* **using** *is-satis-g.simps append-g.simps* **by** *metis*
next
case (*GCons x′ b′ c′* Γ′)
hence $x \neq x'$ **using** *wfG-cons-append* **by** *metis*
hence *is-satis-g i* (((*x′*, *b′*, *c′*[*x*::=*v*]$_{cv}$) #$_\Gamma$ (Γ′[*x*::=*v*]$_{\Gamma v}$) @ *G*)) **using** *subst-gv.simps GCons* **by** *auto*
hence ∗:*is-satis i c′*[*x*::=*v*]$_{cv}$ ∧ *is-satis-g i* ((Γ′[*x*::=*v*]$_{\Gamma v}$) @ *G*) **using** *subst-gv.simps* **by** *auto*

have *is-satis-g* (*i*($x \mapsto s$)) ((*x′*, *b′*, *c′*) #$_\Gamma$ (Γ′@ (*x*, *b*, *c0*) #$_\Gamma$ *G*)) **proof**(*subst is-satis-g.simps*,*rule*)
show *is-satis* (*i*($x \mapsto s$)) *c′* **proof**(*subst subst-c-satis-full*[*symmetric*])
show ‹*eval-v i v s*› **using** *GCons* **by** *auto*
show ‹ Θ ; $\mathcal{B}$; ((*x′*, *b′*, *c′*) #$_\Gamma$Γ′)@(*x*, *b*, *c0*) #$_\Gamma$ *G* ⊢$_{wf}$ *c′* › **using** *GCons wfC-refl* **by** *auto*
show ‹*wfI* Θ ((((*x′*, *b′*, *c′*) #$_\Gamma$ Γ′)[*x*::=*v*]$_{\Gamma v}$) @ *G*) *i*› **using** *GCons* **by** *auto*
show ‹ Θ ; $\mathcal{B}$; *G* ⊢$_{wf}$ *v* : *b* › **using** *GCons* **by** *auto*
show ‹*is-satis i c′*[*x*::=*v*]$_{cv}$› **using** ∗ **by** *auto*
qed
show *is-satis-g* (*i*($x \mapsto s$)) (Γ′ @ (*x*, *b*, *c0*) #$_\Gamma$ *G*) **proof**(*rule GCons*(*1*))
show ‹*eval-v i v s*› **using** *GCons* **by** *auto*
show ‹*is-satis* (*i*($x \mapsto s$)) *c0*› **using** *GCons* **by** *metis*
show ‹*atom x* ♯ *G*› **using** *GCons* **by** *auto*
show ‹ Θ ; $\mathcal{B}$⊢$_{wf}$ Γ′ @ (*x*, *b*, *c0*) #$_\Gamma$ *G* › **using** *GCons wfG-elims append-g.simps* **by** *metis*
show ‹*is-satis-g i* (Γ′[*x*::=*v*]$_{\Gamma v}$ @ *G*)› **using** ∗ **by** *auto*
show *wfI* Θ (Γ′[*x*::=*v*]$_{\Gamma v}$ @ *G*) *i* **using** *GCons wfI-def subst-g-assoc-cons* ‹*x*≠*x′*› **by** *auto*
show Θ ; $\mathcal{B}$; *G* ⊢$_{wf}$ *v* : *b* **using** *GCons* **by** *auto*
qed
qed
moreover have ((*x′*, *b′*, *c′*) #$_\Gamma$ Γ′ @ (*x*, *b*, *c0*) #$_\Gamma$ *G*) = (((*x′*, *b′*, *c′*) #$_\Gamma$ Γ′) @ (*x*, *b*, *c0*) #$_\Gamma$ *G*)
by *auto*
ultimately show *?case* **using** *GCons* **by** *metis*
qed

lemma *is-satis-eq*:
assumes *wfI* Θ *G i* **and** *wfCE* Θ $\mathcal{B}$ *G e b*
shows *is-satis i* (*e* == *e*)
proof(*rule*)
obtain *s* **where** *eval-e i e s* **using** *eval-e-exist assms* **by** *metis*
thus *eval-c i* (*e* == *e*) *True* **using** *eval-c-eqI* **by** *metis*
qed

14.4 Validity

lemma *subst-self-valid*:
fixes *v*::*v*
assumes Θ ; $\mathcal{B}$; *G* ⊢ *v* ⇒ {| *z* : *b* | *c* |} **and** *atom z* ♯ *v*
shows Θ ; $\mathcal{B}$; *G* ⊨ *c*[*z*::=*v*]$_{cv}$
proof −
have *c*= (*CE-val* (*V-var z*) == *CE-val v*) **using** *infer-v-form2 assms* **by** *presburger*
hence *c*[*z*::=*v*]$_{cv}$ = (*CE-val* (*V-var z*) == *CE-val v*)[*z*::=*v*]$_{cv}$ **by** *auto*
also have ... = (((*CE-val* (*V-var z*))[*z*::=*v*]$_{cev}$) == ((*CE-val v*)[*z*::=*v*]$_{cev}$)) **by** *fastforce*

also have ... = $((CE\text{-}val\ v)\ ==\ ((CE\text{-}val\ v)[z::=v]_{cev}))$ **using** *subst-cev.simps subst-vv.simps* **by** *presburger*
also have ... = $(CE\text{-}val\ v\ ==\ CE\text{-}val\ v\ )$ **using** *infer-v-form subst-cev.simps assms forget-subst-vv* **by** *presburger*
finally have $*: c[z::=v]_{cv} = (CE\text{-}val\ v\ ==\ CE\text{-}val\ v\ )$ **by** *auto*

have $**: \Theta\ ;\ \mathcal{B}\ ;\ G \vdash_{wf} CE\text{-}val\ v : b$ **using** *wfCE-valI assms infer-v-v-wf b-of.simps* **by** *metis*

show *?thesis* **proof**(*rule validI*)
show $\Theta\ ;\ \mathcal{B}\ ;\ G \vdash_{wf} c[z::=v]_{cv}$ **proof** −
have $\Theta\ ;\ \mathcal{B}\ ;\ G \vdash_{wf} v : b$ **using** *infer-v-v-wf assms b-of.simps* **by** *metis*
moreover have $\Theta \vdash_{wf} ([]::\Phi) \ \wedge\ \Theta\ ;\ \mathcal{B}\ ;\ G \vdash_{wf} []_\Delta$ **using** *wfD-emptyI wfPhi-emptyI infer-v-wf assms* **by** *auto*
ultimately show *?thesis* **using** $*$ *wfCE-valI wfC-eqI* **by** *metis*
qed
show $\forall\, i.\ wfI\ \Theta\ G\ i \wedge is\text{-}satis\text{-}g\ i\ G \longrightarrow is\text{-}satis\ i\ c[z::=v]_{cv}$ **proof**(*rule,rule*)
fix *i*
assume ‹*wfI* Θ *G i* $\wedge$ *is-satis-g i G*›
thus ‹*is-satis i* $c[z::=v]_{cv}$› **using** $*$ $**$ *is-satis-eq* **by** *auto*
qed
qed
qed

lemma *subst-valid-simple*:
fixes *v*::*v*
assumes $\Theta\ ;\ \mathcal{B}\ ;\ G \vdash v \Rightarrow \{\!|\ z0 : b \mid c0\ |\!\}$ **and**
atom z0 $\sharp$ *c* **and** *atom z0* $\sharp$ *v*
$\Theta;\ \mathcal{B}\ ;\ (z0,b,c0)\#_\Gamma G \models c[z::=V\text{-}var\ z0]_{cv}$
shows $\Theta\ ;\ \mathcal{B}\ ;\ G \models c[z::=v]_{cv}$
proof −
have $\Theta\ ;\ \mathcal{B}\ ;\ G \models c0[z0::=v]_{cv}$ **using** *subst-self-valid assms* **by** *metis*
moreover have *atom z0* $\sharp$ *G* **using** *assms valid-wf-all* **by** *meson*
moreover have *wfV* Θ $\mathcal{B}$ *G v b* **using** *infer-v-v-wf assms b-of.simps* **by** *metis*
moreover have $(c[z::=V\text{-}var\ z0]_{cv})[z0::=v]_{cv} = c[z::=v]_{cv}$ **using** *subst-v-simple-commute assms subst-v-c-def* **by** *metis*
ultimately show *?thesis* **using** *valid-trans assms subst-defs* **by** *metis*
qed

lemma *wfI-subst1*:
assumes *wfI* Θ $(G'[x::=v]_{\Gamma v}$ @ $G)$ *i* **and** *wfG* Θ $\mathcal{B}$ $(G'$ @ $(x,\ b,\ c[z::=[x]^v]_{cv})\ \#_\Gamma\ G)$ **and** *eval-v i v sv* **and** *wfRCV* Θ *sv b*
shows *wfI* Θ $(G'$ @ $(x,\ b,\ c[z::=[x]^v]_{cv})\ \#_\Gamma\ G)\ (\ i(\ x \mapsto sv))$
proof −
{
fix *xa*::*x* **and** *ba*::*b* **and** *ca*::*c*
assume *as*: $(xa,ba,ca) \in setG\ ((G'$ @ $((x,\ b,\ c[z::=[x]^v]_{cv})\ \#_\Gamma\ G)))$
then have $\exists\, s.\ Some\ s = (i(x \mapsto sv))\ xa \wedge wfRCV\ \Theta\ s\ ba$
proof(*cases x=xa*)
case *True*
have $Some\ sv = (i(x \mapsto sv))\ x \wedge wfRCV\ \Theta\ sv\ b$ **using** *as assms wfI-def* **by** *auto*
moreover have *b=ba* **using** *assms as True wfG-member-unique* **by** *metis*
ultimately show *?thesis* **using** *True* **by** *auto*

next
case *False*

then obtain *ca'* **where** (*xa*, *ba*, *ca'*) ∈ *setG* (*G'*[*x*::=*v*]$_{\Gamma v}$ @ *G*) **using** *wfG-member-subst2 assms as* **by** *metis*
then obtain *s* **where** *Some s* = *i xa* ∧ *wfRCV* Θ *s ba* **using** *wfI-def assms False* **by** *blast*
thus *?thesis* **using** *False* **by** *auto*
qed
}
from *this* **show** *?thesis* **using** *wfI-def allI* **by** *blast*
qed

lemma *subst-valid*:
fixes *v*::*v* **and** *c'*::*c* **and** Γ ::Γ
assumes Θ ; $\mathcal{B}$; Γ ⊨ *c*[*z*::=*v*]$_{cv}$ **and** Θ ; $\mathcal{B}$; Γ⊢$_{wf}$ *v* : *b* **and**
Θ ; $\mathcal{B}$⊢$_{wf}$ Γ **and** *atom x* ♯ *c* **and** *atom x* ♯ Γ **and**
Θ ; $\mathcal{B}$⊢$_{wf}$ (Γ'@(*x*,*b*,*c*[*z*::=[*x*]^v]$_{cv}$) #$_\Gamma$ Γ) **and**
Θ ; $\mathcal{B}$; Γ'@(*x*,*b*, *c*[*z*::=[*x*]^v]$_{cv}$) #$_\Gamma$ Γ ⊨ *c'* (**is** Θ ; $\mathcal{B}$; *?G* ⊨ *c'*)
shows Θ ; $\mathcal{B}$; Γ'[*x*::=*v*]$_{\Gamma v}$@Γ ⊨ *c'*[*x*::=*v*]$_{cv}$
proof −
have ∗:*wfC* Θ $\mathcal{B}$ (Γ'@(*x*,*b*, *c*[*z*::=[*x*]^v]$_{cv}$) #$_\Gamma$ Γ) *c'* **using** *valid.simps assms* **by** *metis*
hence *wfC* Θ $\mathcal{B}$ (Γ'[*x*::=*v*]$_{\Gamma v}$ @ Γ) (*c'*[*x*::=*v*]$_{cv}$) **using** *wf-subst*(*2*)[*OF* ∗] *b-of.simps assms subst-g-inside wfC-wf* **by** *metis*
moreover have ∀ *i*. *wfI* Θ (Γ'[*x*::=*v*]$_{\Gamma v}$ @ Γ) *i* ∧ *is-satis-g i* (Γ'[*x*::=*v*]$_{\Gamma v}$ @ Γ) ⟶ *is-satis i* (*c'*[*x*::=*v*]$_{cv}$)

proof(*rule*,*rule*)
fix *i*
assume *as*: *wfI* Θ (Γ'[*x*::=*v*]$_{\Gamma v}$ @ Γ) *i* ∧ *is-satis-g i* (Γ'[*x*::=*v*]$_{\Gamma v}$ @ Γ)
thm *valid.simps*
hence *wfig*: *wfI* Θ Γ *i* **using** *wfI-suffix infer-v-wf assms* **by** *metis*
then obtain *s* **where** *s*:*eval-v i v s* **and** *b*:*wfRCV* Θ *s b* **using** *eval-v-exist infer-v-v-wf b-of.simps assms* **by** *metis*
thm *is-satis-g-i-upd2*
have *is1*: *is-satis-g* (*i*(*x* ↦ *s*)) (Γ' @ (*x*, *b*, *c*[*z*::=[*x*]^v]$_{cv}$) #$_\Gamma$ Γ) **proof**(*rule is-satis-g-i-upd2*)
show *is-satis* (*i*(*x* ↦ *s*)) (*c*[*z*::=[*x*]^v]$_{cv}$) **proof** −
have *is-satis i* (*c*[*z*::=*v*]$_{cv}$)
using *subst-valid-simple assms as valid.simps infer-v-wf assms*
is-satis-g-suffix wfI-suffix **by** *metis*
hence *is-satis i* ((*c*[*z*::=[*x*]^v]$_{cv}$)[*x*::=*v*]$_{cv}$) **using** *assms subst-v-simple-commute*[*of x c z v*] *subst-v-c-def* **by** *metis*
moreover have Θ ; $\mathcal{B}$; (*x*, *b*, *c*[*z*::=[*x*]^v]$_{cv}$) #$_\Gamma$ Γ ⊢$_{wf}$ *c*[*z*::=[*x*]^v]$_{cv}$ **using** *wfC-refl wfG-suffix assms* **by** *metis*
moreover have Θ ; $\mathcal{B}$; Γ⊢$_{wf}$ *v* : *b* **using** *assms infer-v-v-wf b-of.simps* **by** *metis*
ultimately show *?thesis* **using** *subst-c-satis*[*OF s* , *of* Θ $\mathcal{B}$ *x b* *c*[*z*::=[*x*]^v]$_{cv}$ Γ *c*[*z*::=[*x*]^v]$_{cv}$] *wfig* **by** *auto*
qed
show *atom x* ♯ Γ **using** *assms* **by** *metis*
show *wfG* Θ $\mathcal{B}$ (Γ' @ (*x*, *b*, *c*[*z*::=[*x*]^v]$_{cv}$) #$_\Gamma$ Γ) **using** *valid-wf-all assms* **by** *metis*
show Θ ; $\mathcal{B}$; Γ⊢$_{wf}$ *v* : *b* **using** *assms infer-v-v-wf* **by** *force*
show *i* ⟦ *v* ⟧ ~ *s* **using** *s* **by** *auto*
show Θ ; Γ'[*x*::=*v*]$_{\Gamma v}$ @ Γ ⊢ *i* **using** *as* **by** *auto*

show $i \models \Gamma'[x::=v]_{\Gamma v}$ @ Γ **using** *as* **by** *auto*
qed
hence *is-satis* ($i($ $x \mapsto s)$) c' **proof** −
have *wfI* Θ (Γ' @ (x, b, $c[z::=[x]^v]_{cv}$) $\#_\Gamma$ Γ) ($i($ $x \mapsto s$))
using *wfI-subst1*[*of* Θ Γ' x v Γ i $\mathcal{B}$ b c z s] *as b s assms* **by** *metis*
thus *?thesis* **using** *is1 valid.simps assms* **by** *presburger*
qed

thus *is-satis i* ($c'[x::=v]_{cv}$) **using** *subst-c-satis-full*[*OF s*] *valid.simps as infer-v-v-wf b-of.simps assms* **by** *metis*

qed
ultimately show *?thesis* **using** *valid.simps* **by** *auto*
qed

lemma *subst-valid-infer-v*:
fixes $v::v$ **and** $c'::c$
assumes Θ ; $\mathcal{B}$; $G \vdash v \Rightarrow \{\!|$ *z0* : b | *c0* $|\!\}$ **and** *atom* $x \sharp c$ **and** *atom* $x \sharp G$ **and** *wfG* Θ $\mathcal{B}$ (G'@(x,b,$c[z::=[x]^v]_{cv}$) $\#_\Gamma$ G) **and** *atom z0* $\sharp$ v
Θ;$\mathcal{B}$;($z0$,b,$c0$)$\#_\Gamma G \models c[z::=V\text{-}var\ z0]_{cv}$ **and** *atom z0* $\sharp$ c **and**
Θ;$\mathcal{B}$;G'@(x,b, $c[z::=[x]^v]_{cv}$) $\#_\Gamma$ $G \models c'$ (**is** Θ ; $\mathcal{B}$; *?G* $\models c'$)
shows Θ;$\mathcal{B}$;$G'[x::=v]_{\Gamma v}$@$G \models c'[x::=v]_{cv}$
proof −
have Θ ; $\mathcal{B}$; $G \models c[z::=v]_{cv}$
using *infer-v-wf subst-valid-simple valid.simps assms* **using** *subst-valid-simple assms valid.simps infer-v-wf assms*
is-satis-g-suffix wfI-suffix **by** *metis*
moreover have *wfV* Θ $\mathcal{B}$ G v b **and** *wfG* Θ $\mathcal{B}$ G
using *assms infer-v-wf b-of.simps* **apply** *metis* **using** *assms infer-v-wf* **by** *metis*
ultimately show *?thesis* **using** *assms subst-valid* **by** *metis*
qed

14.5 Subtyping

lemma *subst-subtype*:
fixes $v::v$
assumes Θ ; $\mathcal{B}$; $\Gamma \vdash v \Rightarrow (\{\!|z0{:}b|c0|\!\})$ **and**
Θ;$\mathcal{B}$;$\Gamma \vdash (\{\!|z0{:}b|c0|\!\}) \lesssim (\{\!|$ z : b | c $|\!\})$ **and**
Θ;$\mathcal{B}$;Γ'@($(x,b,c[z::=[x]^v]_{cv})\#_\Gamma\Gamma$) $\vdash (\{\!|$ *z1* : *b1* | *c1* $|\!\}) \lesssim (\{\!|$ *z2* : *b1* | *c2* $|\!\})$ (**is** Θ ; $\mathcal{B}$; *?G1* $\vdash$ *?t1* $\lesssim$ *?t2*) **and**
atom z $\sharp$ (x,v) $\wedge$ *atom z0* $\sharp$ (c,x,v,z,Γ) $\wedge$ *atom z1* $\sharp$ (x,v) $\wedge$ *atom z2* $\sharp$ (x,v) **and** *wsV* Θ $\mathcal{B}$ Γ v
shows Θ;$\mathcal{B}$;$\Gamma'[x::=v]_{\Gamma v}$@$\Gamma \vdash$ $\{\!|$ *z1* : *b1* | *c1* $|\!\}[x::=v]_{\tau v} \lesssim$ $\{\!|$ *z2* : *b1* | *c2* $|\!\}[x::=v]_{\tau v}$
proof −
have *z2*: *atom z2* $\sharp$ (x,v) **using** *assms* **by** *auto*
hence $x \neq$ *z2* **by** *auto*

obtain *xx*::x **where** *xxf*: *atom xx* $\sharp$ (x,*z1*, *c1*, *z2*, *c2*, Γ' @ (x, b, $c[z::=[x]^v]_{cv}$) $\#_\Gamma$ Γ, *c1*$[x::=v]_{cv}$, *c2*$[x::=v]_{cv}$, $\Gamma'[x::=v]_{\Gamma v}$ @ Γ,
(Θ , $\mathcal{B}$, $\Gamma'[x::=v]_{\Gamma v}$@$\Gamma$, *z1* , *c1*$[x::=v]_{cv}$, *z2* , *c2*$[x::=v]_{cv}$)) (**is** *atom xx* $\sharp$ *?tup*)
using *obtain-fresh* **by** *blast*
hence *xxf2*: *atom xx* $\sharp$ (*z1*, *c1*, *z2*, *c2*, Γ' @ (x, b, $c[z::=[x]^v]_{cv}$) $\#_\Gamma$ Γ) **using** *fresh-prod9 fresh-prod5* **by** *fast*

have *vd1*: $\Theta;\mathcal{B};((xx,\ b1,\ c1[z1::=V\text{-}var\ xx]_{cv})\ \#_\Gamma\ \Gamma')[x::=v]_{\Gamma v}\ @\ \Gamma\ \models\ (c2[z2::=V\text{-}var\ xx]_{cv})[x::=v]_{cv}$
proof(*rule subst-valid-infer-v*[*of* Θ - - - *z0 b c0* - *c*, **where** $z{=}z$])
show $\Theta\ ;\ \mathcal{B}\ ;\ \Gamma\ \vdash\ v\ \Rightarrow\ \{\!|\ z0 : b\ |\ c0\ |\!\}$ **using** *assms* **by** *auto*

show *xf*: *atom x* $\sharp\ \Gamma$ **using** *subtype-g-wf wfG-inside-fresh-suffix assms* **by** *metis*

show *atom x* $\sharp$ *c* **proof** −
have *wfT* $\Theta\ \mathcal{B}\ \Gamma\ (\{\!|\ z : b\ |\ c\ |\!\})$ **using** *subtype-wf*[*OF assms*(*2*)] **by** *auto*
moreover have $x \neq z$ **using** *assms*(*4*)
using *fresh-Pair not-self-fresh* **by** *blast*
ultimately show *?thesis* **using** *xf wfT-fresh-c assms* **by** *presburger*
qed

show $\Theta\ ;\ \mathcal{B}\vdash_{wf}\ ((xx,\ b1,\ c1[z1::=V\text{-}var\ xx]_{cv})\ \#_\Gamma\ \Gamma')\ @\ (x,\ b,\ c[z::=[x]^v]_{cv})\ \#_\Gamma\ \Gamma$
proof(*subst append-g.simps*,*rule wfG-consI*)
show ∗: ‹ $\Theta\ ;\ \mathcal{B}\vdash_{wf}\ \Gamma'\ @\ (x,\ b,\ c[z::=[x]^v]_{cv})\ \#_\Gamma\ \Gamma$ › **using** *subtype-g-wf assms* **by** *metis*
show ‹*atom xx* $\sharp\ \Gamma'\ @\ (x,\ b,\ c[z::=[x]^v]_{cv})\ \#_\Gamma\ \Gamma$› **using** *xxf fresh-prod9* **by** *metis*
show ‹ $\Theta\ ;\ \mathcal{B}\vdash_{wf}\ b1$ › **using** *subtype-elims*[*OF assms*(*3*)] *wfT-wfC wfC-wf wfG-cons* **by** *metis*
show $\Theta\ ;\ \mathcal{B}\ ;\ (xx,\ b1,\ TRUE)\ \#_\Gamma\ \Gamma'\ @\ (x,\ b,\ c[z::=[x]^v]_{cv})\ \#_\Gamma\ \Gamma\ \vdash_{wf}\ c1[z1::=V\text{-}var\ xx]_{cv}$
proof(*rule wfT-wfC*)
have $\{\!|\ z1 : b1\ |\ c1\ |\!\} = \{\!|\ xx : b1\ |\ c1[z1::=V\text{-}var\ xx]_{cv}\ |\!\}$ **using** *xxf fresh-prod9 type-eq-subst xxf2 fresh-prodN* **by** *metis*
thus $\Theta\ ;\ \mathcal{B}\ ;\ \Gamma'\ @\ (x,\ b,\ c[z::=[x]^v]_{cv})\ \#_\Gamma\ \Gamma\ \vdash_{wf}\ \{\!|\ xx : b1\ |\ c1[z1::=V\text{-}var\ xx]_{cv}\ |\!\}$ **using** *subtype-wfT*[*OF assms*(*3*)] **by** *metis*
show *atom xx* $\sharp\ \Gamma'\ @\ (x,\ b,\ c[z::=[x]^v]_{cv})\ \#_\Gamma\ \Gamma$ **using** *xxf fresh-prod9* **by** *metis*
qed
qed

show *atom z0* $\sharp$ *v* **using** *assms fresh-prod5* **by** *auto*
have $\Theta\ ;\ \mathcal{B}\ ;\ (z0,\ b,\ c0)\ \#_\Gamma\ \Gamma\ \models\ c[z::=V\text{-}var\ z0]_v$
apply(*rule obtain-fresh*[*of* (*z0*,*c0*, Γ, *c*, *z*)],*rule subtype-valid*[*OF assms*(*2*), *THEN valid-flip*],
(*fastforce simp add*: *assms fresh-prodN*)+) **done**
thus $\Theta\ ;\ \mathcal{B}\ ;\ (z0,\ b,\ c0)\ \#_\Gamma\ \Gamma\ \models\ c[z::=V\text{-}var\ z0]_{cv}$ **using** *subst-defs* **by** *auto*

show *atom z0* $\sharp$ *c* **using** *assms fresh-prod5* **by** *auto*
show $\Theta\ ;\ \mathcal{B}\ ;\ ((xx,\ b1,\ c1[z1::=V\text{-}var\ xx]_{cv})\ \#_\Gamma\ \Gamma')\ @\ (x,\ b,\ c[z::=[x]^v]_{cv})\ \#_\Gamma\ \Gamma\ \models\ c2[z2::=V\text{-}var\ xx]_{cv}$
using *subtype-valid assms*(*3*) *xxf xxf2 fresh-prodN append-g.simps subst-defs* **by** *metis*
qed

have *xfw1*: *atom z1* $\sharp\ v \wedge$ *atom x* $\sharp\ [\ xx\ ]^v \wedge x \neq z1$
apply(*intro conjI*)
apply(*simp add*: *assms xxf fresh-at-base fresh-prodN freshers fresh-x-neq*)+
using *fresh-x-neq fresh-prodN xxf* **apply** *blast*
using *fresh-x-neq fresh-prodN assms* **by** *blast*

have *xfw2*: *atom z2* $\sharp\ v \wedge$ *atom x* $\sharp\ [\ xx\ ]^v \wedge x \neq z2$
apply(*auto simp add*: *assms xxf fresh-at-base fresh-prodN freshers*)
by(*insert xxf fresh-at-base fresh-prodN assms*, *fast*+)

have *wf1*: *wfT* $\Theta\ \mathcal{B}\ (\Gamma'[x::=v]_{\Gamma v}@\Gamma)\ (\{\!|\ z1 : b1\ |\ c1[x::=v]_{cv}\ |\!\})$ **proof** −

have *wfT* Θ $\mathcal{B}$ ($\Gamma'[x::=v]_{\Gamma v}$@$\Gamma$) ({| *z1* : *b1* | *c1* |})$[x::=v]_{\tau v}$
using *wf-subst*(*4*) *assms b-of.simps infer-v-v-wf subtype-wf subst-tv.simps subst-g-inside wfT-wf*
by *metis*
moreover have *atom z1* $\sharp$ (*x*,*v*) **using** *assms* **by** *auto*
ultimately show *?thesis* **using** *subst-tv.simps* **by** *auto*
qed
moreover have *wf2*: *wfT* Θ $\mathcal{B}$ ($\Gamma'[x::=v]_{\Gamma v}$@$\Gamma$) ({| *z2* : *b1* | *c2*$[x::=v]_{cv}$ |}) **proof** −
have *wfT* Θ $\mathcal{B}$ ($\Gamma'[x::=v]_{\Gamma v}$@$\Gamma$) ({| *z2* : *b1* | *c2* |})$[x::=v]_{\tau v}$ **using** *wf-subst*(*4*) *assms b-of.simps infer-v-v-wf subtype-wf subst-tv.simps subst-g-inside wfT-wf* **by** *metis*
moreover have *atom z2* $\sharp$ (*x*,*v*) **using** *assms* **by** *auto*
ultimately show *?thesis* **using** *subst-tv.simps* **by** *auto*
qed
moreover have Θ ; $\mathcal{B}$; (*xx*, *b1*, *c1*$[x::=v]_{cv}$[*z1*::=*V-var xx*]$_{cv}$) #$_\Gamma$ ($\Gamma'[x::=v]_{\Gamma v}$ @ Γ) $\models$ (*c2*$[x::=v]_{cv}$)[*z2*::=*V-var xx*]$_{cv}$ **proof** −
have *xx* $\neq$ *x* **using** *xxf fresh-Pair fresh-at-base* **by** *fast*
hence ((*xx*, *b1*, *subst-cv c1 z1* (*V-var xx*)) #$_\Gamma$ Γ')$[x::=v]_{\Gamma v}$ = (*xx*, *b1*, (*subst-cv c1 z1* (*V-var xx*))$[x::=v]_{cv}$) #$_\Gamma$ ($\Gamma'[x::=v]_{\Gamma v}$)
using *subst-gv.simps* **by** *auto*
moreover have (*c1*[*z1*::=*V-var xx*]$_{cv}$)$[x::=v]_{cv}$ = (*c1*$[x::=v]_{cv}$) [*z1*::=*V-var xx*]$_{cv}$ **using** *subst-cv-commute-subst xfw1* **by** *metis*
moreover have *c2*[*z2*::=[*xx*]^v]$_{cv}$$[x::=v]_{cv}$ = (*c2*$[x::=v]_{cv}$)[*z2*::=*V-var xx*]$_{cv}$ **using** *subst-cv-commute-subst xfw2* **by** *metis*
ultimately show *?thesis* **using** *vd1 append-g.simps* **by** *metis*
qed
moreover have *atom xx* $\sharp$ (Θ , $\mathcal{B}$, $\Gamma'[x::=v]_{\Gamma v}$@$\Gamma$, *z1* , *c1*$[x::=v]_{cv}$, *z2* ,*c2*$[x::=v]_{cv}$)
using *xxf fresh-prodN* **by** *metis*
ultimately have Θ ; $\mathcal{B}$; $\Gamma'[x::=v]_{\Gamma v}$@$\Gamma$ ⊢ {| *z1* : *b1* | *c1*$[x::=v]_{cv}$ |} ≲ {| *z2* : *b1* | *c2*$[x::=v]_{cv}$ |}
using *subtype-baseI subst-defs* **by** *metis*
thus *?thesis* **using** *subst-tv.simps assms* **by** *presburger*
qed

lemma *subst-subtype-tau*:
fixes *v*::*v*
assumes Θ ; $\mathcal{B}$; Γ ⊢ *v* ⇒ τ **and**
Θ ; $\mathcal{B}$; Γ ⊢ τ ≲ ({| *z* : *b* | *c* |})
Θ ; $\mathcal{B}$; Γ'@((*x*,*b*,*c*[*z*::=[*x*]^v]$_{cv}$)#$_\Gamma\Gamma$) ⊢ *τ1* ≲ *τ2* **and**
atom z $\sharp$ (*x*,*v*)
shows Θ ; $\mathcal{B}$; $\Gamma'[x::=v]_{\Gamma v}$@$\Gamma$ ⊢ *τ1*$[x::=v]_{\tau v}$ ≲ *τ2*$[x::=v]_{\tau v}$
proof −
obtain *z0* **and** *b0* **and** *c0* **where** *zbc0*: τ=({| *z0* : *b0* | *c0* |}) ∧ *atom z0* $\sharp$ (*c*,*x*,*v*,*z*,Γ)
using *obtain-fresh-z* **by** *metis*
obtain *z1* **and** *b1* **and** *c1* **where** *zbc1*: *τ1*=({| *z1* : *b1* | *c1* |}) ∧ *atom z1* $\sharp$ (*x*,*v*)
using *obtain-fresh-z* **by** *metis*
obtain *z2* **and** *b2* **and** *c2* **where** *zbc2*: *τ2*=({| *z2* : *b2* | *c2* |}) ∧ *atom z2* $\sharp$ (*x*,*v*)
using *obtain-fresh-z* **by** *metis*

have *b0*=*b* **using** *subtype-eq-base zbc0 assms* **by** *blast*

hence *vinf*: Θ ; $\mathcal{B}$; Γ ⊢ *v* ⇒ {| *z0* : *b* | *c0* |} **using** *assms zbc0* **by** *blast*
have *vsub*: Θ ; $\mathcal{B}$; Γ ⊢{| *z0* : *b* | *c0* |} ≲ {| *z* : *b* | *c* |} **using** *assms zbc0* ‹*b0*=*b*› **by** *blast*
have *beq*:*b1*=*b2* **using** *subtype-eq-base*
using *zbc1 zbc2 assms* **by** *blast*

have Θ ; $\mathcal{B}$; $\Gamma'[x::=v]_{\Gamma v}$ @ Γ $\vdash$ $\{\!|$ *z1* : *b1* | *c1* $|\!\}[x::=v]_{\tau v} \lesssim \{\!|$ *z2* : *b1* | *c2* $|\!\}[x::=v]_{\tau v}$
proof(*rule subst-subtype*[*OF vinf vsub*])
show Θ ; $\mathcal{B}$; Γ'@((*x*,*b*,*c*[*z*::=[*x*]^v]$_{cv}$)#$_\Gamma\Gamma$) $\vdash$ $\{\!|$ *z1* : *b1* | *c1* $|\!\} \lesssim \{\!|$ *z2* : *b1* | *c2* $|\!\}$
using *beq assms zbc1 zbc2* **by** *auto*
show *atom z* $\sharp$ (*x*, *v*) $\wedge$ *atom z0* $\sharp$ (*c*, *x*, *v*, *z*, Γ) $\wedge$ *atom z1* $\sharp$ (*x*, *v*) $\wedge$ *atom z2* $\sharp$ (*x*, *v*)
using *zbc0 zbc1 zbc2 assms* **by** *blast*
show *wfV* Θ $\mathcal{B}$ Γ *v* (*b-of* τ) **using** *infer-v-wf assms* **by** *simp*
qed

thus *?thesis* **using** *zbc1 zbc2* ‹*b1*=*b2*› *assms* **by** *blast*
qed

lemma *subtype-if1*:
fixes *v*::*v*
assumes *P* ; $\mathcal{B}$; Γ $\vdash$ *t1* $\lesssim$ *t2* **and** *wfV P* $\mathcal{B}$ Γ *v* (*base-for-lit l*) **and**
atom z1 $\sharp$ *v* **and** *atom z2* $\sharp$ *v* **and** *atom z1* $\sharp$ *t1* **and** *atom z2* $\sharp$ *t2* **and** *atom z1* $\sharp$ Γ **and** *atom z2* $\sharp$ Γ
shows *P* ; $\mathcal{B}$; Γ $\vdash$ $\{\!|$ *z1* : *b-of t1* | *CE-val v* == *CE-val* (*V-lit l*) *IMP* (*c-of t1 z1*) $|\!\} \lesssim \{\!|$ *z2* : *b-of t2* | *CE-val v* == *CE-val* (*V-lit l*) *IMP* (*c-of t2 z2*) $|\!\}$
proof −
obtain *z1'* **where** *t1*: *t1* = $\{\!|$ *z1'* : *b-of t1* | *c-of t1 z1'* $|\!\}$ $\wedge$ *atom z1'* $\sharp$ (*z1*,Γ,*t1*) **using** *obtain-fresh-z-c-of* **by** *metis*
obtain *z2'* **where** *t2*: *t2* = $\{\!|$ *z2'* : *b-of t2* | *c-of t2 z2'* $|\!\}$ $\wedge$ *atom z2'* $\sharp$ (*z2*,*t2*) **using** *obtain-fresh-z-c-of* **by** *metis*
have *beq*:*b-of t1* = *b-of t2* **using** *subtype-eq-base2 assms* **by** *auto*

have *c1*: (*c-of t1 z1'*)[*z1'*::=[*z1*]^v]$_{cv}$ = *c-of t1 z1* **using** *c-of-switch t1 assms* **by** *simp*
have *c2*: (*c-of t2 z2'*)[*z2'*::=[*z2*]^v]$_{cv}$ = *c-of t2 z2* **using** *c-of-switch t2 assms* **by** *simp*

have *P* ; $\mathcal{B}$; Γ $\vdash$ $\{\!|$ *z1* : *b-of t1* | [*v*]^{ce} == [[*l*]^v]^{ce} *IMP* (*c-of t1 z1'*)[*z1'*::=[*z1*]^v]$_v$ $|\!\} \lesssim \{\!|$ *z2* : *b-of t1* | [*v*]^{ce} == [[*l*]^v]^{ce} *IMP* (*c-of t2 z2'*)[*z2'*::=[*z2*]^v]$_v$ $|\!\}$
proof(*rule subtype-if*)
show ‹*P* ; $\mathcal{B}$; Γ $\vdash$ $\{\!|$ *z1'* : *b-of t1* | *c-of t1 z1'* $|\!\} \lesssim \{\!|$ *z2'* : *b-of t1* | *c-of t2 z2'* $|\!\}$› **using** *t1 t2 assms beq* **by** *auto*
show ‹ *P* ; $\mathcal{B}$; Γ $\vdash_{wf}$ $\{\!|$ *z1* : *b-of t1* | [*v*]^{ce} == [[*l*]^v]^{ce} *IMP* (*c-of t1 z1'*)[*z1'*::=[*z1*]^v]$_v$ $|\!\}$ › **using** *wfT-wfT-if-rev assms subtype-wfT c1 subst-defs* **by** *metis*
show ‹ *P* ; $\mathcal{B}$; Γ $\vdash_{wf}$ $\{\!|$ *z2* : *b-of t1* | [*v*]^{ce} == [[*l*]^v]^{ce} *IMP* (*c-of t2 z2'*)[*z2'*::=[*z2*]^v]$_v$ $|\!\}$ › **using** *wfT-wfT-if-rev assms subtype-wfT c2 subst-defs beq* **by** *metis*
show ‹*atom z1* $\sharp$ *v*› **using** *assms* **by** *auto*
show ‹*atom z1'* $\sharp$ Γ› **using** *t1* **by** *auto*
show ‹*atom z1* $\sharp$ *c-of t1 z1'*› **using** *t1 assms c-of-fresh* **by** *force*
show ‹*atom z2* $\sharp$ *c-of t2 z2'*› **using** *t2 assms c-of-fresh* **by** *force*
show ‹*atom z2* $\sharp$ *v*› **using** *assms* **by** *auto*
qed
then show *?thesis* **using** *t1 t2 assms c1 c2 beq subst-defs* **by** *metis*
qed

14.6 Values

lemma *subst-infer-aux*:
fixes τ_1::τ **and** *v'*::*v*
assumes Θ ; $\mathcal{B}$; Γ $\vdash$ *v'*$[x::=v]_{vv}$ $\Rightarrow$ τ_1 **and** Θ ; $\mathcal{B}$; Γ' $\vdash$ *v'* $\Rightarrow$ τ_2 **and** *b-of* τ_1 = *b-of* τ_2

shows $\tau_1 = (\tau_2[x::=v]_{\tau v})$
proof −
obtain *z1* **and** *b1* **where** *zb1*: $\tau_1 = (\{\!|\ z1 : b1 \mid C\text{-}eq\ (CE\text{-}val\ (V\text{-}var\ z1))\ (CE\text{-}val\ (v'[x::=v]_{vv}))\ |\!\}) \wedge atom\ z1 \sharp ((CE\text{-}val\ (v'[x::=v]_{vv}),\ CE\text{-}val\ v),v'[x::=v]_{vv})$
using *infer-v-form-fresh*[*OF assms*(*1*)] **by** *fastforce*
obtain *z2* **and** *b2* **where** *zb2*: $\tau_2 = (\{\!|\ z2 : b2 \mid C\text{-}eq\ (CE\text{-}val\ (V\text{-}var\ z2))\ (CE\text{-}val\ v')\ |\!\}) \wedge atom\ z2 \sharp ((CE\text{-}val\ (v'[x::=v]_{vv}),\ CE\text{-}val\ v,x,v),v')$
using *infer-v-form-fresh* [*OF assms*(*2*)] **by** *fastforce*
have *beq*: *b1* = *b2* **using** *assms zb1 zb2* **by** *simp*

hence $(\{\!|\ z2 : b2 \mid C\text{-}eq\ (CE\text{-}val\ (V\text{-}var\ z2))\ (CE\text{-}val\ v')\ |\!\})[x::=v]_{\tau v} = (\{\!|\ z2 : b2 \mid C\text{-}eq\ (CE\text{-}val\ (V\text{-}var\ z2))\ (CE\text{-}val\ (v'[x::=v]_{vv}))\ |\!\})$
using *subst-tv.simps subst-cv.simps subst-ev.simps forget-subst-vv*[*of x V-var z2*] *zb2* **by** *force*
also have ... = $(\{\!|\ z1 : b1 \mid C\text{-}eq\ (CE\text{-}val\ (V\text{-}var\ z1))\ (CE\text{-}val\ (v'[x::=v]_{vv}))\ |\!\})$
using *type-e-eq*[*of z2 CE-val* $(v'[x::=v]_{vv})$*z1 b1*] *zb1 zb2 fresh-PairD*(*1*) *assms beq* **by** *metis*
finally show *?thesis* **using** *zb1 zb2* **by** *argo*
qed

lemma *subst-t-b-eq*:
fixes *x*::*x* **and** *v*::*v*
shows *b-of* $(\tau[x::=v]_{\tau v})$ = *b-of* τ
proof −
obtain *z* **and** *b* **and** *c* **where** $\tau = \{\!|\ z : b \mid c\ |\!\} \wedge atom\ z \sharp (x,v)$
using *has-fresh-z* **by** *blast*
thus *?thesis* **using** *subst-tv.simps* **by** *simp*
qed

lemma *fresh-g-fresh-v*:
fixes *x*::*x*
assumes *atom x* $\sharp\ \Gamma$ **and** *wfV* $\Theta\ \mathcal{B}\ \Gamma\ v\ b$
shows *atom x* $\sharp$ *v*
using *assms wfV-supp wfX-wfY wfG-atoms-supp-eq fresh-def*
by (*metis wfV-x-fresh*)

lemma *infer-v-fresh-g-fresh-v*:
fixes *x*::*x* **and** Γ::Γ **and** *v*::*v*
assumes *atom x* $\sharp\ \Gamma'@\Gamma$ **and** $\Theta\ ;\ \mathcal{B}\ ;\ \Gamma \vdash v \Rightarrow \tau$
shows *atom x* $\sharp$ *v*
proof −
have *atom x* $\sharp\ \Gamma$ **using** *fresh-suffix assms* **by** *auto*
moreover have *wfV* $\Theta\ \mathcal{B}\ \Gamma\ v$ (*b-of* τ) **using** *infer-v-wf assms* **by** *auto*
ultimately show *?thesis* **using** *fresh-g-fresh-v* **by** *metis*
qed

lemma *infer-v-fresh-g-fresh-xv*:
fixes *xa*::*x* **and** *v*::*v* **and** Γ::Γ
assumes *atom xa* $\sharp\ \Gamma'@((x,b,c[z::=[x]^v]_{cv})\#_\Gamma\Gamma)$ **and** $\Theta\ ;\ \mathcal{B}\ ;\ \Gamma \vdash v \Rightarrow \tau$
shows *atom xa* $\sharp\ (x,v)$
proof −
have *atom xa* $\sharp$ *x* **using** *assms fresh-in-g fresh-def* **by** *blast*
moreover have $\Gamma'@((x,b,c[z::=[x]^v]_{cv})\#_\Gamma\Gamma) = ((\Gamma'@(x,b,c[z::=[x]^v]_{cv})\#_\Gamma GNil)@\Gamma)$ **using** *append-g.simps append-g-assoc* **by** *simp*

moreover hence *atom xa* $\sharp$ *v* **using** *infer-v-fresh-g-fresh-v assms* **by** *metis*
ultimately show *?thesis* **by** *auto*
qed

lemma *wfG-subst-infer-v*:
fixes *v*::*v*
assumes Θ ; $\mathcal{B} \vdash_{wf} \Gamma'$ @ (*x*, *b*, *c0*[*z0*::=[*x*]^v]$_{cv}$) $\#_\Gamma$ Γ **and** Θ ; $\mathcal{B}$; $\Gamma \vdash v \Rightarrow \tau$ **and** *b-of* τ = *b*
shows Θ ; $\mathcal{B} \vdash_{wf} \Gamma'[x::=v]_{\Gamma v}$ @ Γ
using *wfG-subst-wfV infer-v-v-wf assms* **by** *auto*

lemma *fresh-subst-gv-inside*:
fixes Γ::Γ
assumes *atom z* $\sharp$ Γ' @ (*x*, *b*$_1$, *c0*[*z0*::=[*x*]^v]$_{cv}$) $\#_\Gamma$ Γ **and** *atom z* $\sharp$ *v*
shows *atom z* $\sharp$ $\Gamma'[x::=v]_{\Gamma v}$@$\Gamma$
unfolding *fresh-append-g* **using** *fresh-append-g assms fresh-subst-gv fresh-GCons* **by** *metis*

lemma *subst-infer-v*:
fixes *v*::*v* **and** *v'*::*v*
assumes Θ ; $\mathcal{B}$; $\Gamma \vdash v \Rightarrow \tau_1$ **and**
Θ ; $\mathcal{B}$; Γ'@((*x*,*b*$_1$,*c0*[*z0*::=[*x*]^v]$_{cv}$)$\#_\Gamma\Gamma$) $\vdash v' \Rightarrow \tau_2$ **and**
Θ ; $\mathcal{B}$; $\Gamma \vdash \tau_1 \lesssim$ ({| *z0* : *b*$_1$ | *c0* |}) **and** *atom z0* $\sharp$ (*x*,*v*)
shows Θ ; $\mathcal{B}$; ($\Gamma'[x::=v]_{\Gamma v}$)@$\Gamma$ $\vdash v'[x::=v]_{vv} \Rightarrow \tau_2[x::=v]_{\tau v}$
using *assms* **proof**(*nominal-induct v' avoiding*: *x v arbitrary*: τ_2 *rule*: *v.strong-induct*)
case (*V-lit l*)

hence $*$: $\vdash$ *l* $\Rightarrow$ τ_2 **using** *infer-v-elims* **by** *metis*
thm *type-eq-flip obtain-fresh-z type-e-eq*
then obtain *z b* **where** *t*: τ_2 = {| *z* : *b* | *CE-val* (*V-var z*) == *CE-val* (*V-lit l*) |} $\wedge$ *atom z* $\sharp$ Γ' @ (*x*, *b*$_1$, *c0*[*z0*::=[*x*]^v]$_{cv}$) $\#_\Gamma$ Γ
using *infer-l-form* $*$ **by** *metis*
hence $**$: $\tau_2[x::=v]_{\tau v}$ = τ_2 **proof** −
have *atom z* $\sharp$ (*x*,*v*) **using** *infer-v-fresh-g-fresh-xv*[*of z*] *V-lit infer-v-wf t* **by** *metis*
moreover have *atom x* $\sharp$ *V-lit l* **using** *v.fresh supp-l-empty fresh-def* **by** *fast*
ultimately show *?thesis* **using** *type-v-subst-fresh t* **by** *metis*
qed
have *b-of* τ_1 = *b*$_1$ **using** *subtype-eq-base2 V-lit b-of.simps* **by** *auto*

show *?case*
proof(*subst subst-vv.simps* , *rule infer-v-litI*)

show $\vdash$ *l* $\Rightarrow$ $\tau_2[x::=v]_{\tau v}$ **using** $*$ $**$ **by** *auto*
show Θ ; $\mathcal{B} \vdash_{wf} \Gamma'[x::=v]_{\Gamma v}$ @ Γ **using** *wfG-subst-infer-v V-lit* ‹*b-of* τ_1 = *b*$_1$› **by** *blast*
qed

next
case (*V-var y*)
have *b*$_1$ = *b-of* τ_1 **using** *subtype-eq-base2 assms b-of.simps* **by** *auto*
then obtain *z* **and** *b* **and** *c* **where** *zb*: τ_2 = {| *z* : *b* | *CE-val* (*V-var z*) == *CE-val* (*V-var y*) |} $\wedge$ *atom z* $\sharp$ *y* $\wedge$ *atom z* $\sharp$ (Γ' @ (*x*, *b*$_1$, *c0*[*z0*::=[*x*]^v]$_{cv}$) $\#_\Gamma$ Γ) $\wedge$ *Some* (*b*,*c*) = *lookup* (Γ' @ (*x*, *b*$_1$, *c0*[*z0*::=[*x*]^v]$_{cv}$) $\#_\Gamma$ Γ) *y*
proof −
assume $\bigwedge$*z b c*. τ_2 = {| *z* : *b* | *CE-val* (*V-var z*) == *CE-val* (*V-var y*) |} $\wedge$ *atom z* $\sharp$ *y* $\wedge$

atom z $\sharp$ (Γ' @ (x, b_1, $c0[z0::=[x]^v]_{cv}$) $\#_\Gamma$ Γ) $\wedge$ *Some* (b, c) = *lookup* (Γ' @ (x, b_1, $c0[z0::=[x]^v]_{cv}$) $\#_\Gamma$ Γ) y $\Longrightarrow$ *thesis*
then show *?thesis*
using *infer-var3*[*OF V-var*(*2*)] **by** *blast*
qed

If y is x then we are dealing with v otherwise substitution is identity

have *wfg1*: *wfG* Θ $\mathcal{B}$ (Γ'@(x,b_1,$c0[z0::=[x]^v]_{cv}$)$\#_\Gamma\Gamma$) **using** *infer-v-wf* **using** *V-var* **by** *fast*
moreover have *wfV* Θ $\mathcal{B}$ Γ v b_1 **using** *infer-v-v-wf V-var* ‹b_1 = *b-of* τ_1› **by** *auto*
ultimately have *wfg*: *wfG* Θ $\mathcal{B}$ (($\Gamma'[x::=v]_{\Gamma v}$)@$\Gamma$) **using** *wf-subst*(*3*)[*OF wfg1*] *subst-g-inside* **by** *metis*

have *wsg1*: *wfG* Θ $\mathcal{B}$ (Γ'@(x,b_1,$c0[z0::=[x]^v]_{cv}$)$\#_\Gamma\Gamma$) **using** *wfg1* **by** *auto*
hence *zf*:*atom z* $\sharp$ (($\Gamma'[x::=v]_{\Gamma v}$)@$\Gamma$) **using** *wfG-xa-fresh-in-subst-v V-var zb subst-g-inside wsg1 subst-defs* **by** *metis*

show *?case* **proof**(*cases* x = y)
case *True*
have *lu*: *Some* (b_1,$c0[z0::=[x]^v]_{cv}$) = *lookup* (Γ'@((x,b_1,$c0[z0::=[x]^v]_{cv}$))$\#_\Gamma\Gamma$) x **using** *lookup-inside-wf wfg1* **by** *metis*

moreover have (*V-var y*)$[x::=v]_{vv}$ = v **by** (*simp add*: *True*)

moreover have Θ ; $\mathcal{B}$; Γ $\vdash$ τ_1 $\lesssim$ (τ_2 $[x::=v]_{\tau v}$) **proof** −
have τ_1 = (τ_2 $[x::=v]_{\tau v}$) **using** *subst-infer-aux* [**where** x=x **and** v=v **and** v'=*V-var y* **and** τ_1=τ_1 **and** τ_2=τ_2,*OF - V-var*(*2*)]
by (*metis Pair-inject True V-var.prems*(*1*) *V-var.prems*(*2*) *assms*(*3*) *b-of.simps calculation*(*2*) *infer-v-elims*(*1*) *infer-v-form lu option.inject subst-infer-aux subtype-eq-base*)
thus *?thesis* **using** *subtype-reflI infer-v-t-wf*
using *assms subtype-reflI2* **by** *metis*
qed

ultimately have Θ ; $\mathcal{B}$; Γ $\vdash$ (*V-var y*)$[x::=v]_{vv}$ $\Rightarrow$ τ_1 $\wedge$ Θ ; $\mathcal{B}$; Γ $\vdash$ τ_1 $\lesssim$ $\tau_2[x::=v]_{\tau v}$
using *V-var True* **by** *argo*
moreover have *setG* Γ $\subseteq$ *setG* ($\Gamma'[x::=v]_{\Gamma v}$ @ Γ) **by** *simp*
moreover have Θ ; $\mathcal{B}\vdash_{wf}$ ($\Gamma'[x::=v]_{\Gamma v}$ @ Γ) **proof** −
have Θ ; $\mathcal{B}\vdash_{wf}$ Γ' @ (x, b_1, $c0[z0::=[x]^v]_{cv}$) $\#_\Gamma$ Γ **using** *infer-v-wf V-var* **by** *auto*
moreover have Θ ; $\mathcal{B}$; Γ $\vdash_{wf}$ v : *b-of* τ_1 **using** *infer-v-v-wf V-var* **by** *auto*
moreover have b_1 = *b-of* τ_1 **using** *subtype-eq-base2 assms b-of.simps* **by** *auto*
ultimately show *?thesis* **using** *wf-subst*(*3*) *subst-g-inside* **by** *metis*
qed
ultimately show *?thesis* **using** *infer-v-g-weakening subtype-weakening wfg append-g-assoc in-set-conv-decomp subset-code*
by (*metis V-var.prems*(*2*) *subst-infer-aux subst-t-b-eq subtype-eq-base2*)
next
case *False*

have (*V-var y*)$[x::=v]_{vv}$ = *V-var y* **by** (*simp add*: *False*)

have *eq* :(*V-var y*)$[x::=v]_{vv}$ = *V-var y* **by** (*simp add*: *False*)
then obtain c' **where** *Some* (b,c') = *lookup* ($\Gamma'[x::=v]_{\Gamma v}$@$\Gamma$) y **using** *subst-lookup*[*of b c* Γ' x b_1 $c0[z0::=[x]^v]_{cv}$ Γ y] *zb False wfg1 V-var* **by** *metis*

hence *a1*: Θ ; $\mathcal{B}$; $(\Gamma'[x::=v]_{\Gamma v}@\Gamma) \vdash$ (*V-var y*) $\Rightarrow$ ($\{\!|$ *z* : *b* | *CE-val* (*V-var z*) == *CE-val* (*V-var y*) $|\!\}$)
using *infer-v-varI*[*of* Θ - - *b c' y z* , *OF wfg*] *wfg zf zb* **by** *metis*
moreover have ($\{\!|$ *z* : *b* | *CE-val* (*V-var z*) == *CE-val* (*V-var y*) $|\!\}$) = $\tau_2[x::=v]_{\tau v}$ **proof** −
have *supp* τ_2 = { *atom y* } ∪ *supp b* **using** *zb supp-v-var-tau False* **by** *force*
hence *atom x* ♯ τ_2 **using** *False fresh-def* **by** *fastforce*
thus *?thesis* **using** *forget-subst-tv zb* **by** *metis*
qed
ultimately show *?thesis* **using** *subtype-reflI infer-v-t-wf eq subtype-reflI2* **by** *metis*
qed

next
case (*V-pair* v_1 v_2)

Unpack into typing for parts

then obtain τ'_1 **and** τ'_2 **and** *z* **where** *t1t2*: Θ ; $\mathcal{B}$; $\Gamma'@((x,b_1,c0[z0::=[x]^v]_{cv})\#_\Gamma\Gamma) \vdash v_1 \Rightarrow \tau'_1 \wedge$
Θ ; $\mathcal{B}$; $\Gamma'@((x,b_1,c0[z0::=[x]^v]_{cv})\#_\Gamma\Gamma) \vdash v_2 \Rightarrow \tau'_2 \wedge$
(τ_2 = ($\{\!|$ *z* : *B-pair* (*b-of* τ'_1) (*b-of* τ'_2) | ((*CE-val* (*V-var z*)) == (*CE-val* (*V-pair* v_1 v_2))) $|\!\}$))
using *infer-v-pair2E* **by** *meson*

Apply IH and repack to get required typing judgement

have *t1''*: Θ ; $\mathcal{B}$; $(\Gamma'[x::=v]_{\Gamma v}@\Gamma) \vdash v_1[x::=v]_{vv} \Rightarrow \tau'_1[x::=v]_{\tau v}$ **using** *V-pair t1t2* **by** *auto*
moreover have *t2''*:Θ ; $\mathcal{B}$; $(\Gamma'[x::=v]_{\Gamma v}@\Gamma) \vdash v_2[x::=v]_{vv} \Rightarrow \tau'_2[x::=v]_{\tau v}$ **using** *V-pair t1t2* **by** *auto*
ultimately obtain τ_3 **where** *t3*: Θ ; $\mathcal{B}$; $(\Gamma'[x::=v]_{\Gamma v}@\Gamma) \vdash$ *V-pair* $(v_1[x::=v]_{vv})$ $(v_2[x::=v]_{vv}) \Rightarrow \tau_3 \wedge$ (*b-of* τ_3 = *B-pair* (*b-of* τ'_1) (*b-of* τ'_2))
using *infer-v-pair2I subst-tbase-eq* **by** *metis*

Show required subtyping judgement

moreover have τ_3 = ($\tau_2[x::=v]_{\tau v}$) **proof** −
have *veq*: *V-pair* $(v_1[x::=v]_{vv})$ $(v_2[x::=v]_{vv})$ = (*V-pair* v_1 v_2)$[x::=v]_{vv}$ **using** *subst-vv.simps* **by** *presburger*
have Θ ; $\mathcal{B}$; $(\Gamma'[x::=v]_{\Gamma v}@\Gamma) \vdash$ (*V-pair* v_1 v_2)$[x::=v]_{vv} \Rightarrow \tau_3$ **using** *veq t3* **by** *simp*
moreover have Θ ; $\mathcal{B}$; $\Gamma'@((x,b_1,c0[z0::=[x]^v]_{cv})\#_\Gamma\Gamma) \vdash$ (*V-pair* v_1 v_2) $\Rightarrow \tau_2$ **using** *V-pair* **by** *simp*
moreover have *b-of* τ_3 = *b-of* τ_2 **proof** −
have *b-of* τ_3 = *B-pair* (*b-of* τ'_1) (*b-of* τ'_2) **using** *t3* **by** *auto*
moreover have *b-of* τ'_1 = *b-of* $\tau'_1[x::=v]_{\tau v}$ **using** *t1'' subst-tbase-eq*
by (*metis* τ.*exhaust b-of.simps*)
moreover have *b-of* τ'_2 = *b-of* $\tau'_2[x::=v]_{\tau v}$ **using** *t2'' subst-tbase-eq*
by (*metis* τ.*exhaust b-of.simps*)
moreover have *b-of* $\tau'_2[x::=v]_{\tau v}$ = *b-of* τ'_2 ∧ *b-of* $\tau'_1[x::=v]_{\tau v}$ = *b-of* τ'_1
using *subst-t-b-eq* **by** *auto*
ultimately show *?thesis* **using** *t1t2 b-of.simps* **by** *metis*
qed
ultimately show *?thesis* **using** *subst-infer-aux* **by** *meson*
qed

ultimately show *?case* **using** *subst-vv.simps* **by** *auto*

next
case (*V-cons s dc w*)

Proof outline: unpack using elimination, apply IH to type of w and then repack using infer v consI

have *eq1*: $(V\text{-}cons\ s\ dc\ w)[x::=v]_{vv} = V\text{-}cons\ s\ dc\ (w[x::=v]_{vv})$ **using** *subst-vv.simps* **by** *presburger*

obtain *dclist x2 b2 c2 z' c' z* **where** $*$:
$\tau_2 = \{\!|\ z : B\text{-}id\ s\ |\ CE\text{-}val\ (V\text{-}var\ z) == CE\text{-}val\ (V\text{-}cons\ s\ dc\ w)\ |\!\} \wedge$
$AF\text{-}typedef\ s\ dclist \in set\ \Theta \wedge$
$(dc, \{\!|\ x2 : b2\ |\ c2\ |\!\}) \in set\ dclist \wedge$
$(\Theta\ ;\ \mathcal{B}\ ;\ \Gamma'\ @\ (x,\ b_1,\ c0[z0::=[x]^v]_{cv})\ \#_\Gamma\ \Gamma\ \vdash w \Rightarrow \{\!|\ z' : b2\ |\ c'\ |\!\}) \wedge$
$(\Theta\ ;\ \mathcal{B}\ ;\ \Gamma'\ @\ (x,\ b_1,\ c0[z0::=[x]^v]_{cv})\ \#_\Gamma\ \Gamma\ \vdash \{\!|\ z' : b2\ |\ c'\ |\!\} \lesssim \{\!|\ x2 : b2\ |\ c2\ |\!\}) \wedge$
$atom\ z \sharp w \wedge atom\ z \sharp \Gamma'\ @\ (x,\ b_1,\ c0[z0::=[x]^v]_{cv})\ \#_\Gamma\ \Gamma$
using *infer-v-elims(4)[OF V-cons(3)]* **by** *metis*

obtain τ_3' **where** *yy*: $\Theta\ ;\ \mathcal{B}\ ;\ (\Gamma'[x::=v]_{\Gamma v}@\Gamma) \vdash w[x::=v]_{vv} \Rightarrow \{\!|\ z' : b2\ |\ c'\ |\!\}[x::=v]_{\tau v}$
using *V-cons (1)[of v x* $\{\!|\ z' : b2\ |\ c'\ |\!\}$*]* **using** *V-cons* $*$ **by** *auto*
then obtain *z3 b3 c3* **where** *yy2*: $atom\ z3 \sharp (x,v) \wedge \{\!|\ z' : b2\ |\ c'\ |\!\}[x::=v]_{\tau v} = \{\!|\ z3 : b3\ |\ c3\ |\!\}$ **using** *obtain-fresh-z* **by** *metis*

hence *b2=b3* **using** *subtype-eq-base2 yy subst-tbase-eq b-of.simps* **by** *metis*

have *zvf*: $atom\ z \sharp (x,v)$ **using** *infer-v-fresh-g-fresh-xv* $*$ *V-cons infer-v-wf* **by** *blast*
hence *zf*: $atom\ z \sharp CE\text{-}val\ (V\text{-}cons\ s\ dc\ (w[x::=v]_{vv}))$
unfolding *ce.fresh v.fresh* **by**(*simp add*: *pure-fresh* $*$)

obtain *z0'*::*x* **where** *z0*:$atom\ z0' \sharp (x,v,w[x::=v]_{vv}, \Gamma'[x::=v]_{\Gamma v}\ @\ \Gamma, CE\text{-}val\ (V\text{-}cons\ s\ dc\ (w[x::=v]_{vv})))$ **using** *obtain-fresh* **by** *metis*
hence *zf2*: $atom\ z0' \sharp CE\text{-}val\ (V\text{-}cons\ s\ dc\ (w[x::=v]_{vv}))$ **using** *e.fresh fresh-Pair v.fresh pure-fresh fresh-prod5* **by** *metis*
hence *zeq*: $\{\!|\ z : B\text{-}id\ s\ |\ CE\text{-}val\ (V\text{-}var\ z) == CE\text{-}val\ (V\text{-}cons\ s\ dc\ (w[x::=v]_{vv}))\ |\!\} =$
$\{\!|\ z0' : B\text{-}id\ s\ |\ CE\text{-}val\ (V\text{-}var\ z0') == CE\text{-}val\ (V\text{-}cons\ s\ dc\ (w[x::=v]_{vv}))\ |\!\}$ **using** *type-e-eq zf e.fresh fresh-Pair* **by** *metis*
moreover have *teq*: $\{\!|\ z : B\text{-}id\ s\ |\ CE\text{-}val\ (V\text{-}var\ z) == CE\text{-}val\ (V\text{-}cons\ s\ dc\ (w[x::=v]_{vv}))\ |\!\} = \tau_2[x::=v]_{\tau v}$
using $*$ *subst-tv.simps subst-ev.simps subst-vv.simps zvf* **by** *simp*

have $**$:$\Theta\ ;\ \mathcal{B}\ ;\ \Gamma'[x::=v]_{\Gamma v}\ @\ \Gamma\ \vdash V\text{-}cons\ s\ dc\ w[x::=v]_{vv} \Rightarrow \{\!|\ z0' : B\text{-}id\ s\ |\ CE\text{-}val\ (V\text{-}var\ z0') == CE\text{-}val\ (V\text{-}cons\ s\ dc\ (w[x::=v]_{vv}))\ |\!\}$
proof
show ‹*AF-typedef s dclist* $\in$ *set* Θ› **using** $*$ **by** *auto*
show ‹$(dc, \{\!|\ x2 : b2\ |\ c2\ |\!\}) \in set\ dclist$› **using** $*$ **by** *auto*
show $***$:‹$\Theta\ ;\ \mathcal{B}\ ;\ \Gamma'[x::=v]_{\Gamma v}\ @\ \Gamma\ \vdash w[x::=v]_{vv} \Rightarrow \{\!|\ z3 : b2\ |\ c3\ |\!\}$› **using** *yy yy2* ‹*b2=b3*› **by** *auto*
show ‹$\Theta\ ;\ \mathcal{B}\ ;\ \Gamma'[x::=v]_{\Gamma v}\ @\ \Gamma\ \vdash \{\!|\ z3 : b2\ |\ c3\ |\!\} \lesssim \{\!|\ x2 : b2\ |\ c2\ |\!\}$› **proof** −
have *xx*: $\Theta\ ;\ \mathcal{B}\ ;\ \Gamma'@((x,b_1,c0[z0::=[x]^v]_{cv})\#_\Gamma\Gamma)\ \vdash \{\!|\ z' : b2\ |\ c'\ |\!\} \lesssim \{\!|\ x2 : b2\ |\ c2\ |\!\}$ **using** $*$ **by** *auto*
hence $\Theta\ ;\ \mathcal{B}\ ;\ (\Gamma'[x::=v]_{\Gamma v}@\Gamma) \vdash \{\!|\ z' : b2\ |\ c'\ |\!\}[x::=v]_{\tau v} \lesssim \{\!|\ x2 : b2\ |\ c2\ |\!\}[x::=v]_{\tau v}$ **using** *subst-subtype-tau*[*OF V-cons(2) assms(3) xx V-cons(5)*] **by** *auto*
moreover have $\vdash_{wf} \Theta$ **using** *infer-v-wf* $*$ **by** *auto*
moreover hence $\{\!|\ x2 : b2\ |\ c2\ |\!\}[x::=v]_{\tau v} = \{\!|\ x2 : b2\ |\ c2\ |\!\}$ **using** *dc-t-closed(1)* $*$ *forget-subst-tv*

fresh-def wfG-nilI **by** *fast*
moreover have Θ ; $\mathcal{B}$; $(\Gamma'[x::=v]_{\Gamma v}@\Gamma) \vdash \{\!|\ z3 : b2\ |\ c3\ |\!\} \lesssim \{\!|\ z' : b2\ |\ c'\ |\!\}[x::=v]_{\tau v}$ **using** *yy yy2* ‹*b2=b3*› *subtype-reflI infer-v-t-wf*[*OF* ***] **by** *metis*
ultimately show *?thesis* **using** *subtype-trans* **by** *metis*
qed
show ‹*atom z0'* ♯ $w[x::=v]_{vv}$› **using** *z0 fresh-Pair* **by** *metis*
show ‹*atom z0'* ♯ $\Gamma'[x::=v]_{\Gamma v}$ @ Γ› **using** *z0* **by** *auto*
qed

moreover hence Θ ; $\mathcal{B}$; $\Gamma'[x::=v]_{\Gamma v}$ @ Γ $\vdash$ $\{\!|\ z0' : B\text{-}id\ s\ |\ CE\text{-}val\ (V\text{-}var\ z0') == CE\text{-}val\ (V\text{-}cons\ s\ dc\ (w[x::=v]_{vv}))\ |\!\} \lesssim \tau_2[x::=v]_{\tau v}$
using *subtype-reflI teq zeq infer-v-t-wf* **by** *metis*

ultimately show *?case* **using** *zeq teq* **by** *auto*
next
case (*V-consp s dc b w*)
from *V-consp*(*3*) *V-consp*(*1,2,4,5*) **show** *?case*
proof(*nominal-induct* Γ' @ (*x*, b_1, *c0*[*z0*::=[*x*]^v]$_{cv}$) $\#_\Gamma$ Γ *V-consp s dc b w* τ_2 *avoiding*: *x v rule*: *infer-v.strong-induct*)
case (*infer-v-conspI bv dclist* Θ *tc* $\mathcal{B}$ *tv z*)

have Θ ; $\mathcal{B}$; $\Gamma'[x::=v]_{\Gamma v}$ @ Γ $\vdash$ (*V-consp s dc b* $w[x::=v]_{vv}$) $\Rightarrow$ $\{\!|\ z : B\text{-}app\ s\ b\ |\ [\ [\ z\ ]^v\ ]^{ce} == [\ V\text{-}consp\ s\ dc\ b\ w[x::=v]_{vv}\ ]^{ce}\ |\!\}$
proof(*rule Typing.infer-v-conspI*[*OF infer-v-conspI*(*5,6*)])
show ‹ Θ ; $\mathcal{B}$; $\Gamma'[x::=v]_{\Gamma v}$ @ Γ $\vdash$ $w[x::=v]_{vv}$ $\Rightarrow$ $tv[x::=v]_{\tau v}$› **proof** −
have *atom z0* ♯ (*x*, *v*) **using** *infer-v-conspI* **by** *metis*
hence Θ ; $\mathcal{B}$; $\Gamma'[x::=v]_{\Gamma v}$ @ Γ $\vdash$ $w[x::=v]_{vv}$ $\Rightarrow$ $tv[x::=v]_{\tau v}$
using *infer-v-conspI*(*21*) *infer-v-conspI*(*24*) *infer-v-conspI*(*3*) *infer-v-conspI* **by** *metis*
thus *?thesis* **using** *subst-tv.simps* **by** *auto*
qed
show ‹Θ ; $\mathcal{B}$; $\Gamma'[x::=v]_{\Gamma v}$ @ Γ $\vdash$ $tv[x::=v]_{\tau v} \lesssim tc[bv::=b]_{\tau b}$› **proof** −
have Θ ; $\mathcal{B}$; $\Gamma'[x::=v]_{\Gamma v}$ @ Γ $\vdash$ $tv[x::=v]_{\tau v} \lesssim tc[bv::=b]_{\tau b}[x::=v]_{\tau v}$
using *infer-v-conspI subst-subtype-tau* **by** *metis*
moreover have *atom x* ♯ $tc[bv::=b]_{\tau b}$ **proof** −
have *supp tc* ⊆ { *atom bv* } **using** *wfTh-poly-lookup-supp infer-v-conspI wfX-wfY* **by** *metis*
hence *atom x* ♯ *tc* **using** *x-not-in-b-set*
using *fresh-def* **by** *fastforce*
moreover have *atom x* ♯ *b* **using** *x-fresh-b* **by** *auto*
ultimately show *?thesis* **using** *fresh-subst-if subst-b-τ-def* **by** *metis*
qed
ultimately show *?thesis* **using** *forget-subst-v subst-v-τ-def* **by** *metis*
qed
show ‹*atom z* ♯ (Θ, $\mathcal{B}$, $\Gamma'[x::=v]_{\Gamma v}$ @ Γ, $w[x::=v]_{vv}$, *b*)› **proof** −
have *atom z* ♯ $w[x::=v]_{vv}$ **using** *fresh-subst-v-if infer-v-conspI subst-v-v-def* **by** *metis*
moreover have *atom z* ♯ $\Gamma'[x::=v]_{\Gamma v}$ @ Γ **using** *fresh-subst-gv-inside infer-v-conspI* **by** *metis*
ultimately show *?thesis* **using** *fresh-prodN infer-v-conspI* **by** *metis*
qed
show ‹*atom bv* ♯ (Θ, $\mathcal{B}$, $\Gamma'[x::=v]_{\Gamma v}$ @ Γ, $w[x::=v]_{vv}$, *b*)› **proof** −
have *atom bv* ♯ $w[x::=v]_{vv}$ **using** *fresh-subst-v-if infer-v-conspI subst-v-v-def* **by** *metis*
moreover have *atom bv* ♯ $\Gamma'[x::=v]_{\Gamma v}$ @ Γ **using** *fresh-subst-gv-inside infer-v-conspI* **by** *metis*
ultimately show *?thesis* **using** *fresh-prodN infer-v-conspI* **by** *metis*
qed

show ‹ Θ ; $\mathcal{B} \vdash_{wf} b$ › **using** *infer-v-conspI* **by** *metis*
qed
moreover have *atom z* ♯ (*x*,*v*) **using** *infer-v-conspI fresh-Pair* **by** *metis*
ultimately show *?case* **using** *subst-vv.simps subst-tv.simps* **by** *auto*
qed
qed

lemma *subst-infer-check-v*:
fixes *v*::*v* **and** *v'*::*v*
assumes Θ ; $\mathcal{B}$; $\Gamma \vdash v \Rightarrow \tau_1$ **and**
check-v Θ $\mathcal{B}$ (Γ'@(x,b_1,*c0*[*z0*::=[*x*]^v]$_{cv}$)#$_\Gamma\Gamma$)) v' τ_2 **and**
Θ ; $\mathcal{B}$; $\Gamma \vdash \tau_1 \lesssim$ {| *z0* : b_1 | *c0* |} **and** *atom z0* ♯ (*x*,*v*)
shows *check-v* Θ $\mathcal{B}$ ((Γ'[*x*::=*v*]$_{\Gamma v}$)@Γ) (v'[*x*::=*v*]$_{vv}$) (τ_2[*x*::=*v*]$_{\tau v}$)
proof −
obtain τ_2' **where** *t2*: *infer-v* Θ $\mathcal{B}$ (Γ' @ (x, b_1, *c0*[*z0*::=[*x*]^v]$_{cv}$) #$_\Gamma$ Γ) v' τ_2' ∧ Θ ; $\mathcal{B}$; (Γ' @ (x, b_1, *c0*[*z0*::=[*x*]^v]$_{cv}$) #$_\Gamma$ Γ) ⊢ $\tau_2' \lesssim \tau_2$
using *check-v-elims assms* **by** *blast*
hence *infer-v* Θ $\mathcal{B}$ ((Γ'[*x*::=*v*]$_{\Gamma v}$)@Γ) (v'[*x*::=*v*]$_{vv}$) (τ_2'[*x*::=*v*]$_{\tau v}$)
using *subst-infer-v*[*OF assms*(*1*) - *assms*(*3*) *assms*(*4*)] **by** *blast*
moreover hence Θ; $\mathcal{B}$; ((Γ'[*x*::=*v*]$_{\Gamma v}$)@Γ) ⊢τ_2'[*x*::=*v*]$_{\tau v} \lesssim$ τ_2[*x*::=*v*]$_{\tau v}$
using *subst-subtype assms t2* **by** (*meson subst-subtype-tau subtype-trans*)
ultimately show *?thesis* **using** *check-v.intros* **by** *blast*
qed

lemma *type-veq-subst*[*simp*]:
assumes *atom z* ♯ (*x*,*v*)
shows {| *z* : *b* | *CE-val* (*V-var z*) == *CE-val v'* |}[*x*::=*v*]$_{\tau v}$ = {| *z* : *b* | *CE-val* (*V-var z*) == *CE-val v'*[*x*::=*v*]$_{vv}$ |}
using *assms* **by** *auto*

lemma *subst-infer-v-form*:
fixes *v*::*v* **and** *v'*::*v* **and** Γ::Γ
assumes Θ ; $\mathcal{B}$; $\Gamma \vdash v \Rightarrow \tau_1$ **and**
Θ ; $\mathcal{B}$; Γ'@((*x*,*b*$_1$,*c0*[*z0*::=[*x*]^v]$_{cv}$)#$_\Gamma\Gamma$) ⊢ $v' \Rightarrow \tau_2$ **and** *b*= *b-of* τ_2
Θ ; $\mathcal{B}$; $\Gamma \vdash \tau_1 \lesssim$ ({| *z0* : b_1 | *c0* |}) **and** *atom z0* ♯ (*x*,*v*) **and** *atom z3'* ♯ (*x*,*v*,*v'*, Γ'@((*x*,*b*$_1$,*c0*[*z0*::=[*x*]^v]$_{cv}$)#$_\Gamma\Gamma$))
shows ‹ Θ ; $\mathcal{B}$; Γ'[*x*::=*v*]$_{\Gamma v}$ @ Γ ⊢ v'[*x*::=*v*]$_{vv}$ ⇒ {| *z3'* : *b* | *CE-val* (*V-var z3'*) == *CE-val v'*[*x*::=*v*]$_{vv}$ |}›
proof −
have Θ ; $\mathcal{B}$; Γ'@((*x*,*b*$_1$,*c0*[*z0*::=[*x*]^v]$_{cv}$)#$_\Gamma\Gamma$) ⊢ $v' \Rightarrow$ {| *z3'* : *b-of* τ_2 | *C-eq* (*CE-val* (*V-var z3'*)) (*CE-val v'*) |}
proof(*rule infer-v-form4*)
show ‹ Θ ; $\mathcal{B}$; Γ' @ (x, b_1, *c0*[*z0*::=[*x*]^v]$_{cv}$) #$_\Gamma$ Γ ⊢ $v' \Rightarrow \tau_2$› **using** *assms* **by** *metis*
show ‹*atom z3'* ♯ (v', Γ' @ (x, b_1, *c0*[*z0*::=[*x*]^v]$_{cv}$) #$_\Gamma$ Γ)› **using** *assms fresh-prodN* **by** *metis*
show ‹*b-of* τ_2 = *b-of* τ_2› **by** *auto*
qed
hence ‹ Θ ; $\mathcal{B}$; Γ'[*x*::=*v*]$_{\Gamma v}$ @ Γ ⊢ v'[*x*::=*v*]$_{vv} \Rightarrow$ {| *z3'* : *b-of* τ_2 | *CE-val* (*V-var z3'*) == *CE-val v'* |}[*x*::=*v*]$_{\tau v}$›
using *subst-infer-v assms* **by** *metis*
thus *?thesis* **using** *type-veq-subst fresh-prodN assms* **by** *metis*
qed

14.7 Expressions

For operator, fst and snd cases, we use elimination to get one or more values, apply the substitution lemma for values. The types always have the same form and are equal under substitution. For function application, the subst value is a subtype of the value which is a subtype of the argument. The return of the function is the same under substitution.

Observe a similar pattern for each case

lemma *subst-infer-e*:
fixes $v::v$ **and** $e::e$ **and** $\Gamma'::\Gamma$
assumes
Θ ; Φ ; $\mathcal{B}$; G ; $\Delta \vdash e \Rightarrow \tau_2$ **and** $G = (\Gamma'@((x,b_1,$*subst-cv c0 z0* $(V\text{-}var\ x))\#_\Gamma\Gamma))$
Θ ; $\mathcal{B}$; $\Gamma \vdash v \Rightarrow \tau_1$ **and**
Θ; $\mathcal{B}$; $\Gamma \vdash \tau_1 \lesssim \{\!|\ z0 : b_1 \mid c0\ |\!\}$ **and** *atom z0* $\sharp$ (x,v)
shows Θ ; Φ ; $\mathcal{B}$; $((\Gamma'[x::=v]_{\Gamma v})@\Gamma)$; $(\Delta[x::=v]_{\Delta v}) \vdash$ (*subst-ev e x v*) $\Rightarrow \tau_2[x::=v]_{\tau v}$
using *assms* **proof**(*nominal-induct avoiding*: *x v rule*: *infer-e.strong-induct*)
case (*infer-e-valI* Θ $\mathcal{B}$ Γ'' Δ Φ v' τ)

have Θ ; Φ ; $\mathcal{B}$; $\Gamma'[x::=v]_{\Gamma v}$ @ Γ ; $\Delta[x::=v]_{\Delta v} \vdash$ (*AE-val* $(v'[x::=v]_{vv})) \Rightarrow \tau[x::=v]_{\tau v}$
proof
show Θ ; $\mathcal{B}$; $\Gamma'[x::=v]_{\Gamma v}$ @ $\Gamma \vdash_{wf} \Delta[x::=v]_{\Delta v}$ **using** *wfD-subst infer-e-valI subtype-eq-base2*
by (*metis b-of.simps infer-v-v-wf subst-g-inside-simple wfD-wf wf-subst*(*11*))
show $\Theta \vdash_{wf} \Phi$ **using** *infer-e-valI* **by** *auto*
show Θ ; $\mathcal{B}$; $\Gamma'[x::=v]_{\Gamma v}$ @ $\Gamma \vdash v'[x::=v]_{vv} \Rightarrow \tau[x::=v]_{\tau v}$ **using** *subst-infer-v infer-e-valI* **using** *wfD-subst infer-e-valI subtype-eq-base2*
by *metis*
qed
thus *?case* **using** *subst-ev.simps* **by** *simp*
next
case (*infer-e-plusI* Θ $\mathcal{B}$ Γ'' Δ Φ *v1 z1 c1 v2 z2 c2 z3*)

hence *z3f*: *atom z3* $\sharp$ *CE-op Plus* $[v1]^{ce}$ $[v2]^{ce}$ **using** *e.fresh ce.fresh opp.fresh* **by** *metis*

obtain *z3'*::*x* **where** $*$:*atom z3'* $\sharp$ $(x,v,$*AE-op Plus v1 v2*, *CE-op Plus* $[v1]^{ce}$ $[v2]^{ce}$, *AE-op Plus* $v1[x::=v]_{vv}$ $v2[x::=v]_{vv}$, *CE-op Plus* $[v1[x::=v]_{vv}]^{ce}$ $[v2[x::=v]_{vv}]^{ce}$,$\Gamma'[x::=v]_{\Gamma v}$ @ Γ)
using *obtain-fresh* **by** *metis*
hence $**$:$(\{\!|\ z3 : B\text{-}int \mid CE\text{-}val\ (V\text{-}var\ z3) == CE\text{-}op\ Plus\ [v1]^{ce}\ [v2]^{ce}\ |\!\}) = \{\!|\ z3' : B\text{-}int \mid CE\text{-}val\ (V\text{-}var\ z3') == CE\text{-}op\ Plus\ [v1]^{ce}\ [v2]^{ce}\ |\!\}$
using *type-e-eq infer-e-plusI fresh-Pair z3f* **by** *metis*

obtain *z1' b1' c1'* **where** *z1*:*atom z1'* $\sharp$ $(x,v) \wedge \{\!|\ z1 : B\text{-}int \mid c1\ |\!\} = \{\!|\ z1' : b1' \mid c1'\ |\!\}$ **using** *obtain-fresh-z* **by** *metis*
obtain *z2' b2' c2'* **where** *z2*:*atom z2'* $\sharp$ $(x,v) \wedge \{\!|\ z2 : B\text{-}int \mid c2\ |\!\} = \{\!|\ z2' : b2' \mid c2'\ |\!\}$ **using** *obtain-fresh-z* **by** *metis*

have *bb*:$b1' = B\text{-}int \wedge b2' = B\text{-}int$ **using** *z1 z2 τ.eq-iff* **by** *metis*

have Θ ; Φ ; $\mathcal{B}$; $\Gamma'[x::=v]_{\Gamma v}$ @ Γ ; $\Delta[x::=v]_{\Delta v} \vdash$ (*AE-op Plus* $(v1[x::=v]_{vv})$ $(v2[x::=v]_{vv})) \Rightarrow \{\!|\ z3' : B\text{-}int \mid CE\text{-}val\ (V\text{-}var\ z3') == CE\text{-}op\ Plus\ ([v1[x::=v]_{vv}]^{ce})\ ([v2[x::=v]_{vv}]^{ce})\ |\!\}$
proof
show ‹ Θ ; $\mathcal{B}$; $\Gamma'[x::=v]_{\Gamma v}$ @ $\Gamma \vdash_{wf} \Delta[x::=v]_{\Delta v}$ ›
using *infer-e-plusI wfD-subst subtype-eq-base2 b-of.simps* **by** *metis*

show ‹ $\Theta \vdash_{wf} \Phi$ › **using** *infer-e-plusI* **by** *blast*
show ‹ Θ ; $\mathcal{B}$; $\Gamma'[x::=v]_{\Gamma v}$ @ Γ $\vdash$ $v1[x::=v]_{vv} \Rightarrow \{\!|$ *z1'* : *B-int* | *c1'*$[x::=v]_{cv}$ $|\!\}$› **using** *subst-tv.simps subst-infer-v infer-e-plusI z1 bb* **by** *metis*
show ‹ Θ ; $\mathcal{B}$; $\Gamma'[x::=v]_{\Gamma v}$ @ Γ $\vdash$ $v2[x::=v]_{vv} \Rightarrow \{\!|$ *z2'* : *B-int* | *c2'*$[x::=v]_{cv}$ $|\!\}$› **using** *subst-tv.simps subst-infer-v infer-e-plusI z2 bb* **by** *metis*
show ‹*atom z3'* $\sharp$ *AE-op Plus* $v1[x::=v]_{vv}$ $v2[x::=v]_{vv}$› **using** *fresh-prod6* $*$ **by** *metis*
show ‹*atom z3'* $\sharp$ $\Gamma'[x::=v]_{\Gamma v}$ @ Γ› **using** $*$ **by** *auto*
qed
moreover have $\{\!|$ *z3'* : *B-int* | *CE-val* (*V-var z3'*) == *CE-op Plus* ($[v1[x::=v]_{vv}]^{ce}$) ($[v2[x::=v]_{vv}]^{ce}$) $|\!\}$ = $\{\!|$ *z3'* : *B-int* | *CE-val* (*V-var z3'*) == *CE-op Plus* $[v1]^{ce}$ $[v2]^{ce}$ $|\!\}[x::=v]_{\tau v}$
by(*subst subst-tv.simps*,*auto simp add*: $*$)
ultimately show *?case* **using** *subst-ev.simps* $*$ $**$ **by** *metis*
next
case (*infer-e-leqI* Θ $\mathcal{B}$ Γ'' Δ Φ *v1 z1 c1 v2 z2 c2 z3*)

hence *z3f*: *atom z3* $\sharp$ *CE-op LEq* $[v1]^{ce}$ $[v2]^{ce}$ **using** *e.fresh ce.fresh opp.fresh* **by** *metis*

obtain *z3'*::*x* **where** $*$:*atom z3'* $\sharp$ (*x*,*v*,*AE-op LEq v1 v2*, *CE-op LEq* $[v1]^{ce}$ $[v2]^{ce}$, *CE-op LEq* $[v1[x::=v]_{vv}]^{ce}$ $[v2[x::=v]_{vv}]^{ce}$, *AE-op LEq* $v1[x::=v]_{vv}$ $v2[x::=v]_{vv}$,$\Gamma'[x::=v]_{\Gamma v}$ @ Γ)
using *obtain-fresh* **by** *metis*
hence $**$:($\{\!|$ *z3* : *B-bool* | *CE-val* (*V-var z3*) == *CE-op LEq* $[v1]^{ce}$ $[v2]^{ce}$ $|\!\}$) = $\{\!|$ *z3'* : *B-bool* | *CE-val* (*V-var z3'*) == *CE-op LEq* $[v1]^{ce}$ $[v2]^{ce}$ $|\!\}$
using *type-e-eq infer-e-leqI fresh-Pair z3f* **by** *metis*

obtain *z1' b1' c1'* **where** *z1*:*atom z1'* $\sharp$ (*x*,*v*) $\wedge$ $\{\!|$ *z1* : *B-int* | *c1* $|\!\}$ = $\{\!|$ *z1'* : *b1'* | *c1'* $|\!\}$ **using** *obtain-fresh-z* **by** *metis*
obtain *z2' b2' c2'* **where** *z2*:*atom z2'* $\sharp$ (*x*,*v*) $\wedge$ $\{\!|$ *z2* : *B-int* | *c2* $|\!\}$ = $\{\!|$ *z2'* : *b2'* | *c2'* $|\!\}$ **using** *obtain-fresh-z* **by** *metis*

have *bb*:*b1'* = *B-int* $\wedge$ *b2'* = *B-int* **using** *z1 z2* τ*.eq-iff* **by** *metis*

have Θ ; Φ ; $\mathcal{B}$; $\Gamma'[x::=v]_{\Gamma v}$ @ Γ ; $\Delta[x::=v]_{\Delta v}$ $\vdash$ (*AE-op LEq* ($v1[x::=v]_{vv}$) ($v2[x::=v]_{vv}$)) $\Rightarrow$ $\{\!|$ *z3'* : *B-bool* | *CE-val* (*V-var z3'*) == *CE-op LEq* ($[v1[x::=v]_{vv}]^{ce}$) ($[v2[x::=v]_{vv}]^{ce}$) $|\!\}$
proof
show ‹ Θ ; $\mathcal{B}$; $\Gamma'[x::=v]_{\Gamma v}$ @ $\Gamma \vdash_{wf} \Delta[x::=v]_{\Delta v}$ › **using** *wfD-subst infer-e-leqI subtype-eq-base2 b-of.simps* **by** *metis*
show ‹ $\Theta \vdash_{wf} \Phi$ › **using** *infer-e-leqI*(*2*) **by** *auto*
show ‹ Θ ; $\mathcal{B}$; $\Gamma'[x::=v]_{\Gamma v}$ @ Γ $\vdash$ $v1[x::=v]_{vv} \Rightarrow \{\!|$ *z1'* : *B-int* | *c1'*$[x::=v]_{cv}$ $|\!\}$› **using** *subst-tv.simps subst-infer-v infer-e-leqI z1 bb* **by** *metis*
show ‹ Θ ; $\mathcal{B}$; $\Gamma'[x::=v]_{\Gamma v}$ @ Γ $\vdash$ $v2[x::=v]_{vv} \Rightarrow \{\!|$ *z2'* : *B-int* | *c2'*$[x::=v]_{cv}$ $|\!\}$› **using** *subst-tv.simps subst-infer-v infer-e-leqI z2 bb* **by** *metis*
show ‹*atom z3'* $\sharp$ *AE-op LEq* $v1[x::=v]_{vv}$ $v2[x::=v]_{vv}$› **using** *fresh-Pair* $*$ **by** *metis*
show ‹*atom z3'* $\sharp$ $\Gamma'[x::=v]_{\Gamma v}$ @ Γ› **using** $*$ **by** *auto*
qed
moreover have $\{\!|$ *z3'* : *B-bool* | *CE-val* (*V-var z3'*) == *CE-op LEq* ($[v1[x::=v]_{vv}]^{ce}$) ($[v2[x::=v]_{vv}]^{ce}$) $|\!\}$ = $\{\!|$ *z3'* : *B-bool* | *CE-val* (*V-var z3'*) == *CE-op LEq* $[v1]^{ce}$ $[v2]^{ce}$ $|\!\}[x::=v]_{\tau v}$
using *subst-tv.simps subst-ev.simps* $*$ **by** *auto*
ultimately show *?case* **using** *subst-ev.simps* $*$ $**$ **by** *metis*
next
case (*infer-e-appI* Θ $\mathcal{B}$ Γ'' Δ Φ *f x' b c* τ' *s' v'* τ)

hence $x \neq x'$ **using** ‹*atom x'* $\sharp$ Γ''› **using** *wfG-inside-x-neq wfX-wfY* **by** *metis*

show *?case* **proof**(*subst subst-ev.simps*,*rule*)
show ‹ Θ ; $\mathcal{B}$; $\Gamma'[x::=v]_{\Gamma v}$ @ $\Gamma \vdash_{wf} \Delta[x::=v]_{\Delta v}$ › **using** *infer-e-appI wfD-subst subtype-eq-base2 b-of.simps* **by** *metis*
show ‹ $\Theta \vdash_{wf} \Phi$ › **using** *infer-e-appI* **by** *metis*
show ‹*Some* (*AF-fundef f* (*AF-fun-typ-none* (*AF-fun-typ x' b c τ' s'*))) = *lookup-fun* Φ *f*› **using** *infer-e-appI* **by** *metis*

have ‹Θ ; $\mathcal{B}$; $\Gamma'[x::=v]_{\Gamma v}$ @ $\Gamma \vdash v'[x::=v]_{vv} \Leftarrow \{\!|\, x' : b \mid c \,|\!\}[x::=v]_{\tau v}$› **proof**(*rule subst-infer-check-v*)
show Θ ; $\mathcal{B}$; $\Gamma \vdash v \Rightarrow \tau_1$ **using** *infer-e-appI* **by** *metis*
show Θ ; $\mathcal{B}$; Γ' @ (x, b_1, *c0*[*z0*::=[x]^v]$_{cv}$) #$_\Gamma$ $\Gamma \vdash v' \Leftarrow \{\!|\, x' : b \mid c \,|\!\}$ **using** *infer-e-appI* **by** *metis*
show Θ ; $\mathcal{B}$; $\Gamma \vdash \tau_1 \lesssim \{\!|\, z0 : b_1 \mid c0 \,|\!\}$ **using** *infer-e-appI* **by** *metis*
show *atom z0* ♯ (x, v) **using** *infer-e-appI* **by** *metis*
qed
moreover have *atom x* ♯ *c* **using** *wfPhi-f-supp-c*[*OF infer-e-appI*(*3*)] *fresh-def* ‹$x \neq x'$›
by (*metis atom-eq-iff empty-iff infer-e-appI.hyps*(*2*) *insert-iff subset-singletonD*)

moreover hence *atom x* ♯ $\{\!|\, x' : b \mid c \,|\!\}$ **using** *τ.fresh supp-b-empty fresh-def* **by** *blast*
ultimately show ‹Θ ; $\mathcal{B}$; $\Gamma'[x::=v]_{\Gamma v}$ @ $\Gamma \vdash v'[x::=v]_{vv} \Leftarrow \{\!|\, x' : b \mid c \,|\!\}$› **using** *forget-subst-tv* **by** *metis*

have *atom x'* ♯ (*x*,*v*) **using** *infer-v-fresh-g-fresh-xv infer-e-appI check-v-wf* **by** *blast*

thus ‹*atom x'* ♯ $\Gamma'[x::=v]_{\Gamma v}$ @ Γ› **using** *infer-e-appI fresh-subst-gv wfD-wf subst-g-inside fresh-Pair* **by** *metis*
have *supp τ'* ⊆ { *atom x'* } ∪ *supp* $\mathcal{B}$ **using** *infer-e-appI wfT-supp wfPhi-f-simple-wfT*
by (*meson infer-e-appI.hyps*(*2*) *le-supI1 wfPhi-f-simple-supp-t*)
hence *atom x* ♯ *τ'* **using** ‹$x \neq x'$› *fresh-def supp-at-base x-not-in-b-set* **by** *fastforce*
thus ‹*τ'*[*x'*::=*v'*[*x*::=*v*]$_{vv}$]$_v$ = *τ*[*x*::=*v*]$_{\tau v}$› **using** *subst-tv-commute infer-e-appI subst-defs* **by** *metis*
qed
next
case (*infer-e-appPI* Θ $\mathcal{B}$ Γ'' Δ Φ *b' f bv x' b c τ' s' v' τ*)

hence $x \neq x'$ **using** ‹*atom x'* ♯ Γ''› **using** *wfG-inside-x-neq wfX-wfY* **by** *metis*

show *?case* **proof**(*subst subst-ev.simps*,*rule*)
show ‹ Θ ; $\mathcal{B}$; $\Gamma'[x::=v]_{\Gamma v}$ @ $\Gamma \vdash_{wf} \Delta[x::=v]_{\Delta v}$ › **using** *infer-e-appPI wfD-subst subtype-eq-base2 b-of.simps* **by** *metis*
show ‹ $\Theta \vdash_{wf} \Phi$ › **using** *infer-e-appPI*(*4*) **by** *auto*
show Θ ; $\mathcal{B} \vdash_{wf} b'$ **using** *infer-e-appPI*(*5*) **by** *auto*
show *Some* (*AF-fundef f* (*AF-fun-typ-some bv* (*AF-fun-typ x' b c τ' s'*))) = *lookup-fun* Φ *f* **using** *infer-e-appPI*(*6*) **by** *auto*

show Θ ; $\mathcal{B}$; $\Gamma'[x::=v]_{\Gamma v}$ @ $\Gamma \vdash v'[x::=v]_{vv} \Leftarrow \{\!|\, x' : b[bv::=b']_b \mid c[bv::=b']_b \,|\!\}$ **proof** −
have ‹Θ ; $\mathcal{B}$; $\Gamma'[x::=v]_{\Gamma v}$ @ $\Gamma \vdash v'[x::=v]_{vv} \Leftarrow \{\!|\, x' : b[bv::=b']_{bb} \mid c[bv::=b']_{cb} \,|\!\}[x::=v]_{\tau v}$› **proof**(*rule subst-infer-check-v*)
show Θ ; $\mathcal{B}$; $\Gamma \vdash v \Rightarrow \tau_1$ **using** *infer-e-appPI* **by** *metis*
show Θ ; $\mathcal{B}$; Γ' @ (x, b_1, *c0*[*z0*::=[x]^v]$_{cv}$) #$_\Gamma$ $\Gamma \vdash v' \Leftarrow \{\!|\, x' : b[bv::=b']_{bb} \mid c[bv::=b']_{cb} \,|\!\}$ **using** *infer-e-appPI subst-defs* **by** *metis*

show Θ ; $\mathcal{B}$; Γ $\vdash$ $\tau_1 \lesssim \{\!|$ *z0* : b_1 | *c0* $|\!\}$ **using** *infer-e-appPI* **by** *metis*
show *atom z0* $\sharp$ (x, v) **using** *infer-e-appPI* **by** *metis*
qed
moreover have *atom* x $\sharp$ c **proof** −
have *supp* $c \subseteq \{$*atom* x', *atom* $bv\}$ **using** *wfPhi-f-poly-supp-c*[*OF infer-e-appPI*(*6*)] *infer-e-appPI* **by** *metis*
thus *?thesis* **unfolding** *fresh-def* **using** ‹$x \neq x'$› *atom-eq-iff* **by** *auto*
qed
moreover hence *atom* x $\sharp$ $\{\!|$ x' : $b[bv::=b']_{bb}$ | $c[bv::=b']_{cb}$ $|\!\}$ **using** τ*.fresh supp-b-empty fresh-def subst-b-fresh-x*
by (*metis subst-b-c-def*)
ultimately show *?thesis* **using** *forget-subst-tv subst-defs* **by** *metis*
qed
have *supp* τ' $\subseteq$ { *atom* x', *atom* bv } **using** *wfPhi-f-poly-supp-t infer-e-appPI* **by** *metis*
hence *atom* x $\sharp$ τ' **using** *fresh-def* ‹$x \neq x'$› **by** *force*
hence $*$:*atom* x $\sharp$ $\tau'[bv::=b']_{\tau b}$ **using** *subst-b-fresh-x subst-b-*τ*-def* **by** *metis*
have *atom* x' $\sharp$ (x,v) **using** *infer-v-fresh-g-fresh-xv infer-e-appPI check-v-wf* **by** *blast*
thus *atom* x' $\sharp$ $\Gamma'[x::=v]_{\Gamma v}$ @ Γ **using** *infer-e-appPI fresh-subst-gv wfD-wf subst-g-inside fresh-Pair* **by** *metis*
show $\tau'[bv::=b']_b[x'::=v'[x::=v]_{vv}]_v$ = $\tau[x::=v]_{\tau v}$ **using** *infer-e-appPI subst-tv-commute*[*OF* $*$] *subst-defs* **by** *metis*
show *atom bv* $\sharp$ (Θ, Φ, $\mathcal{B}$, $\Gamma'[x::=v]_{\Gamma v}$ @ Γ, $\Delta[x::=v]_{\Delta v}$, b', $v'[x::=v]_{vv}$, $\tau[x::=v]_{\tau v}$)

apply(*unfold fresh-prodN*, *intro conjI*)
apply(*simp add*: *infer-e-appPI*)
apply(*simp add*: *infer-e-appPI*)
apply(*simp add*: *infer-e-appPI*)
apply(*subst subst-g-inside*[*symmetric*])
apply((*insert infer-e-appPI wfX-wfY*) [*1*], *fast*)
apply(*metis fresh-subst-gv-if infer-e-appPI*)
apply(*simp add*: *fresh-prodN fresh-subst-dv-if infer-e-appPI*)
apply(*simp add*: *infer-e-appPI*)
apply(*simp add*: *fresh-prodN fresh-subst-v-if subst-v-v-def infer-e-appPI*)
apply(*simp add*: *fresh-prodN fresh-subst-v-if subst-v-*τ*-def infer-e-appPI*)
done

qed
next
case (*infer-e-fstI* Θ $\mathcal{B}$ Γ'' Δ Φ v' z' *b1 b2 c z*)

hence *zf*: *atom z* $\sharp$ *CE-fst* $[v']^{ce}$ **using** *ce.fresh e.fresh opp.fresh* **by** *metis*

obtain *z3'*::x **where** $*$:*atom z3'* $\sharp$ (x,v,*AE-fst v'*, *CE-fst* $[v']^{ce}$, *AE-fst* $v'[x::=v]_{vv}$,$\Gamma'[x::=v]_{\Gamma v}$ @ Γ) **using** *obtain-fresh* **by** *auto*
hence $**$:($\{\!|$ z : *b1* | *CE-val* (*V-var z*) == *CE-fst* $[v']^{ce}$ $|\!\}$) = $\{\!|$ *z3'* : *b1* | *CE-val* (*V-var z3'*) == *CE-fst* $[v']^{ce}$ $|\!\}$
using *type-e-eq infer-e-fstI*(*4*) *fresh-Pair zf* **by** *metis*

obtain *z1' b1' c1'* **where** *z1*:*atom z1'* $\sharp$ (x,v) $\wedge$ $\{\!|$ z' : *B-pair b1 b2* | c $|\!\}$ = $\{\!|$ *z1'* : *b1'* | *c1'* $|\!\}$ **using** *obtain-fresh-z* **by** *metis*

have *bb*:*b1'* = *B-pair b1 b2* **using** *z1* τ*.eq-iff* **by** *metis*

have Θ ; Φ ; $\mathcal{B}$; $\Gamma'[x::=v]_{\Gamma v}$ @ Γ ; $\Delta[x::=v]_{\Delta v}$ $\vdash$ (*AE-fst* $v'[x::=v]_{vv}$) $\Rightarrow$ $\{\!|$ *z3'* : *b1* | *CE-val* (*V-var z3'*) == *CE-fst* $[v'[x::=v]_{vv}]^{ce}$ $|\!\}$
proof
show ‹ Θ ; $\mathcal{B}$; $\Gamma'[x::=v]_{\Gamma v}$ @ Γ $\vdash_{wf}$ $\Delta[x::=v]_{\Delta v}$ › **using** *wfD-subst infer-e-fstI subtype-eq-base2 b-of.simps* **by** *metis*
show ‹ $\Theta\vdash_{wf}$ Φ › **using** *infer-e-fstI* **by** *metis*
show ‹ Θ ; $\mathcal{B}$; $\Gamma'[x::=v]_{\Gamma v}$ @ Γ $\vdash$ $v'[x::=v]_{vv}$ $\Rightarrow$ $\{\!|$ *z1'* : *B-pair b1 b2* | *c1'*$[x::=v]_{cv}$ $|\!\}$› **using** *subst-tv.simps subst-infer-v infer-e-fstI z1 bb* **by** *metis*

show ‹*atom z3'* $\sharp$ *AE-fst* $v'[x::=v]_{vv}$ › **using** *fresh-Pair* $*$ **by** *metis*
show ‹*atom z3'* $\sharp$ $\Gamma'[x::=v]_{\Gamma v}$ @ Γ› **using** $*$ **by** *auto*
qed
moreover have $\{\!|$ *z3'* : *b1* | *CE-val* (*V-var z3'*) == *CE-fst* $[v'[x::=v]_{vv}]^{ce}$ $|\!\}$ = $\{\!|$ *z3'* : *b1* | *CE-val* (*V-var z3'*) == *CE-fst* $[v']^{ce}$ $|\!\}[x::=v]_{\tau v}$
using *subst-tv.simps subst-ev.simps* $*$ **by** *auto*
ultimately show *?case* **using** *subst-ev.simps* $*$ $**$ **by** *metis*
next
case (*infer-e-sndI* Θ $\mathcal{B}$ Γ'' Δ Φ *v' z' b1 b2 c z*)
hence *zf*: *atom z* $\sharp$ *CE-snd* $[v']^{ce}$ **using** *ce.fresh e.fresh opp.fresh* **by** *metis*

obtain *z3'*::*x* **where** $*$:*atom z3'* $\sharp$ (*x*,*v*,*AE-snd v'*, *CE-snd* $[v']^{ce}$, *AE-snd* $v'[x::=v]_{vv}$,$\Gamma'[x::=v]_{\Gamma v}$ @ Γ ,*v'*, Γ'') **using** *obtain-fresh* **by** *auto*
hence $**$:($\{\!|$ *z* : *b2* | *CE-val* (*V-var z*) == *CE-snd* $[v']^{ce}$ $|\!\}$) = $\{\!|$ *z3'* : *b2* | *CE-val* (*V-var z3'*) == *CE-snd* $[v']^{ce}$ $|\!\}$
using *type-e-eq infer-e-sndI*(*4*) *fresh-Pair zf* **by** *metis*

obtain *z1' b2' c1'* **where** *z1*:*atom z1'* $\sharp$ (*x*,*v*) $\wedge$ $\{\!|$ *z'*: *B-pair b1 b2* | *c* $|\!\}$ = $\{\!|$ *z1'* : *b2'* | *c1'* $|\!\}$ **using** *obtain-fresh-z* **by** *metis*

have *bb*:*b2'* = *B-pair b1 b2* **using** *z1* τ*.eq-iff* **by** *metis*

have Θ ; Φ ; $\mathcal{B}$; $\Gamma'[x::=v]_{\Gamma v}$ @ Γ ; $\Delta[x::=v]_{\Delta v}$ $\vdash$ (*AE-snd* ($v'[x::=v]_{vv}$)) $\Rightarrow$ $\{\!|$ *z3'* : *b2* | *CE-val* (*V-var z3'*) == *CE-snd* ($[v'[x::=v]_{vv}]^{ce}$) $|\!\}$
proof
show ‹ Θ ; $\mathcal{B}$; $\Gamma'[x::=v]_{\Gamma v}$ @ Γ $\vdash_{wf}$ $\Delta[x::=v]_{\Delta v}$ › **using** *wfD-subst infer-e-sndI subtype-eq-base2 b-of.simps* **by** *metis*
show ‹ $\Theta\vdash_{wf}$ Φ › **using** *infer-e-sndI* **by** *metis*
show ‹ Θ ; $\mathcal{B}$; $\Gamma'[x::=v]_{\Gamma v}$ @ Γ $\vdash$ $v'[x::=v]_{vv}$ $\Rightarrow$ $\{\!|$ *z1'* : *B-pair b1 b2* | *c1'*$[x::=v]_{cv}$ $|\!\}$› **using** *subst-tv.simps subst-infer-v infer-e-sndI z1 bb* **by** *metis*

show ‹*atom z3'* $\sharp$ *AE-snd* $v'[x::=v]_{vv}$ › **using** *fresh-Pair* $*$ **by** *metis*
show ‹*atom z3'* $\sharp$ $\Gamma'[x::=v]_{\Gamma v}$ @ Γ› **using** $*$ **by** *auto*
qed
moreover have $\{\!|$ *z3'* : *b2* | *CE-val* (*V-var z3'*) == *CE-snd* ($[v'[x::=v]_{vv}]^{ce}$) $|\!\}$ = $\{\!|$ *z3'* : *b2* | *CE-val* (*V-var z3'*) == *CE-snd* $[v']^{ce}$ $|\!\}[x::=v]_{\tau v}$
by(*subst subst-tv.simps*, *auto simp add*: *fresh-prodN* $*$)
ultimately show *?case* **using** *subst-ev.simps* $*$ $**$ **by** *metis*
next
case (*infer-e-lenI* Θ $\mathcal{B}$ Γ'' Δ Φ *v' z' c z*)
hence *zf*: *atom z* $\sharp$ *CE-len* $[v']^{ce}$ **using** *ce.fresh e.fresh opp.fresh* **by** *metis*
obtain *z3'*::*x* **where** $*$:*atom z3'* $\sharp$ (*x*,*v*,*AE-len v'*, *CE-len* $[v']^{ce}$, *AE-len* $v'[x::=v]_{vv}$,$\Gamma'[x::=v]_{\Gamma v}$ @ Γ , Γ'',*v'*) **using** *obtain-fresh* **by** *auto*

hence $**$:($\{\!|$ z : *B-int* $|$ *CE-val* (*V-var* z) $==$ *CE-len* $[v']^{ce}$ $|\!\}$) = $\{\!|$ $z3'$: *B-int* $|$ *CE-val* (*V-var* $z3'$) $==$ *CE-len* $[v']^{ce}$ $|\!\}$
using *type-e-eq infer-e-lenI fresh-Pair zf* **by** *metis*

have $***$: Θ ; $\mathcal{B}$; Γ'' $\vdash$ $v' \Rightarrow \{\!|$ $z3'$: *B-bitvec* $|$ *CE-val* (*V-var* $z3'$) == *CE-val* v' $|\!\}$
using *infer-e-lenI infer-v-form3*[*OF infer-e-lenI*(*3*), *of z3'*] *b-of.simps* $*$ *fresh-Pair* **by** *metis*

have Θ ; Φ ; $\mathcal{B}$; $\Gamma'[x::=v]_{\Gamma v}$ @ Γ ; $\Delta[x::=v]_{\Delta v}$ $\vdash$ (*AE-len* ($v'[x::=v]_{vv}$)) $\Rightarrow$ $\{\!|$ $z3'$: *B-int* $|$ *CE-val* (*V-var z3'*) $==$ *CE-len* ($[v'[x::=v]_{vv}]^{ce}$) $|\!\}$
proof
show ‹ Θ ; $\mathcal{B}$; $\Gamma'[x::=v]_{\Gamma v}$ @ Γ $\vdash_{wf}$ $\Delta[x::=v]_{\Delta v}$ › **using** *wfD-subst infer-e-lenI subtype-eq-base2 b-of.simps* **by** *metis*
show ‹ $\Theta\vdash_{wf}$ Φ › **using** *infer-e-lenI* **by** *metis*

have ‹ Θ ; $\mathcal{B}$; $\Gamma'[x::=v]_{\Gamma v}$ @ Γ $\vdash$ $v'[x::=v]_{vv} \Rightarrow \{\!|$ $z3'$: *B-bitvec* $|$ *CE-val* (*V-var z3'*) $==$ *CE-val* v' $|\!\}[x::=v]_{\tau v}$›
proof(*rule subst-infer-v*)
show ‹ Θ ; $\mathcal{B}$; $\Gamma \vdash v \Rightarrow \tau_1$› **using** *infer-e-lenI* **by** *metis*
show ‹ Θ ; $\mathcal{B}$; Γ' @ (x, b_1, *c0*[*z0*::=[x]^v]$_{cv}$) $\#_\Gamma$ $\Gamma \vdash v' \Rightarrow \{\!|$ $z3'$: *B-bitvec* $|$ [[$z3'$]^v]^{ce} $==$ [v']^{ce} $|\!\}$› **using** $***$ *infer-e-lenI* **by** *metis*
show Θ ; $\mathcal{B}$; Γ $\vdash \tau_1 \lesssim \{\!|$ *z0* : b_1 $|$ *c0* $|\!\}$ **using** *infer-e-lenI* **by** *metis*
show *atom z0* $\sharp$ (x, v) **using** *infer-e-lenI* **by** *metis*
qed

thus ‹ Θ ; $\mathcal{B}$; $\Gamma'[x::=v]_{\Gamma v}$ @ Γ $\vdash$ $v'[x::=v]_{vv} \Rightarrow \{\!|$ $z3'$: *B-bitvec* $|$ *CE-val* (*V-var z3'*) $==$ *CE-val* $v'[x::=v]_{vv}$ $|\!\}$›
using *subst-tv.simps subst-ev.simps fresh-Pair* $*$ *fresh-prodN subst-vv.simps* **by** *auto*

show ‹*atom z3'* $\sharp$ *AE-len* $v'[x::=v]_{vv}$› **using** *fresh-Pair* $*$ **by** *metis*
show ‹*atom z3'* $\sharp$ $\Gamma'[x::=v]_{\Gamma v}$ @ Γ› **using** *fresh-Pair* $*$ **by** *metis*
qed

moreover have $\{\!|$ $z3'$: *B-int* $|$ *CE-val* (*V-var z3'*) $==$ *CE-len* ($[v'[x::=v]_{vv}]^{ce}$) $|\!\}$ = $\{\!|$ $z3'$: *B-int* $|$ *CE-val* (*V-var z3'*) $==$ *CE-len* $[v']^{ce}$ $|\!\}[x::=v]_{\tau v}$
using *subst-tv.simps subst-ev.simps* $*$ **by** *auto*

ultimately show *?case* **using** *subst-ev.simps* $*$ $**$ **by** *metis*
next
case (*infer-e-mvarI* Θ $\mathcal{B}$ Γ'' Φ Δ u τ)

have Θ ; Φ ; $\mathcal{B}$; $\Gamma'[x::=v]_{\Gamma v}$ @ Γ ; $\Delta[x::=v]_{\Delta v}$ $\vdash$ (*AE-mvar u*) $\Rightarrow$ $\tau[x::=v]_{\tau v}$
proof
show ‹ Θ ; $\mathcal{B}\vdash_{wf}$ $\Gamma'[x::=v]_{\Gamma v}$ @ Γ› **proof** −
have *wfV* Θ $\mathcal{B}$ Γ v (*b-of* τ_1) **using** *infer-v-wf infer-e-mvarI* **by** *auto*
moreover have *b-of* τ_1 = b_1 **using** *subtype-eq-base2 infer-e-mvarI b-of.simps* **by** *simp*
ultimately show *?thesis* **using** *wf-subst*(*3*)[*OF infer-e-mvarI*(*1*), *of* Γ' x b_1 *c0*[*z0*::=[x]^v]$_{cv}$ Γ v] *infer-e-mvarI subst-g-inside* **by** *metis*
qed
show ‹ $\Theta\vdash_{wf}$ Φ › **using** *infer-e-mvarI* **by** *auto*
show ‹ Θ ; $\mathcal{B}$; $\Gamma'[x::=v]_{\Gamma v}$ @ $\Gamma \vdash_{wf}$ $\Delta[x::=v]_{\Delta v}$ › **using** *wfD-subst infer-e-mvarI subtype-eq-base2 b-of.simps* **by** *metis*
show ‹(u, $\tau[x::=v]_{\tau v}$) $\in$ *setD* $\Delta[x::=v]_{\Delta v}$› **using** *infer-e-mvarI subst-dv-member* **by** *metis*

qed
moreover have $(AE\text{-}mvar\ u) = (AE\text{-}mvar\ u)[x::=v]_{ev}$ **using** *subst-ev.simps* **by** *auto*
ultimately show *?case* **by** *auto*

next
case (*infer-e-concatI* $\Theta\ \mathcal{B}\ \Gamma''\ \Delta\ \Phi$ *v1 z1 c1 v2 z2 c2 z3*)

hence *zf*: *atom z3* $\sharp$ *CE-concat* $[v1]^{ce}$ $[v2]^{ce}$ **using** *ce.fresh e.fresh opp.fresh* **by** *metis*

obtain *z3'*::*x* **where** $*$:*atom z3'* $\sharp$ (*x*,*v*,*v1*,*v2*,*AE-concat v1 v2*, *CE-concat* $[v1]^{ce}$ $[v2]^{ce}$, *AE-concat* $(v1[x::=v]_{vv})$ $(v2[x::=v]_{vv})$,$\Gamma'[x::=v]_{\Gamma v}$ @ Γ , Γ'',*v1* , *v2*) **using** *obtain-fresh* **by** *auto*

hence $**$:($\{\!|$ *z3* : *B-bitvec* | *CE-val* (*V-var z3*) == *CE-concat* $[v1]^{ce}$ $[v2]^{ce}$ $|\!\}$) = $\{\!|$ *z3'* : *B-bitvec* | *CE-val* (*V-var z3'*) == *CE-concat* $[v1]^{ce}$ $[v2]^{ce}$ $|\!\}$
using *type-e-eq infer-e-concatI fresh-Pair zf* **by** *metis*
have *zfx*: *atom x* $\sharp$ *z3'* **using** *fresh-at-base fresh-prodN* $*$ **by** *auto*

have *v1*: Θ ; $\mathcal{B}$; Γ'' $\vdash$ *v1* $\Rightarrow$ $\{\!|$ *z3'* : *B-bitvec* | *CE-val* (*V-var z3'*) == *CE-val v1* $|\!\}$
using *infer-e-concatI infer-v-form3 b-of.simps* $*$ *fresh-Pair* **by** *metis*
have *v2*: Θ ; $\mathcal{B}$; Γ'' $\vdash$ *v2* $\Rightarrow$ $\{\!|$ *z3'* : *B-bitvec* | *CE-val* (*V-var z3'*) == *CE-val v2* $|\!\}$
using *infer-e-concatI infer-v-form3 b-of.simps* $*$ *fresh-Pair* **by** *metis*

have Θ ; Φ ; $\mathcal{B}$; $\Gamma'[x::=v]_{\Gamma v}$ @ Γ ; $\Delta[x::=v]_{\Delta v}$ $\vdash$ (*AE-concat* $(v1[x::=v]_{vv})$ $(v2[x::=v]_{vv})$) $\Rightarrow$ $\{\!|$ *z3'* : *B-bitvec* | *CE-val* (*V-var z3'*) == *CE-concat* $([v1[x::=v]_{vv}]^{ce})$ $([v2[x::=v]_{vv}]^{ce})$ $|\!\}$
proof
show ‹ Θ ; $\mathcal{B}$; $\Gamma'[x::=v]_{\Gamma v}$ @ Γ $\vdash_{wf}$ $\Delta[x::=v]_{\Delta v}$ › **using** *wfD-subst infer-e-concatI subtype-eq-base2 b-of.simps* **by** *metis*
show ‹ $\Theta\vdash_{wf}$ Φ › **by**(*simp add*: *infer-e-concatI*)
show ‹ Θ ; $\mathcal{B}$; $\Gamma'[x::=v]_{\Gamma v}$ @ Γ $\vdash$ *v1*$[x::=v]_{vv}$ $\Rightarrow$ $\{\!|$ *z3'* : *B-bitvec* | *CE-val* (*V-var z3'*) == *CE-val* $(v1[x::=v]_{vv})$ $|\!\}$›
using *subst-infer-v-form infer-e-concatI fresh-prodN* $*$ *b-of.simps* **by** *metis*
show ‹ Θ ; $\mathcal{B}$; $\Gamma'[x::=v]_{\Gamma v}$ @ Γ $\vdash$ *v2*$[x::=v]_{vv}$ $\Rightarrow$ $\{\!|$ *z3'* : *B-bitvec* | *CE-val* (*V-var z3'*) == *CE-val* $(v2[x::=v]_{vv})$ $|\!\}$›
using *subst-infer-v-form infer-e-concatI fresh-prodN* $*$ *b-of.simps* **by** *metis*
show ‹*atom z3'* $\sharp$ *AE-concat v1*$[x::=v]_{vv}$ *v2*$[x::=v]_{vv}$› **using** *fresh-Pair* $*$ **by** *metis*
show ‹*atom z3'* $\sharp$ $\Gamma'[x::=v]_{\Gamma v}$ @ Γ› **using** *fresh-Pair* $*$ **by** *metis*
qed

moreover have $\{\!|$ *z3'* : *B-bitvec* | *CE-val* (*V-var z3'*) == *CE-concat* $([v1[x::=v]_{vv}]^{ce})$ $([v2[x::=v]_{vv}]^{ce})$ $|\!\}$ = $\{\!|$ *z3'* : *B-bitvec* | *CE-val* (*V-var z3'*) == *CE-concat* $[v1]^{ce}$ $[v2]^{ce}$ $|\!\}[x::=v]_{\tau v}$
using *subst-tv.simps subst-ev.simps* $*$ **by** *auto*

ultimately show *?case* **using** *subst-ev.simps* $**$ $*$ **by** *metis*
next
thm *subst-tv.simps*
case (*infer-e-splitI* $\Theta\ \mathcal{B}\ \Gamma''\ \Delta\ \Phi$ *v1 z1 c1 v2 z2 z3*)
hence $*$:*atom z3* $\sharp$ (*x*,*v*) **using** *fresh-Pair* **by** *auto*
have ‹*x* $\neq$ *z3* › **using** *infer-e-splitI* **by** *force*
have Θ ; Φ ; $\mathcal{B}$; $\Gamma'[x::=v]_{\Gamma v}$ @
Γ ; $\Delta[x::=v]_{\Delta v}$ $\vdash$ *AE-split* *v1*$[x::=v]_{vv}$ *v2*$[x::=v]_{vv}$ $\Rightarrow$
$\{\!|$ *z3* : [*B-bitvec* , *B-bitvec*]^b | [*v1*$[x::=v]_{vv}$]^{ce} == [[#*1*[[*z3*]^v]^{ce}]^{ce} @@ [#*2*[[*z3*]^v]^{ce}]^{ce}]^{ce} *AND*

$[|\ [\#1[\ [\ z3\ ]^v\ ]^{ce}]^{ce}\ |]^{ce}\ ==\ [\ v2[x::=v]_{vv}\ ]^{ce}\ \}\!\rangle$

proof

show $\langle\ \Theta\ ;\ \mathcal{B}\ ;\ \Gamma'[x::=v]_{\Gamma v}\ @\ \Gamma \vdash_{wf}\ \Delta[x::=v]_{\Delta v}\ \rangle$ **using** *wfD-subst infer-e-splitI subtype-eq-base2 b-of.simps* **by** *metis*

show $\langle\ \Theta\ \vdash_{wf}\ \Phi\ \rangle$ **using** *infer-e-splitI* **by** *auto*

have $\langle\ \Theta\ ;\ \mathcal{B}\ ;\ \Gamma'[x::=v]_{\Gamma v}\ @\ \Gamma \vdash\ v1[x::=v]_{vv} \Rightarrow \{\!|\ z1\ :\ B\text{-}bitvec\ |\ c1\ |\!\}[x::=v]_{\tau v}\rangle$
using *subst-infer-v infer-e-splitI* **by** *metis*

thus $\langle\ \Theta\ ;\ \mathcal{B}\ ;\ \Gamma'[x::=v]_{\Gamma v}\ @\ \Gamma \vdash\ v1[x::=v]_{vv} \Rightarrow \{\!|\ z1\ :\ B\text{-}bitvec\ |\ c1[x::=v]_{cv}\ |\!\}\rangle$
using *infer-e-splitI subst-tv.simps fresh-Pair* **by** *metis*

have $\langle x \neq z2\ \rangle$ **using** *infer-e-splitI* **by** *force*

have $(\{\!|\ z2\ :\ B\text{-}int\ |\ ([\ leq\ [\ [\ L\text{-}num\ 0\ ]^v\ ]^{ce}\ [\ [\ z2\ ]^v\ ]^{ce}\ ]^{ce}\ ==\ [\ [\ L\text{-}true\ ]^v\ ]^{ce})$
$AND\ \ ([\ leq\ [\ [\ z2\ ]^v\ ]^{ce}\ [|\ [\ v1[x::=v]_{vv}\ ]^{ce}\ |]^{ce}\ ]^{ce}\ ==\ [\ [\ L\text{-}true\ ]^v\ ]^{ce}\ )\ |\!\}) =$
$(\{\!|\ z2\ :\ B\text{-}int\ |\ ([\ leq\ [\ [\ L\text{-}num\ \ 0\ ]^v\ ]^{ce}\ [\ [\ z2\ ]^v\ ]^{ce}\ ]^{ce}\ ==\ [\ [\ L\text{-}true\ ]^v\ ]^{ce}\ )$
$AND\ \ ([\ leq\ [\ [\ z2\ ]^v\ ]^{ce}\ [|\ [\ v1\ ]^{ce}\ |]^{ce}\ ]^{ce}\ ==\ [\ [\ L\text{-}true\ ]^v\ ]^{ce}\ )\ |\!\}[x::=v]_{\tau v})$
unfolding *subst-cv.simps subst-cev.simps subst-vv.simps* **using** $\langle x \neq z2\rangle$ *infer-e-splitI fresh-Pair* **by** *simp*

thus $\langle\Theta\ ;\ \mathcal{B}\ ;\ \Gamma'[x::=v]_{\Gamma v}\ @$
$\Gamma\ \vdash\ v2[x::=v]_{vv} \Leftarrow \{\!|\ z2\ :\ B\text{-}int\ |\ [\ leq\ [\ [\ L\text{-}num\ 0\ ]^v\ ]^{ce}\ [\ [\ z2\ ]^v\ ]^{ce}\ ]^{ce}\ ==\ [\ [\ L\text{-}true\ ]^v\ ]^{ce}$
$AND\ \ [\ leq\ [\ [\ z2\ ]^v\ ]^{ce}\ [|\ [\ v1[x::=v]_{vv}\ ]^{ce}\ |]^{ce}\ ]^{ce}\ ==\ [\ [\ L\text{-}true\ ]^v\ ]^{ce}\ \ |\!\}\rangle$
using *infer-e-splitI subst-infer-check-v fresh-Pair* **by** *metis*

show $\langle atom\ z1\ \sharp\ AE\text{-}split\ v1[x::=v]_{vv}\ v2[x::=v]_{vv}\rangle$ **using** *infer-e-splitI fresh-subst-vv-if* **by** *auto*
show $\langle atom\ z2\ \sharp\ AE\text{-}split\ v1[x::=v]_{vv}\ v2[x::=v]_{vv}\rangle$ **using** *infer-e-splitI fresh-subst-vv-if* **by** *auto*
show $\langle atom\ z3\ \sharp\ AE\text{-}split\ v1[x::=v]_{vv}\ v2[x::=v]_{vv}\rangle$ **using** *infer-e-splitI fresh-subst-vv-if* **by** *auto*

show $\langle atom\ z3\ \sharp\ \Gamma'[x::=v]_{\Gamma v}\ @\ \Gamma\rangle$ **using** *fresh-subst-gv-inside infer-e-splitI* **by** *metis*
show $\langle atom\ z2\ \sharp\ \Gamma'[x::=v]_{\Gamma v}\ @\ \Gamma\rangle$ **using** *fresh-subst-gv-inside infer-e-splitI* **by** *metis*
show $\langle atom\ z1\ \sharp\ \Gamma'[x::=v]_{\Gamma v}\ @\ \Gamma\rangle$ **using** *fresh-subst-gv-inside infer-e-splitI* **by** *metis*

qed

thus *?case* **apply** (*subst subst-tv.simps*)
using *infer-e-splitI fresh-Pair* **apply** *metis*
unfolding *subst-tv.simps subst-ev.simps subst-cv.simps subst-cev.simps subst-vv.simps* $*$
using $\langle x \neq z3\rangle$ **by** *simp*

qed

lemma *infer-e-uniqueness*:
assumes $\Theta\ ;\ \Phi\ ;\ \mathcal{B}\ ;\ GNil\ ;\ \Delta\ \vdash\ e_1 \Rightarrow \tau_1$ **and** $\Theta\ ;\ \Phi\ ;\ \mathcal{B}\ ;\ GNil\ ;\ \Delta\ \vdash\ e_1 \Rightarrow \tau_2$
shows $\tau_1 = \tau_2$
using *assms* **proof**(*nominal-induct rule*:*e.strong-induct*)
case (*AE-val x*)
then show *?case* **using** *infer-e-elims*(*7*)[*OF AE-val*(*1*)] *infer-e-elims*(*7*)[*OF AE-val*(*2*)] *infer-v-uniqueness* **by** *metis*
next
case (*AE-app f v*)

obtain *x1 b1 c1 s1' τ1'* **where** *t1*: *Some* (*AF-fundef f* (*AF-fun-typ-none* (*AF-fun-typ x1 b1 c1 τ1' s1'*))) = *lookup-fun* Φ *f* $\wedge$ $\tau_1 = \tau 1'[x1::=v]_{\tau v}$ **using** *infer-e-app2E*[*OF AE-app*(*1*)] **by** *metis*
moreover obtain *x2 b2 c2 s2' τ2'* **where** *t2*: *Some* (*AF-fundef f* (*AF-fun-typ-none* (*AF-fun-typ x2 b2 c2 τ2' s2'*))) = *lookup-fun* Φ *f* $\wedge$ $\tau_2 = \tau 2'[x2::=v]_{\tau v}$ **using** *infer-e-app2E*[*OF AE-app*(*2*)] **by**

metis

have $\tau 1'[x1::=v]_{\tau v} = \tau 2'[x2::=v]_{\tau v}$ **using** *t1* **and** *t2* *fun-ret-unique* **by** *metis*
thus *?thesis* **using** *t1 t2* **by** *auto*
next
case (*AE-appP f b v*)
obtain *bv1 x1 b1 c1 s1' τ1'* **where** *t1*: *Some* (*AF-fundef f* (*AF-fun-typ-some bv1* (*AF-fun-typ x1 b1 c1 τ1' s1'*))) = *lookup-fun* Φ *f* ∧ $\tau_1 = \tau 1'[bv1::=b]_{\tau b}[x1::=v]_{\tau v}$ **using** *infer-e-appP2E*[*OF AE-appP*(*1*)] **by** *metis*
moreover obtain *bv2 x2 b2 c2 s2' τ2'* **where** *t2*: *Some* (*AF-fundef f* (*AF-fun-typ-some bv2* (*AF-fun-typ x2 b2 c2 τ2' s2'*))) = *lookup-fun* Φ *f* ∧ $\tau_2 = \tau 2'[bv2::=b]_{\tau b}[x2::=v]_{\tau v}$ **using** *infer-e-appP2E*[*OF AE-appP*(*2*)] **by** *metis*

have $\tau 1'[bv1::=b]_{\tau b}[x1::=v]_{\tau v} = \tau 2'[bv2::=b]_{\tau b}[x2::=v]_{\tau v}$ **using** *t1* **and** *t2* *fun-poly-ret-unique* **by** *metis*
thus *?thesis* **using** *t1 t2* **by** *auto*
next
case (*AE-op opp v1 v2*)
show *?case* **proof**(*cases opp=Plus*)
case *True*
hence Θ ; Φ ; $\mathcal{B}$; *GNil* ; Δ ⊢ *AE-op Plus v1 v2* ⇒ τ_1 **and** Θ ; Φ ; $\mathcal{B}$; *GNil* ; Δ ⊢ *AE-op Plus v1 v2* ⇒ τ_2 **using** *AE-op* **by** *auto*
thm *infer-e-elims*(*3*)
thus *?thesis* **using** *infer-e-elims*(*11*)[*OF* ‹Θ ; Φ ; $\mathcal{B}$; *GNil* ; Δ ⊢ *AE-op Plus v1 v2* ⇒ τ_1›] *infer-e-elims*(*11*)[*OF* ‹Θ ; Φ ; $\mathcal{B}$; *GNil* ; Δ ⊢ *AE-op Plus v1 v2* ⇒ τ_2›]
by *force*
next
case *False*
hence *opp* = *LEq* **using** *opp.strong-exhaust* **by** *auto*
hence Θ ; Φ ; $\mathcal{B}$; *GNil* ; Δ ⊢ *AE-op LEq v1 v2* ⇒ τ_1 **and** Θ ; Φ ; $\mathcal{B}$; *GNil* ; Δ ⊢ *AE-op LEq v1 v2* ⇒ τ_2 **using** *AE-op* **by** *auto*
thm *infer-e-elims*(*3*)
thus *?thesis* **using** *infer-e-elims*(*12*)[*OF* ‹Θ ; Φ ; $\mathcal{B}$; *GNil* ; Δ ⊢ *AE-op LEq v1 v2* ⇒ τ_1›] *infer-e-elims*(*12*)[*OF* ‹Θ ; Φ ; $\mathcal{B}$; *GNil* ; Δ ⊢ *AE-op LEq v1 v2* ⇒ τ_2›]
by *force*
qed
next
case (*AE-concat v1 v2*)

obtain *z3*::*x* **where** *t1*:τ_1 = {‖ *z3* : *B-bitvec* | [[*z3*]^v]^{ce} == *CE-concat* [*v1*]^{ce} [*v2*]^{ce} ‖} ∧ *atom z3* ♯ *v1* ∧ *atom z3* ♯ *v2* **using** *infer-e-elims*(*18*)[*OF AE-concat*(*1*)] **by** *metis*
obtain *z3'*::*x* **where** *t2*:τ_2 = {‖ *z3'* : *B-bitvec* | [[*z3'*]^v]^{ce} == *CE-concat* [*v1*]^{ce} [*v2*]^{ce} ‖} ∧ *atom z3'* ♯ *v1* ∧ *atom z3'* ♯ *v2* **using** *infer-e-elims*(*18*)[*OF AE-concat*(*2*)] **by** *metis*

thus *?case* **using** *t1 t2 type-e-eq ce.fresh* **by** *metis*

next
case (*AE-fst v*)

obtain *z1* **and** *b1* **where** τ_1 = {‖ *z1* : *b1* | *CE-val* (*V-var z1*) == (*CE-fst* [*v*]^{ce}) ‖} **using** *infer-v-form AE-fst* **by** *auto*

obtain $xx :: x$ **and** $bb :: b$ **and** $xxa :: x$ **and** $bba :: b$ **and** $cc :: c$ **where**
$f1$: $\tau_2 = \{\!|\ xx : bb \mid CE\text{-}val\ (V\text{-}var\ xx) == CE\text{-}fst\ [v]^{ce}\ |\!\} \wedge \Theta\ ;\ \mathcal{B}\ ;\ GNil \vdash_{wf} \Delta \wedge \Theta\ ;\ \mathcal{B}\ ;\ GNil \vdash v \Rightarrow \{\!|\ xxa : B\text{-}pair\ bb\ bba \mid cc\ |\!\} \wedge atom\ xx \sharp v$
using *infer-e-elims(8)[OF AE-fst(2)]* **by** *metis*
obtain $xxb :: x$ **and** $bbb :: b$ **and** $xxc :: x$ **and** $bbc :: b$ **and** $cca :: c$ **where**
$f2$: $\tau_1 = \{\!|\ xxb : bbb \mid CE\text{-}val\ (V\text{-}var\ xxb) == CE\text{-}fst\ [v]^{ce}\ |\!\} \wedge \Theta\ ;\ \mathcal{B}\ ;\ GNil \vdash_{wf} \Delta \wedge \Theta\ ;\ \mathcal{B}\ ;\ GNil \vdash v \Rightarrow \{\!|\ xxc : B\text{-}pair\ bbb\ bbc \mid cca\ |\!\} \wedge atom\ xxb \sharp v$
using *infer-e-elims(8)[OF AE-fst(1)]* **by** *metis*
then have $B\text{-}pair\ bb\ bba = B\text{-}pair\ bbb\ bbc$
using *f1* **by** (*metis* (*no-types*) *b-of.simps infer-v-uniqueness*)
then have $\{\!|\ xx : bbb \mid CE\text{-}val\ (V\text{-}var\ xx) == CE\text{-}fst\ [v]^{ce}\ |\!\} = \tau_2$
using *f1* **by** *auto*
then show *?thesis*
using *f2* **by** (*meson ce.fresh fresh-GNil type-e-eq wfG-x-fresh-in-v-simple*)
next
case (*AE-snd v*)
obtain $xx :: x$ **and** $bb :: b$ **and** $xxa :: x$ **and** $bba :: b$ **and** $cc :: c$ **where**
$f1$: $\tau_2 = \{\!|\ xx : bba \mid CE\text{-}val\ (V\text{-}var\ xx) == CE\text{-}snd\ [v]^{ce}\ |\!\} \wedge \Theta\ ;\ \mathcal{B}\ ;\ GNil \vdash_{wf} \Delta \wedge \Theta\ ;\ \mathcal{B}\ ;\ GNil \vdash v \Rightarrow \{\!|\ xxa : B\text{-}pair\ bb\ bba \mid cc\ |\!\} \wedge atom\ xx \sharp v$
using *infer-e-elims(9)[OF AE-snd(2)]* **by** *metis*
obtain $xxb :: x$ **and** $bbb :: b$ **and** $xxc :: x$ **and** $bbc :: b$ **and** $cca :: c$ **where**
$f2$: $\tau_1 = \{\!|\ xxb : bbc \mid CE\text{-}val\ (V\text{-}var\ xxb) == CE\text{-}snd\ [v]^{ce}\ |\!\} \wedge \Theta\ ;\ \mathcal{B}\ ;\ GNil \vdash_{wf} \Delta \wedge \Theta\ ;\ \mathcal{B}\ ;\ GNil \vdash v \Rightarrow \{\!|\ xxc : B\text{-}pair\ bbb\ bbc \mid cca\ |\!\} \wedge atom\ xxb \sharp v$
using *infer-e-elims(9)[OF AE-snd(1)]* **by** *metis*
then have $B\text{-}pair\ bb\ bba = B\text{-}pair\ bbb\ bbc$
using *f1* **by** (*metis* (*no-types*) *b-of.simps infer-v-uniqueness*)
then have $\{\!|\ xx : bbc \mid CE\text{-}val\ (V\text{-}var\ xx) == CE\text{-}snd\ [v]^{ce}\ |\!\} = \tau_2$
using *f1* **by** *auto*
then show *?thesis*
using *f2* **by** (*meson ce.fresh fresh-GNil type-e-eq wfG-x-fresh-in-v-simple*)
next
case (*AE-mvar x*)
then show *?case* **using** *infer-e-elims(10)[OF AE-mvar(1)] infer-e-elims(10)[OF AE-mvar(2)] wfD-unique* **by** *metis*
next
case (*AE-len x*)
then show *?case* **using** *infer-e-elims(16)[OF AE-len(1)] infer-e-elims(16)[OF AE-len(2)]* **by** *force*
next
case (*AE-split x1a x2*)
then show *?case* **using** *infer-e-elims(22)[OF AE-split(1)] infer-e-elims(22)[OF AE-split(2)]* **by** *force*
qed

14.8 Statements

method *subst-mth* = (*metis subst-g-inside infer-e-wf infer-v-wf infer-v-wf*)

lemma *subst-infer-check-v1*:
fixes $v{::}v$ **and** $v'{::}v$ **and** $\Gamma{::}\Gamma$
assumes $\Gamma = \Gamma_1@((x,b_1,c0[z0::=[x]^v]_{cv})\#_\Gamma\Gamma_2)$ **and**
$\Theta\ ;\ \mathcal{B}\ ;\ \Gamma_2 \vdash v \Rightarrow \tau_1$ **and**
$\Theta\ ;\ \mathcal{B}\ ;\ \Gamma \vdash v' \Leftarrow \tau_2$ **and**
$\Theta\ ;\ \mathcal{B}\ ;\ \Gamma_2 \vdash \tau_1 \lesssim \{\!|\ z0 : b_1 \mid c0\ |\!\}$ **and** $atom\ z0 \sharp (x,v)$

shows Θ ; $\mathcal{B}$; $\Gamma[x::=v]_{\Gamma v} \vdash v'[x::=v]_{vv} \Leftarrow \tau_2[x::=v]_{\tau v}$
using *subst-g-inside check-v-wf assms subst-infer-check-v* **by** *metis*

method *subst-tuple-mth* **uses** *add* = (
(*unfold fresh-prodN*), (*simp add*: *add*)+,
(*rule*,*metis fresh-z-subst-g add fresh-Pair*),
(*metis fresh-subst-dv add fresh-Pair*))

thm *subst-valid-simple*

lemma *infer-v-c-valid*:
assumes Θ ; $\mathcal{B}$; $\Gamma \vdash v \Rightarrow \tau$ **and** Θ ; $\mathcal{B}$; $\Gamma \vdash \tau \lesssim \{\!|\, z : b \mid c \,|\!\}$
shows $\langle \Theta$; $\mathcal{B}$; $\Gamma \models c[z::=v]_{cv} \rangle$
proof −
obtain *z1* **and** *b1* **and** *c1* **where** $*{:}\tau = \{\!|\, z1 : b1 \mid c1 \,|\!\} \wedge$ *atom z1* $\sharp$ (*c*,*v*,Γ) **using** *obtain-fresh-z* **by** *metis*
then have *b1* = *b* **using** *assms subtype-eq-base* **by** *metis*
moreover then have Θ ; $\mathcal{B}$; $\Gamma \vdash v \Rightarrow \{\!|\, z1 : b \mid c1 \,|\!\}$ **using** *assms* $*$ **by** *auto*

moreover have Θ ; $\mathcal{B}$; (*z1*, *b*, *c1*) $\#_\Gamma$ $\Gamma \models c[z::=[\ z1\ ]^v]_{cv}$ **proof** −
have Θ ; $\mathcal{B}$; (*z1*, *b*, *c1*$[z1::=[\ z1\ ]^v]_v$) $\#_\Gamma$ $\Gamma \models c[z::=[\ z1\ ]^v]_v$
using *subtype-valid*[*OF assms*(*2*), *of z1 z1 b c1 z c*] $*$ *fresh-prodN* $\langle b1 = b \rangle$ **by** *metis*
moreover have *c1*$[z1::=[\ z1\ ]^v]_v$ = *c1* **using** *subst-v-v-def* **by** *simp*
ultimately show *?thesis* **using** *subst-v-c-def* **by** *metis*
qed
ultimately show *?thesis* **using** $*$ *fresh-prodN subst-valid-simple* **by** *metis*
qed

Substitution Lemma for Statements

lemma *subst-infer-check-s*:
fixes *v*::*v* **and** *s*::*s* **and** *cs*::*branch-s* **and** *x*::*x* **and** *c*::*c* **and** *b*::*b* **and**
Γ_1::Γ **and** Γ_2::Γ **and** *css*::*branch-list*
assumes Θ ; $\mathcal{B}$; $\Gamma_1 \vdash v \Rightarrow \tau$ **and** Θ ; $\mathcal{B}$; $\Gamma_1 \vdash \tau \lesssim \{\!|\, z : b \mid c \,|\!\}$ **and**
atom z $\sharp$ (*x*, *v*)
shows Θ ; Φ ; $\mathcal{B}$; Γ; Δ $\vdash s \Leftarrow \tau' \Longrightarrow$
$\Gamma = (\Gamma_2@((x,b,c[z::=[x]^v]_{cv})\#_\Gamma\Gamma_1)) \Longrightarrow$
Θ ; Φ ; $\mathcal{B}$; $\Gamma[x::=v]_{\Gamma v}$; $\Delta[x::=v]_{\Delta v} \vdash s[x::=v]_{sv} \Leftarrow \tau'[x::=v]_{\tau v}$
and
Θ ; Φ ; $\mathcal{B}$; Γ; Δ; *tid* ; *cons* ; *const* ; $v' \vdash cs \Leftarrow \tau' \Longrightarrow$
$\Gamma = (\Gamma_2@((x,b,c[z::=[x]^v]_{cv})\#_\Gamma\Gamma_1)) \Longrightarrow$
Θ ; Φ ; $\mathcal{B}$; $\Gamma[x::=v]_{\Gamma v}$; $\Delta[x::=v]_{\Delta v}$;
tid ; *cons* ; *const* ; $v'[x::=v]_{vv} \vdash cs[x::=v]_{sv} \Leftarrow \tau'[x::=v]_{\tau v}$
and
Θ ; Φ ; $\mathcal{B}$; Γ; Δ; *tid* ; *dclist* ; $v' \vdash css \Leftarrow \tau' \Longrightarrow$
$\Gamma = (\Gamma_2@((x,b,c[z::=[x]^v]_{cv})\#_\Gamma\Gamma_1)) \Longrightarrow$
Θ ; Φ ; $\mathcal{B}$; $\Gamma[x::=v]_{\Gamma v}$; $\Delta[x::=v]_{\Delta v}$; *tid* ; *dclist* ; $v'[x::=v]_{vv} \vdash$
subst-branchlv css x v $\Leftarrow \tau'[x::=v]_{\tau v}$

using *assms* **proof**(*nominal-induct* τ' **and** τ' **and** τ' *avoiding*: *x v arbitrary*: Γ_2 **and** Γ_2 **and** Γ_2
rule: *check-s-check-branch-s-check-branch-list.strong-induct*)
case (*check-valI* Θ $\mathcal{B}$ Γ Δ Φ v' τ' τ'')

have *sg*: $\Gamma[x::=v]_{\Gamma v} = \Gamma_2[x::=v]_{\Gamma v}@\Gamma_1$ **using** *check-valI* **by** *subst-mth*
thm *wf-subst(12)*
have $\Theta\ ;\ \Phi\ ;\ \mathcal{B}\ ;\ \Gamma[x::=v]_{\Gamma v}\ ;\ \Delta[x::=v]_{\Delta v}\ \vdash\ (AS\text{-}val\ (v'[x::=v]_{vv})) \Leftarrow \tau''[x::=v]_{\tau v}$ **proof**
have $\Theta\ ;\ \mathcal{B}\ ;\ \Gamma_1 \vdash_{wf} v : b$ **using** *infer-v-v-wf subtype-eq-base2 b-of.simps check-valI* **by** *metis*
thus $\langle \Theta\ ;\ \mathcal{B}\ ;\ \Gamma[x::=v]_{\Gamma v} \vdash_{wf} \Delta[x::=v]_{\Delta v} \rangle$ **using** *wf-subst(15) check-valI* **by** *auto*
show $\langle\ \Theta \vdash_{wf} \Phi\ \rangle$ **using** *check-valI* **by** *auto*
show $\langle\ \Theta\ ;\ \mathcal{B}\ ;\ \Gamma[x::=v]_{\Gamma v}\ \vdash\ v'[x::=v]_{vv} \Rightarrow \tau'[x::=v]_{\tau v} \rangle$ **proof**(*subst sg*, *rule subst-infer-v*)
show $\Theta\ ;\ \mathcal{B}\ ;\ \Gamma_1 \vdash v \Rightarrow \tau$ **using** *check-valI* **by** *auto*
show $\Theta\ ;\ \mathcal{B}\ ;\ \Gamma_2\ @\ (x,\ b,\ c[z::=[x]^v]_{cv})\ \#_\Gamma\ \Gamma_1 \vdash v' \Rightarrow \tau'$ **using** *check-valI* **by** *metis*
show $\Theta\ ;\ \mathcal{B}\ ;\ \Gamma_1\ \vdash \tau \lesssim \{\!|\ z{:}\ b\ \ |\ c\ |\!\}$ **using** *check-valI* **by** *auto*
show *atom z* $\sharp$ $(x,\ v)$ **using** *check-valI* **by** *auto*
qed
show $\langle \Theta\ ;\ \mathcal{B}\ ;\ \Gamma[x::=v]_{\Gamma v}\ \vdash\ \tau'[x::=v]_{\tau v} \lesssim \tau''[x::=v]_{\tau v} \rangle$ **using** *subst-subtype-tau check-valI sg* **by** *metis*
qed

thus *?case* **using** *Typing.check-valI subst-sv.simps sg* **by** *auto*
next
case (*check-letI xa* Θ Φ $\mathcal{B}$ Γ Δ *ea* τa *za sa ba ca*)
have $*$:(*AS-let xa ea sa*)$[x::=v]_{sv}$=(*AS-let xa* $(ea[x::=v]_{ev})$ $sa[x::=v]_{sv}$)
using *subst-sv.simps* $\langle$ *atom xa* $\sharp$ *x*$\rangle$ $\langle$ *atom xa* $\sharp$ *v*$\rangle$ **by** *auto*
show *?case* **unfolding** $*$ **proof**

show *atom xa* $\sharp$ $(\Theta,\Phi,\mathcal{B},\Gamma[x::=v]_{\Gamma v},\Delta[x::=v]_{\Delta v},ea[x::=v]_{ev},\tau a[x::=v]_{\tau v})$
by(*subst-tuple-mth add*: *check-letI*)

show *atom za* $\sharp$ $(xa,\Theta,\Phi,\mathcal{B},\Gamma[x::=v]_{\Gamma v},\ \Delta[x::=v]_{\Delta v},ea[x::=v]_{ev},$
$\tau a[x::=v]_{\tau v},sa[x::=v]_{sv})$
by(*subst-tuple-mth add*: *check-letI*)

show $\Theta;\ \Phi;\ \mathcal{B};\ \Gamma[x::=v]_{\Gamma v};\ \Delta[x::=v]_{\Delta v} \vdash$
$ea[x::=v]_{ev} \Rightarrow\ \{\!|\ za : ba \mid ca[x::=v]_{cv}\ |\!\}$
proof −
have $\Theta;\ \Phi;\ \mathcal{B};\ \Gamma_2[x::=v]_{\Gamma v}\ @\ \Gamma_1;\ \Delta[x::=v]_{\Delta v} \vdash$
$ea[x::=v]_{ev} \Rightarrow \{\!|\ za : ba \mid ca\ |\!\}[x::=v]_{\tau v}$
using *check-letI subst-infer-e* **by** *metis*
thus *?thesis* **using** *check-letI subst-tv.simps*
by (*metis fresh-prod2I infer-e-wf subst-g-inside-simple*)
qed

show $\Theta;\ \Phi;\ \mathcal{B};\ (xa,\ ba,\ ca[x::=v]_{cv}[za::=V\text{-}var\ xa]_v)\ \#_\Gamma\ \Gamma[x::=v]_{\Gamma v};$
$\Delta[x::=v]_{\Delta v}\ \vdash sa[x::=v]_{sv} \Leftarrow \tau a[x::=v]_{\tau v}$
proof −
have $\Theta;\ \Phi;\ \mathcal{B};\ ((xa,\ ba,\ ca[za::=V\text{-}var\ xa]_v)\ \#_\Gamma\ \Gamma)[x::=v]_{\Gamma v}\ ;$
$\Delta[x::=v]_{\Delta v}\ \vdash\ sa[x::=v]_{sv} \Leftarrow \tau a[x::=v]_{\tau v}$
apply(*rule check-letI*(*23*)[*of* $(xa,\ ba,\ ca[za::=V\text{-}var\ xa]_{cv})\ \#_\Gamma\ \Gamma_2$])
by(*metis check-letI append-g.simps subst-defs*)+

moreover have $(xa,\ ba,\ ca[x::=v]_{cv}[za::=V\text{-}var\ xa]_{cv})\ \#_\Gamma\ \Gamma[x::=v]_{\Gamma v} =$
$((xa,\ ba,\ ca[za::=V\text{-}var\ xa]_{cv})\ \#_\Gamma\ \Gamma)[x::=v]_{\Gamma v}$
using *subst-cv-commute subst-gv.simps check-letI*
by (*metis ms-fresh-all*(*39*) *ms-fresh-all*(*49*) *subst-cv-commute-subst*)

ultimately show *?thesis*
using *subst-defs* **by** *auto*
qed
qed
next
case (*check-assertI xa* Θ Φ $\mathcal{B}$ Γ Δ *ca* τ *s*)
show *?case* **unfolding** *subst-sv.simps* **proof**
show ‹*atom xa* ♯ (Θ, Φ, $\mathcal{B}$, $\Gamma[x::=v]_{\Gamma v}$, $\Delta[x::=v]_{\Delta v}$, *ca*$[x::=v]_{cv}$, $\tau[x::=v]_{\tau v}$, *s*$[x::=v]_{sv}$)›
by(*subst-tuple-mth add*: *check-assertI*)
have *xa* ≠ *x* **using** *check-assertI* **by** *fastforce*
thus ‹ Θ ; Φ ; $\mathcal{B}$; (*xa*, *B-bool*, *ca*$[x::=v]_{cv}$) #$_\Gamma$ $\Gamma[x::=v]_{\Gamma v}$; $\Delta[x::=v]_{\Delta v}$ ⊢ *s*$[x::=v]_{sv}$ ⇐ $\tau[x::=v]_{\tau v}$›

using *check-assertI*(*12*)[*of* (*xa*, *B-bool*, *c*) #$_\Gamma$ Γ_2 *x v*] *check-assertI subst-gv.simps append-g.simps*
by *metis*
have ‹Θ ; $\mathcal{B}$; $\Gamma_2[x::=v]_{\Gamma v}$ @ Γ_1 ⊨ *ca*$[x::=v]_{cv}$ › **proof**(*rule subst-valid*)
show ‹Θ ; $\mathcal{B}$; Γ_1 ⊨ *c*$[z::=v]_{cv}$ › **using** *infer-v-c-valid check-assertI* **by** *metis*
show ‹ Θ ; $\mathcal{B}$; Γ_1 ⊢$_{wf}$ *v* : *b* › **using** *check-assertI infer-v-wf b-of.simps subtype-eq-base*
by (*metis subtype-eq-base2*)
show ‹ Θ ; $\mathcal{B}$ ⊢$_{wf}$ Γ_1 › **using** *check-assertI infer-v-wf* **by** *metis*
have Θ ; $\mathcal{B}$ ⊢$_{wf}$ Γ_2 @ (*x*, *b*, *c*$[z::=[\ x\ ]^v]_{cv}$) #$_\Gamma$ Γ_1 **using** *check-assertI wfX-wfY* **by** *metis*
thus ‹*atom x* ♯ Γ_1› **using** *check-assertI wfG-suffix wfG-elims* **by** *metis*

moreover have Θ ; $\mathcal{B}$; Γ_1 ⊢$_{wf}$ {| *z* : *b* | *c* |} **using** *subtype-wfT check-assertI* **by** *metis*

moreover have *x* ≠ *z* **using** *fresh-Pair check-assertI fresh-x-neq* **by** *metis*
ultimately show ‹*atom x* ♯ *c*› **using** *check-assertI wfT-fresh-c* **by** *metis*

show ‹ Θ ; $\mathcal{B}$ ⊢$_{wf}$ Γ_2 @ (*x*, *b*, *c*$[z::=[\ x\ ]^v]_{cv}$) #$_\Gamma$ Γ_1 › **using** *check-assertI wfX-wfY* **by** *metis*
show ‹Θ ; $\mathcal{B}$; Γ_2 @ (*x*, *b*, *c*$[z::=[\ x\ ]^v]_{cv}$) #$_\Gamma$ Γ_1 ⊨ *ca* › **using** *check-assertI* **by** *auto*
qed
thus ‹Θ ; $\mathcal{B}$; $\Gamma[x::=v]_{\Gamma v}$ ⊨ *ca*$[x::=v]_{cv}$ › **using** *check-assertI*
proof −
show *?thesis*
by (*metis* (*no-types*) ‹Γ = Γ_2 @ (*x*, *b*, *c*$[z::=[\ x\ ]^v]_{cv}$) #$_\Gamma$ Γ_1› ‹Θ ; $\mathcal{B}$; Γ ⊨ *ca*› ‹Θ ; $\mathcal{B}$; $\Gamma_2[x::=v]_{\Gamma v}$ @ Γ_1 ⊨ *ca*$[x::=v]_{cv}$› *subst-g-inside valid-g-wf*)
qed

have Θ ; $\mathcal{B}$; Γ_1 ⊢$_{wf}$ *v* : *b* **using** *infer-v-wf b-of.simps check-assertI*
by (*metis subtype-eq-base2*)
thus ‹ Θ ; $\mathcal{B}$; $\Gamma[x::=v]_{\Gamma v}$ ⊢$_{wf}$ $\Delta[x::=v]_{\Delta v}$ › **using** *wf-subst2*(*6*) *check-assertI* **by** *metis*
qed
next
case (*check-branch-list-consI* Θ Φ $\mathcal{B}$ Γ Δ *tid dclist vv cs* τ *css*)
show *?case* **unfolding** ∗ **using** *subst-sv.simps check-branch-list-consI* **by** *simp*
next
case (*check-branch-list-finalI* Θ Φ $\mathcal{B}$ Γ Δ *tid dclist v cs* τ)
show *?case* **unfolding** ∗ **using** *subst-sv.simps check-branch-list-finalI* **by** *simp*
next
case (*check-branch-s-branchI* Θ $\mathcal{B}$ Γ Δ τ *const xa* Φ *tid cons va sa*)
hence ∗:(*AS-branch cons xa sa*)$[x::=v]_{sv}$ = (*AS-branch cons xa sa*$[x::=v]_{sv}$) **using** *subst-branchv.simps fresh-Pair* **by** *metis*

show *?case* **unfolding** $*$ **proof**

show Θ ; $\mathcal{B}$; $\Gamma[x::=v]_{\Gamma v} \vdash_{wf} \Delta[x::=v]_{\Delta v}$
using *wf-subst check-branch-s-branchI subtype-eq-base2 b-of.simps infer-v-wf* **by** *metis*

show $\vdash_{wf} \Theta$ **using** *check-branch-s-branchI* **by** *metis*

show Θ ; $\mathcal{B}$; $\Gamma[x::=v]_{\Gamma v} \vdash_{wf} \tau[x::=v]_{\tau v}$
using *wf-subst check-branch-s-branchI subtype-eq-base2 b-of.simps infer-v-wf* **by** *metis*

show *wft*:Θ ; {||} ; *GNil*$\vdash_{wf}$ *const* **using** *check-branch-s-branchI* **by** *metis*

show *atom xa* $\sharp$ (Θ, Φ, $\mathcal{B}$, $\Gamma[x::=v]_{\Gamma v}$, $\Delta[x::=v]_{\Delta v}$, *tid*, *cons*, *const*,$va[x::=v]_{vv}$, $\tau[x::=v]_{\tau v}$)
apply(*unfold fresh-prodN*, (*simp add*: *check-branch-s-branchI*)+)
apply(*rule*,*metis fresh-z-subst-g check-branch-s-branchI fresh-Pair*)
by(*metis fresh-subst-dv check-branch-s-branchI fresh-Pair*)

have Θ ; Φ ; $\mathcal{B}$; ((*xa*, *b-of const*, *CE-val va* == *CE-val*(*V-cons tid cons* (*V-var xa*)) *AND c-of const xa*) $\#_\Gamma$ Γ)$[x::=v]_{\Gamma v}$; $\Delta[x::=v]_{\Delta v}$ $\vdash$ $sa[x::=v]_{sv} \Leftarrow \tau[x::=v]_{\tau v}$
using *check-branch-s-branchI* **by** (*metis append-g.simps*(*2*))

moreover have (*xa*, *b-of const*, *CE-val* $va[x::=v]_{vv}$ == *CE-val* (*V-cons tid cons* (*V-var xa*)) *AND c-of* (*const*) *xa*) $\#_\Gamma$ $\Gamma[x::=v]_{\Gamma v}$ =
((*xa*, *b-of const* , *CE-val va* == *CE-val* (*V-cons tid cons* (*V-var xa*)) *AND c-of const xa*) $\#_\Gamma$ Γ)$[x::=v]_{\Gamma v}$
proof −
have $*$:$xa \neq x$ **using** *check-branch-s-branchI fresh-at-base* **by** *metis*
have *atom x* $\sharp$ *const* **using** *wfT-nil-supp*[*OF wft*] *fresh-def* **by** *auto*
hence *atom x* $\sharp$ (*const*,*xa*) **using** *fresh-at-base wfT-nil-supp*[*OF wft*] *fresh-Pair fresh-def* $*$ **by** *auto*
moreover hence (*c-of* (*const*) *xa*)$[x::=v]_{cv}$ = *c-of* (*const*) *xa*
using *c-of-fresh*[*of x const xa*] *forget-subst-cv wfT-nil-supp wft* **by** *metis*
moreover hence (*V-cons tid cons* (*V-var xa*))$[x::=v]_{vv}$ = (*V-cons tid cons* (*V-var xa*)) **using** *check-branch-s-branchI subst-vv.simps* $*$ **by** *metis*
ultimately show *?thesis* **using** *subst-gv.simps check-branch-s-branchI subst-cv.simps subst-cev.simps* $*$ **by** *presburger*
qed

ultimately show Θ ; Φ ; $\mathcal{B}$; (*xa*, *b-of const*, *CE-val* $va[x::=v]_{vv}$ == *CE-val* (*V-cons tid cons* (*V-var xa*)) *AND c-of const xa*) $\#_\Gamma$ $\Gamma[x::=v]_{\Gamma v}$; $\Delta[x::=v]_{\Delta v}$ $\vdash$ $sa[x::=v]_{sv} \Leftarrow \tau[x::=v]_{\tau v}$
by *metis*
qed

next
case (*check-let2I xa* Θ Φ $\mathcal{B}$ *G* Δ *t s1* τa *s2*)
hence $*$:(*AS-let2 xa t s1 s2*)$[x::=v]_{sv}$ = (*AS-let2 xa* $t[x::=v]_{\tau v}$ ($s1[x::=v]_{sv}$) $s2[x::=v]_{sv}$) **using** *subst-sv.simps fresh-Pair* **by** *metis*
have $xa \neq x$ **using** *check-let2I fresh-at-base* **by** *metis*
show *?case* **unfolding** $*$ **proof**
show *atom xa* $\sharp$ (Θ, Φ, $\mathcal{B}$, $G[x::=v]_{\Gamma v}$, $\Delta[x::=v]_{\Delta v}$, $t[x::=v]_{\tau v}$, $s1[x::=v]_{sv}$, $\tau a[x::=v]_{\tau v}$)
by(*subst-tuple-mth add*: *check-let2I*)
show Θ ; Φ ; $\mathcal{B}$; $G[x::=v]_{\Gamma v}$; $\Delta[x::=v]_{\Delta v}$ $\vdash$ $s1[x::=v]_{sv} \Leftarrow t[x::=v]_{\tau v}$ **using** *check-let2I* **by** *metis*

have Θ ; Φ ; $\mathcal{B}$; $((xa,\ b\text{-}of\ t,\ c\text{-}of\ t\ xa)\ \#_\Gamma\ G)[x::=v]_{\Gamma v}$; $\Delta[x::=v]_{\Delta v}$ $\vdash$ $s2[x::=v]_{sv} \Leftarrow \tau a[x::=v]_{\tau v}$
proof(*rule check-let2I*(*14*))
show ‹$(xa,\ b\text{-}of\ t,\ c\text{-}of\ t\ xa)\ \#_\Gamma\ G = (((xa,\ b\text{-}of\ t,\ c\text{-}of\ t\ xa)\#_\Gamma\ \Gamma_2))\ @\ (x,\ b,\ c[z::=[\ x\ ]^v]_{cv})\ \#_\Gamma\ \Gamma_1$›
using *check-let2I append-g.simps* **by** *metis*
show ‹Θ ; $\mathcal{B}$; $\Gamma_1 \vdash v \Rightarrow \tau$› **using** *check-let2I* **by** *metis*
show ‹Θ ; $\mathcal{B}$; $\Gamma_1\ \vdash \tau \lesssim \{\!|\ z : b\ \mid c\ |\!\}$› **using** *check-let2I* **by** *metis*
show ‹$atom\ z \sharp (x,\ v)$› **using** *check-let2I* **by** *metis*
qed
moreover have $c\text{-}of\ t[x::=v]_{\tau v}\ xa = (c\text{-}of\ t\ xa)[x::=v]_{cv}$ **using** *subst-v-c-of fresh-Pair check-let2I* **by** *metis*
moreover have $b\text{-}of\ t[x::=v]_{\tau v} = b\text{-}of\ t$ **using** *b-of.simps subst-tv.simps b-of-subst* **by** *metis*
ultimately show Θ ; Φ ; $\mathcal{B}$; $(xa,\ b\text{-}of\ t[x::=v]_{\tau v},\ c\text{-}of\ t[x::=v]_{\tau v}\ xa)\ \#_\Gamma\ G[x::=v]_{\Gamma v}$; $\Delta[x::=v]_{\Delta v}$ $\vdash$ $s2[x::=v]_{sv} \Leftarrow \tau a[x::=v]_{\tau v}$
using *check-let2I*(*14*) *subst-gv.simps* ‹$xa \neq x$› *b-of.simps* **by** *metis*
qed

next

case (*check-varI u* Θ Φ $\mathcal{B}$ Γ Δ τ' *va* τ'' *s*)
have $**$: $\Gamma[x::=v]_{\Gamma v} = \Gamma_2[x::=v]_{\Gamma v} @ \Gamma_1$ **using** *subst-g-inside check-s-wf check-varI* **by** *meson*

have Θ ; Φ ;$\mathcal{B}$; *subst-gv* Γ *x v* ; $\Delta[x::=v]_{\Delta v}$ $\vdash$ *AS-var u* $\tau'[x::=v]_{\tau v}$ $(va[x::=v]_{vv})$ $(subst\text{-}sv\ s\ x\ v) \Leftarrow \tau''[x::=v]_{\tau v}$
proof(*rule Typing.check-varI*)
show $atom\ u \sharp (\Theta,\ \Phi,\ \mathcal{B},\ \Gamma[x::=v]_{\Gamma v},\ \Delta[x::=v]_{\Delta v},\ \tau'[x::=v]_{\tau v},\ va[x::=v]_{vv},\ \tau''[x::=v]_{\tau v})$
by(*subst-tuple-mth add*: *check-varI*)
show Θ ; $\mathcal{B}$; $\Gamma[x::=v]_{\Gamma v}\ \vdash va[x::=v]_{vv} \Leftarrow \tau'[x::=v]_{\tau v}$ **using** *check-varI subst-infer-check-v* $**$ **by** *metis*
show Θ ; Φ ; $\mathcal{B}$; *subst-gv* Γ *x v* ; $(u,\ \tau'[x::=v]_{\tau v})\ \#_\Delta\ \Delta[x::=v]_{\Delta v}$ $\vdash$ $s[x::=v]_{sv} \Leftarrow \tau''[x::=v]_{\tau v}$
proof −
have *wfD* Θ $\mathcal{B}$ $(\Gamma_2\ @\ (x,\ b,\ c[z::=[x]^v]_{cv})\ \#_\Gamma\ \Gamma_1)$ $((u,\tau')\#_\Delta\ \Delta)$ **using** *check-varI check-s-wf* **by** *meson*
moreover have *wfV* Θ $\mathcal{B}$ Γ_1 *v* (*b-of* τ) **using** *infer-v-wf check-varI*(*6*) *check-varI* **by** *metis*
have *wfD* Θ $\mathcal{B}$ $(\Gamma[x::=v]_{\Gamma v})$ $((u,\tau'[x::=v]_{\tau v})\ \#_\Delta\ \Delta[x::=v]_{\Delta v})$ **proof**(*subst subst-dv.simps*(*2*)[*symmetric*], *subst* $**$, *rule wfD-subst*)
show Θ ; $\mathcal{B}$; $\Gamma_1 \vdash v \Rightarrow \tau$ **using** *check-varI* **by** *auto*
show Θ ; $\mathcal{B}$; $\Gamma_2\ @\ (x,\ b,\ c[z::=[x]^v]_{cv})\ \#_\Gamma\ \Gamma_1 \vdash_{wf} (u,\ \tau')\ \#_\Delta\ \Delta$ **using** *check-varI check-s-wf* **by** *simp*
show *b-of* $\tau = b$ **using** *check-varI subtype-eq-base2 b-of.simps* **by** *auto*
qed
thus *?thesis* **using** *check-varI* **by** *auto*
qed
qed
moreover have *atom u* $\sharp$ (x,v) **using** *u-fresh-xv* **by** *auto*
ultimately show *?case* **using** *subst-sv.simps*(*7*) **by** *auto*

next
case (*check-assignI P* Φ $\mathcal{B}$ Γ Δ *u* $\tau 1$ v' *z1* τ')

have *wfG P* $\mathcal{B}$ Γ **using** *check-v-wf check-assignI* **by** *simp*
hence *gs*: $\Gamma_2[x::=v]_{\Gamma v}\ @\ \Gamma_1 = \Gamma[x::=v]_{\Gamma v}$ **using** *subst-g-inside check-assignI* **by** *simp*

have $P\ ;\ \Phi\ ;\ \mathcal{B}\ ;\ \Gamma[x::=v]_{\Gamma v}\ ;\ \Delta[x::=v]_{\Delta v}\ \vdash$ *AS-assign* $u\ (v'[x::=v]_{vv}) \Leftarrow \tau'[x::=v]_{\tau v}$
proof(*rule Typing.check-assignI*)
show $P \vdash_{wf} \Phi$ **using** *check-assignI* **by** *auto*
show *wfD* $P\ \mathcal{B}\ (\Gamma[x::=v]_{\Gamma v})\ \Delta[x::=v]_{\Delta v}$ **using** *wf-subst*(15)[*OF check-assignI*(2)] *gs infer-v-v-wf check-assignI b-of.simps subtype-eq-base2* **by** *metis*
thus $(u,\ \tau 1[x::=v]_{\tau v}) \in$ *setD* $\Delta[x::=v]_{\Delta v}$ **using** *check-assignI subst-dv-member* **by** *metis*
thus $P\ ;\ \mathcal{B}\ ;\ \Gamma[x::=v]_{\Gamma v}\ \vdash v'[x::=v]_{vv} \Leftarrow \tau 1[x::=v]_{\tau v}$ **using** *subst-infer-check-v check-assignI gs* **by** *metis*

have $P\ ;\ \mathcal{B}\ ;\ \Gamma_2[x::=v]_{\Gamma v}\ @\ \Gamma_1\ \vdash \{\!|\ z : B\text{-}unit\ |\ TRUE\ |\!\}[x::=v]_{\tau v} \lesssim \tau'[x::=v]_{\tau v}$ **proof**(*rule subst-subtype-tau*)
show $P\ ;\ \mathcal{B}\ ;\ \Gamma_1\ \vdash v \Rightarrow \tau$ **using** *check-assignI* **by** *auto*
show $P\ ;\ \mathcal{B}\ ;\ \Gamma_1\ \vdash \tau \lesssim \{\!|\ z : b\ |\ c\ |\!\}$ **using** *check-assignI* **by** *meson*
show $P\ ;\ \mathcal{B}\ ;\ \Gamma_2\ @\ (x,\ b,\ c[z::=[x]^v]_{cv})\ \#_\Gamma\ \Gamma_1\ \vdash \{\!|\ z : B\text{-}unit\ |\ TRUE\ |\!\} \lesssim \tau'$ **using** *check-assignI* **by** (*metis Abs1-eq-iff*(3) *τ.eq-iff c.fresh*(1) *c.perm-simps*(1))
show *atom* $z \sharp (x,\ v)$ **using** *check-assignI* **by** *auto*
qed
moreover have $\{\!|\ z : B\text{-}unit\ |\ TRUE\ |\!\}[x::=v]_{\tau v} = \{\!|\ z : B\text{-}unit\ |\ TRUE\ |\!\}$ **using** *subst-tv.simps subst-cv.simps check-assignI* **by** *presburger*
ultimately show $P\ ;\ \mathcal{B}\ ;\ \Gamma[x::=v]_{\Gamma v}\ \vdash \{\!|\ z : B\text{-}unit\ |\ TRUE\ |\!\} \lesssim \tau'[x::=v]_{\tau v}$ **using** *gs* **by** *auto*
qed
thus *?case* **using** *subst-sv.simps*(5) **by** *auto*

next
case (*check-whileI* $\Theta\ \Phi\ \mathcal{B}\ \Gamma\ \Delta$ *s1* z' *s2* τ')
have *wfG* $\Theta\ \mathcal{B}\ (\Gamma_2\ @\ (x,\ b,\ c[z::=[x]^v]_{cv})\ \#_\Gamma\ \Gamma_1)$ **using** *check-whileI check-s-wf* **by** *meson*
hence $**$: $\Gamma[x::=v]_{\Gamma v} = \Gamma_2[x::=v]_{\Gamma v}@\Gamma_1$ **using** *subst-g-inside wf check-whileI* **by** *auto*
have *teq*: $(\{\!|\ z : B\text{-}unit\ |\ TRUE\ |\!\})[x::=v]_{\tau v} = (\{\!|\ z : B\text{-}unit\ |\ TRUE\ |\!\})$ **by**(*auto simp add*: *subst-sv.simps check-whileI*)
moreover have $(\{\!|\ z : B\text{-}unit\ |\ TRUE\ |\!\}) = (\{\!|\ z' : B\text{-}unit\ |\ TRUE\ |\!\})$ **using** *type-eq-flip c.fresh flip-fresh-fresh* **by** *metis*
ultimately have *teq2*:$(\{\!|\ z' : B\text{-}unit\ |\ TRUE\ |\!\})[x::=v]_{\tau v} = (\{\!|\ z' : B\text{-}unit\ |\ TRUE\ |\!\})$ **by** *metis*

hence $\Theta\ ;\ \Phi\ ;\ \mathcal{B}\ ;\ \Gamma[x::=v]_{\Gamma v}\ ;\ \Delta[x::=v]_{\Delta v}\ \vdash s1[x::=v]_{sv} \Leftarrow \{\!|\ z' : B\text{-}bool\ |\ TRUE\ |\!\}$ **using** *check-whileI subst-sv.simps subst-top-eq* **by** *metis*
moreover have $\Theta\ ;\ \Phi\ ;\ \mathcal{B}\ ;\ \Gamma[x::=v]_{\Gamma v}\ ;\ \Delta[x::=v]_{\Delta v}\ \vdash s2[x::=v]_{sv} \Leftarrow \{\!|\ z' : B\text{-}unit\ |\ TRUE\ |\!\}$ **using** *check-whileI subst-top-eq* **by** *metis*
moreover have $\Theta\ ;\ \mathcal{B}\ ;\ \Gamma[x::=v]_{\Gamma v}\ \vdash \{\!|\ z' : B\text{-}unit\ |\ TRUE\ |\!\} \lesssim \tau'[x::=v]_{\tau v}$ **proof** −
have $\Theta\ ;\ \mathcal{B}\ ;\ \Gamma_2[x::=v]_{\Gamma v}\ @\ \Gamma_1\ \vdash \{\!|\ z' : B\text{-}unit\ |\ TRUE\ |\!\}[x::=v]_{\tau v} \lesssim \tau'[x::=v]_{\tau v}$ **proof**(*rule subst-subtype-tau*)
show $\Theta\ ;\ \mathcal{B}\ ;\ \Gamma_1\ \vdash v \Rightarrow \tau$ **by**(*auto simp add*: *check-whileI*)
show $\Theta\ ;\ \mathcal{B}\ ;\ \Gamma_1\ \vdash \tau \lesssim \{\!|\ z : b\ |\ c\ |\!\}$ **by**(*auto simp add*: *check-whileI*)
show $\Theta\ ;\ \mathcal{B}\ ;\ \Gamma_2\ @\ (x,\ b,\ c[z::=[x]^v]_{cv})\ \#_\Gamma\ \Gamma_1\ \vdash \{\!|\ z' : B\text{-}unit\ |\ TRUE\ |\!\} \lesssim \tau'$ **using** *check-whileI* **by** *metis*
show *atom* $z \sharp (x,\ v)$ **by**(*auto simp add*: *check-whileI*)
qed
thus *?thesis* **using** *teq2* $**$ **by** *auto*
qed

ultimately have $\Theta\ ;\ \Phi\ ;\ \mathcal{B}\ ;\ \Gamma[x::=v]_{\Gamma v}\ ;\ \Delta[x::=v]_{\Delta v}\ \vdash$ *AS-while* $s1[x::=v]_{sv}\ s2[x::=v]_{sv} \Leftarrow \tau'[x::=v]_{\tau v}$

using *Typing.check-whileI* **by** *metis*
then show *?case* **using** *subst-sv.simps* **by** *metis*
next
case (*check-seqI P* Φ $\mathcal{B}$ Γ Δ *s1 z s2* τ)
hence *P* ; Φ; $\mathcal{B}$; $\Gamma[x::=v]_{\Gamma v}$; $\Delta[x::=v]_{\Delta v}$ ⊢ *AS-seq* (*s1*$[x::=v]_{sv}$) (*s2*$[x::=v]_{sv}$) ⇐ $\tau[x::=v]_{\tau v}$
using *Typing.check-seqI subst-top-eq check-seqI* **by** *metis*
then show *?case* **using** *subst-sv.simps* **by** *metis*
next
case (*check-caseI* Θ Φ $\mathcal{B}$ Γ Δ *tid dclist v' cs* τ *za*)

have *wf*: *wfG* Θ $\mathcal{B}$ Γ **using** *check-caseI check-v-wf* **by** *simp*
have ∗∗: $\Gamma[x::=v]_{\Gamma v} = \Gamma_2[x::=v]_{\Gamma v}@\Gamma_1$ **using** *subst-g-inside wf check-caseI* **by** *auto*

have Θ ; Φ ; $\mathcal{B}$; $\Gamma[x::=v]_{\Gamma v}$; $\Delta[x::=v]_{\Delta v}$ ⊢ *AS-match* ($v'[x::=v]_{vv}$) (*subst-branchlv cs x v*) ⇐ $\tau[x::=v]_{\tau v}$ **proof**(*rule Typing.check-caseI*)
show *check-branch-list* Θ Φ $\mathcal{B}$ ($\Gamma[x::=v]_{\Gamma v}$) $\Delta[x::=v]_{\Delta v}$ *tid dclist* $v'[x::=v]_{vv}$ (*subst-branchlv cs x v*) ($\tau[x::=v]_{\tau v}$) **using** *check-caseI* **by** *auto*
show *AF-typedef tid dclist* ∈ *set* Θ **using** *check-caseI* **by** *auto*
show Θ ; $\mathcal{B}$; $\Gamma[x::=v]_{\Gamma v}$ ⊢ $v'[x::=v]_{vv}$ ⇐ {| *za* : *B-id tid* | *TRUE* |} **proof** −
have Θ ; $\mathcal{B}$; Γ_2 @ (*x*, *b*, *c*[*z*::=[*x*]^v]$_{cv}$) #Γ Γ_1 ⊢ *v'* ⇐ {| *za* : *B-id tid* | *TRUE* |}
using *check-caseI* **by** *argo*
hence Θ ; $\mathcal{B}$; $\Gamma_2[x::=v]_{\Gamma v}$ @ Γ_1 ⊢ $v'[x::=v]_{vv}$ ⇐ ({| *za* : *B-id tid* | *TRUE* |})$[x::=v]_{\tau v}$
using *check-caseI subst-infer-check-v*[*OF check-caseI*(7) - *check-caseI*(8) *check-caseI*(9)] **by** *meson*
moreover have ({| *za* : *B-id tid* | *TRUE* |}) = (({| *za* : *B-id tid* | *TRUE* |})$[x::=v]_{\tau v}$)
using *subst-cv.simps subst-tv.simps subst-cv-true* **by** *fast*
ultimately show *?thesis* **using** *check-caseI* ∗∗ **by** *argo*
qed
show *wfTh* Θ **using** *check-caseI* **by** *auto*
qed
thus *?case* **using** *subst-branchlv.simps subst-sv.simps*(4) **by** *metis*
next
case (*check-ifI z'* Θ Φ $\mathcal{B}$ Γ Δ *va s1 s2* τ')
show *?case* **unfolding** *subst-sv.simps* **proof**
show ‹*atom z'* ♯ (Θ, Φ, $\mathcal{B}$, $\Gamma[x::=v]_{\Gamma v}$, $\Delta[x::=v]_{\Delta v}$, $va[x::=v]_{vv}$, *s1*$[x::=v]_{sv}$, *s2*$[x::=v]_{sv}$, $\tau'[x::=v]_{\tau v}$)›

by(*subst-tuple-mth add*: *check-ifI*)
have ∗:{| *z'* : *B-bool* | *TRUE* |}$[x::=v]_{\tau v}$ = {| *z'* : *B-bool* | *TRUE* |} **using** *subst-tv.simps check-ifI*
by (*metis freshers*(19) *subst-cv.simps*(1) *type-eq-subst*)
have ∗∗: $\Gamma[x::=v]_{\Gamma v} = \Gamma_2[x::=v]_{\Gamma v}@\Gamma_1$ **using** *subst-g-inside wf check-ifI check-v-wf* **by** *metis*
show ‹Θ ; $\mathcal{B}$; $\Gamma[x::=v]_{\Gamma v}$ ⊢ $va[x::=v]_{vv}$ ⇐ {| *z'* : *B-bool* | *TRUE* |}›
proof(*subst* ∗[*symmetric*], *rule subst-infer-check-v1*[**where** Γ_1=Γ_2 **and** Γ_2=Γ_1])
show Γ = Γ_2 @ ((*x*, *b* ,*c*[*z*::=[*x*]^v]$_{cv}$) #Γ Γ_1) **using** *check-ifI* **by** *metis*
show ‹ Θ ; $\mathcal{B}$; Γ_1 ⊢ *v* ⇒ τ› **using** *check-ifI* **by** *metis*
show ‹Θ ; $\mathcal{B}$; Γ ⊢ *va* ⇐ {| *z'* : *B-bool* | *TRUE* |}› **using** *check-ifI* **by** *metis*
show ‹Θ ; $\mathcal{B}$; Γ_1 ⊢ τ ≲ {| *z* : *b* | *c* |}› **using** *check-ifI* **by** *metis*
show ‹*atom z* ♯ (*x*, *v*)› **using** *check-ifI* **by** *metis*
qed

have {| *z'* : *b-of* $\tau'[x::=v]_{\tau v}$ | [$va[x::=v]_{vv}$]^{ce} == [[*L-true*]^v]^{ce} *IMP c-of* $\tau'[x::=v]_{\tau v}$ *z'* |}
= {| *z'* : *b-of* τ' | [*va*]^{ce} == [[*L-true*]^v]^{ce} *IMP c-of* τ' *z'* |}$[x::=v]_{\tau v}$
by(*simp add*: *subst-tv.simps fresh-Pair check-ifI b-of-subst subst-v-c-of*)

thus ‹ Θ ; Φ ; $\mathcal{B}$; $\Gamma[x::=v]_{\Gamma v}$; $\Delta[x::=v]_{\Delta v}$ $\vdash$ $s1[x::=v]_{sv} \Leftarrow \{\!|\ z' : b\text{-}of\ \tau'[x::=v]_{\tau v}\ |\ [\ va[x::=v]_{vv}\ ]^{ce}\ ==\ [\ [\ L\text{-}true\ ]^{v}\ ]^{ce}\ \ IMP\ \ c\text{-}of\ \tau'[x::=v]_{\tau v}\ z'\ |\!\}$ ›
using *check-ifI* **by** *metis*
have $\{\!|\ z' : b\text{-}of\ \tau'[x::=v]_{\tau v}\ |\ [\ va[x::=v]_{vv}\ ]^{ce}\ ==\ [\ [\ L\text{-}false\ ]^{v}\ ]^{ce}\ \ IMP\ \ c\text{-}of\ \tau'[x::=v]_{\tau v}\ z'\ |\!\}$ = $\{\!|\ z' : b\text{-}of\ \tau'\ |\ [\ va\ ]^{ce}\ ==\ [\ [\ L\text{-}false\ ]^{v}\ ]^{ce}\ \ IMP\ \ c\text{-}of\ \tau'\ z'\ |\!\}[x::=v]_{\tau v}$
by(*simp add*: *subst-tv.simps fresh-Pair check-ifI b-of-subst subst-v-c-of*)
thus ‹ Θ ; Φ ; $\mathcal{B}$; $\Gamma[x::=v]_{\Gamma v}$; $\Delta[x::=v]_{\Delta v}$ $\vdash$ $s2[x::=v]_{sv} \Leftarrow \{\!|\ z' : b\text{-}of\ \tau'[x::=v]_{\tau v}\ |\ [\ va[x::=v]_{vv}\ ]^{ce}\ ==\ [\ [\ L\text{-}false\ ]^{v}\ ]^{ce}\ \ IMP\ \ c\text{-}of\ \tau'[x::=v]_{\tau v}\ z'\ |\!\}$ ›
using *check-ifI* **by** *metis*
qed
qed

lemma *subst-check-check-s*:
fixes $v::v$ **and** $s::s$ **and** $cs::branch\text{-}s$ **and** $x::x$ **and** $c::c$ **and** $b::b$ **and** $\Gamma_1::\Gamma$ **and** $\Gamma_2::\Gamma$
assumes Θ ; $\mathcal{B}$; $\Gamma_1 \vdash v \Leftarrow \{\!|\ z : b \mid c\ |\!\}$ **and** $atom\ z \sharp (x, v)$
and *check-s* $\Theta\ \Phi\ \mathcal{B}\ \Gamma\ \Delta\ \ s\ \ \tau'$ **and** $\Gamma\ = (\Gamma_2@((x,b,c[z::=[x]^v]_{cv})\#_\Gamma\Gamma_1))$
shows *check-s* $\Theta\ \Phi\ \mathcal{B}$ (*subst-gv* $\Gamma\ x\ v$) $\Delta[x::=v]_{\Delta v}$ ($s[x::=v]_{sv}$) (*subst-tv* $\tau'\ x\ v$)
proof −
obtain τ **where** Θ ; $\mathcal{B}$; $\Gamma_1 \vdash v \Rightarrow \tau \wedge \Theta$; $\mathcal{B}$; $\Gamma_1 \vdash \tau \lesssim \{\!|\ z : b \mid c\ |\!\}$ **using** *check-v-elims assms* **by** *auto*
thus *?thesis* **using** *subst-infer-check-s assms* **by** *metis*
qed

If a statement checks against a type τ then it checks against a supertype of τ

lemma *check-s-supertype*:
fixes $v::v$ **and** $s::s$ **and** $cs::branch\text{-}s$ **and** $x::x$ **and** $c::c$ **and** $b::b$ **and** $\Gamma::\Gamma$ **and** $\Gamma'::\Gamma$ **and** $css::branch\text{-}list$
shows *check-s* $\Theta\ \Phi\ \mathcal{B}\ G\ \Delta\ \ s\ \ t1 \Longrightarrow \Theta$; $\mathcal{B}$; $G \vdash t1 \lesssim t2 \Longrightarrow$ *check-s* $\Theta\ \Phi\ \mathcal{B}\ G\ \Delta\ \ s\ \ t2$ **and**
check-branch-s $\Theta\ \Phi\ \mathcal{B}\ G\ \Delta$ *tid cons const v cs t1* $\Longrightarrow \Theta$; $\mathcal{B}$; $G \vdash t1 \lesssim t2 \Longrightarrow$ *check-branch-s* $\Theta\ \Phi\ \mathcal{B}\ G\ \Delta$ *tid cons const v cs t2* **and**
check-branch-list $\Theta\ \Phi\ \mathcal{B}\ G\ \Delta$ *tid dclist v css t1* $\Longrightarrow \Theta$; $\mathcal{B}$; $G \vdash t1 \lesssim t2 \Longrightarrow$ *check-branch-list* $\Theta\ \Phi\ \mathcal{B}\ G\ \Delta$ *tid dclist v css t2*
proof(*induct arbitrary*: *t2* **and** *t2* **and** *t2 rule*: *check-s-check-branch-s-check-branch-list.inducts*)
case (*check-valI* $\Theta\ \mathcal{B}\ \Gamma\ \Delta\ \ \Phi\ v\ \tau'\ \tau$)
hence Θ ; $\mathcal{B}$; $\Gamma \vdash \tau' \lesssim t2$ **using** *subtype-trans* **by** *meson*
then show *?case* **using** *subtype-trans Typing.check-valI check-valI* **by** *metis*

next
case (*check-letI x* $\Theta\ \Phi\ \mathcal{B}\ \Gamma\ \Delta\ e\ \tau\ z\ s\ b\ c$)
show *?case* **proof**(*rule Typing.check-letI*)
show *atom x* $\sharp(\Theta, \Phi, \mathcal{B}, \Gamma, \Delta, e, t2)$ **using** *check-letI subtype-fresh-tau fresh-prodN* **by** *metis*
thm *subtype-fresh-tau*
show *atom z* $\sharp$ $(x, \Theta, \Phi, \mathcal{B}, \Gamma, \Delta, e, t2, s)$ **using** *check-letI*(*2*) *subtype-fresh-tau*[*of z* τ Γ - - *t2*] *fresh-prodN check-letI*(*6*) **by** *auto*
show Θ ; Φ ; $\mathcal{B}$; Γ ; $\Delta \vdash e \Rightarrow \{\!|\ z : b\ \mid c\ |\!\}$ **using** *check-letI* **by** *meson*

have *wfG* $\Theta\ \mathcal{B}$ $((x,\ b,\ c[z::=[x]^v]_v)\ \#_\Gamma\ \Gamma)$ **using** *check-letI check-s-wf subst-defs* **by** *metis*
moreover have *setG* $\Gamma \subseteq$ *setG* $((x,\ b,\ c[z::=[x]^v]_v)\ \#_\Gamma\ \Gamma)$ **by** *auto*
ultimately have Θ ; $\mathcal{B}$; $(x,\ b,\ c[z::=[x]^v]_v)\ \#_\Gamma\ \Gamma\ \vdash \tau \lesssim t2$ **using** *subtype-weakening*[*OF check-letI*(*6*)] **by** *auto*
thus Θ ; Φ ; $\mathcal{B}$; $(x,\ b,\ c[z::=[x]^v]_v)\ \#_\Gamma\ \Gamma$; $\Delta\ \vdash s \Leftarrow t2$ **using** *check-letI subst-defs* **by** *metis*
qed

next

next
case (*check-branch-list-consI* Θ Φ $\mathcal{B}$ Γ Δ *tid dclist v cs* τ *css*)
then show *?case* **using** *Typing.check-branch-list-consI* **by** *auto*
next
case (*check-branch-list-finalI* Θ Φ $\mathcal{B}$ Γ Δ *tid dclist v cs* τ)
then show *?case* **using** *Typing.check-branch-list-finalI* **by** *auto*
next
case (*check-branch-s-branchI* Θ $\mathcal{B}$ Γ Δ τ *const x* Φ *tid cons v s*)
show *?case* **proof**
have *atom x* $\sharp$ *t2* **using** *subtype-fresh-tau*[*of x* τ] *check-branch-s-branchI*(*5*,*8*) *fresh-prodN* **by** *metis*
thus *atom x* $\sharp$ (Θ, Φ, $\mathcal{B}$, Γ, Δ, *tid*, *cons*, *const*, *v*, *t2*) **using** *check-branch-s-branchI fresh-prodN* **by** *metis*
show *wfT* Θ $\mathcal{B}$ Γ *t2* **using** *subtype-wf check-branch-s-branchI* **by** *meson*
show Θ ; Φ ; $\mathcal{B}$; (*x*, *b-of const*, *CE-val v* == *CE-val*(*V-cons tid cons* (*V-var x*)) *AND c-of const x*) $\#_\Gamma$ Γ ; Δ $\vdash$ *s* $\Leftarrow$ *t2* **proof** −
have *wfG* Θ $\mathcal{B}$ ((*x*, *b-of const*, *CE-val v* == *CE-val*(*V-cons tid cons* (*V-var x*)) *AND c-of const x*) $\#_\Gamma$ Γ) **using** *check-s-wf check-branch-s-branchI* **by** *metis*
moreover have *setG* Γ $\subseteq$ *setG* ((*x*, *b-of const*, *CE-val v* == *CE-val* (*V-cons tid cons* (*V-var x*)) *AND c-of const x*) $\#_\Gamma$ Γ) **by** *auto*
hence Θ ; $\mathcal{B}$; ((*x*, *b-of const*, *CE-val v* == *CE-val*(*V-cons tid cons* (*V-var x*)) *AND c-of const x*) $\#_\Gamma$ Γ) $\vdash$ τ $\lesssim$ *t2*
using *check-branch-s-branchI subtype-weakening*
using *calculation* **by** *presburger*
thus *?thesis* **using** *check-branch-s-branchI* **by** *presburger*
qed
qed(*auto simp add*: *check-branch-s-branchI*)

next
case (*check-ifI z* Θ Φ $\mathcal{B}$ Γ Δ *v s1 s2* τ)
show *?case* **proof**(*rule Typing.check-ifI*)
have $*$:*atom z* $\sharp$ *t2* **using** *subtype-fresh-tau*[*of z* τ Γ] *check-ifI fresh-prodN* **by** *auto*
thus ‹*atom z* $\sharp$ (Θ, Φ, $\mathcal{B}$, Γ, Δ, *v*, *s1*, *s2*, *t2*)› **using** *check-ifI fresh-prodN* **by** *auto*
show ‹Θ ; $\mathcal{B}$; Γ $\vdash$ *v* $\Leftarrow$ {| *z* : *B-bool* | *TRUE* |}› **using** *check-ifI* **by** *auto*
show ‹ Θ ; Φ ; $\mathcal{B}$; Γ ; Δ $\vdash$ *s1* $\Leftarrow$ {| *z* : *b-of t2* | [*v*]^{ce} == [[*L-true*]^{v}]^{ce} *IMP c-of t2 z* |}›
using *check-ifI subtype-if1 fresh-prodN base-for-lit.simps b-of.simps* $*$ *check-v-wf* **by** *metis*

show ‹ Θ ; Φ ; $\mathcal{B}$; Γ ; Δ $\vdash$ *s2* $\Leftarrow$ {| *z* : *b-of t2* | [*v*]^{ce} == [[*L-false*]^{v}]^{ce} *IMP c-of t2 z* |}›
using *check-ifI subtype-if1 fresh-prodN base-for-lit.simps b-of.simps* $*$ *check-v-wf* **by** *metis*
qed
next
case (*check-assertI x* Θ Φ $\mathcal{B}$ Γ Δ *c* τ *s*)
thm *subtype-fresh-tau*[**where** *?t1.0*=τ **and** *?x*=*x*]
show *?case* **proof**
have *atom x* $\sharp$ *t2* **using** *subtype-fresh-tau*[*OF - -* ‹Θ ; $\mathcal{B}$; Γ $\vdash$ τ $\lesssim$ *t2*›] *check-assertI fresh-prodN* **by** *simp*
thus *atom x* $\sharp$ (Θ, Φ, $\mathcal{B}$, Γ, Δ, *c*, *t2*, *s*) **using** *subtype-fresh-tau check-assertI fresh-prodN* **by** *simp*
have Θ ; $\mathcal{B}$; (*x*, *B-bool*, *c*) $\#_\Gamma$ Γ $\vdash$ τ $\lesssim$ *t2* **apply**(*rule subtype-weakening*)

using *check-assertI* **apply** *simp*
using *setG.simps* **apply** *blast*
using *check-assertI check-s-wf* **by** *simp*
thus Θ ; Φ ; $\mathcal{B}$; $(x, B\text{-}bool, c)$ $\#_\Gamma$ Γ ; Δ $\vdash$ $s \Leftarrow t2$ **using** *check-assertI* **by** *simp*
show Θ ; $\mathcal{B}$; Γ $\models$ c **using** *check-assertI* **by** *auto*
show Θ ; $\mathcal{B}$; $\Gamma \vdash_{wf} \Delta$ **using** *check-assertI* **by** *auto*
qed
next
case (*check-let2I x P* Φ $\mathcal{B}$ *G* Δ *t s1* τ *s2*)
have *wfG P* $\mathcal{B}$ $((x, b\text{-}of\ t, c\text{-}of\ t\ x)$ $\#_\Gamma$ *G*)
using *check-let2I check-s-wf* **by** *metis*
moreover have *setG G* $\subseteq$ *setG* $((x, b\text{-}of\ t, c\text{-}of\ t\ x)$ $\#_\Gamma$ *G*) **by** *auto*
ultimately have $*$:*P* ; $\mathcal{B}$; $(x, b\text{-}of\ t, c\text{-}of\ t\ x)$ $\#_\Gamma$ *G* $\vdash$ $\tau \lesssim t2$ **using** *check-let2I subtype-weakening*
by *metis*
show *?case* **proof**(*rule Typing.check-let2I*)
have *atom x* $\sharp$ *t2* **using** *subtype-fresh-tau*[*of x* τ] *check-let2I fresh-prodN* **by** *metis*
thus *atom x* $\sharp$ (*P*, Φ, $\mathcal{B}$, *G*, Δ, *t*, *s1*, *t2*) **using** *check-let2I fresh-prodN* **by** *metis*
show *P* ; Φ ; $\mathcal{B}$; *G* ; Δ $\vdash$ *s1* $\Leftarrow$ *t* **using** *check-let2I* **by** *blast*
show *P* ; Φ ; $\mathcal{B}$;$(x, b\text{-}of\ t, c\text{-}of\ t\ x\ )$ $\#_\Gamma$ *G* ; Δ $\vdash$ *s2* $\Leftarrow$ *t2* **using** *check-let2I* $*$ **by** *blast*
qed
next
case (*check-varI u* Θ Φ $\mathcal{B}$ Γ Δ τ' *v* τ *s*)
show *?case* **proof**(*rule Typing.check-varI*)
have *atom u* $\sharp$ *t2* **using** *u-fresh-t* **by** *auto*
thus ‹*atom u* $\sharp$ (Θ, Φ, $\mathcal{B}$, Γ, Δ, τ', *v*, *t2*)› **using** *check-varI fresh-prodN* **by** *auto*
show ‹Θ ; $\mathcal{B}$; Γ $\vdash$ $v \Leftarrow \tau'$› **using** *check-varI* **by** *auto*
show ‹ Θ ; Φ ; $\mathcal{B}$; Γ ; (u, τ') $\#_\Delta$ Δ $\vdash$ $s \Leftarrow t2$› **using** *check-varI* **by** *auto*
qed
next
case (*check-assignI* Δ *u* τ *P G v z* τ')
then show *?case* **using** *Typing.check-assignI* **by** (*meson subtype-trans*)
next
case (*check-whileI* Δ *G P s1 z s2* τ')
then show *?case* **using** *Typing.check-whileI* **by** (*meson subtype-trans*)
next
case (*check-seqI* Δ *G P s1 z s2* τ)
then show *?case* **using** *Typing.check-seqI* **by** *blast*
next
case (*check-caseI* Δ Γ Θ *tid cs* τ *v z*)
then show *?case* **using** *Typing.check-caseI subtype-trans* **by** *meson*

qed

lemma *subtype-let*:
fixes s'::*s* **and** *cs*::*branch-s* **and** *css*::*branch-list* **and** *v*::*v*
shows Θ ; Φ ; $\mathcal{B}$; *GNil* ; Δ $\vdash$ *AS-let x* e_1 *s* $\Leftarrow$ $\tau \Longrightarrow \Theta$; Φ ; $\mathcal{B}$; *GNil* ; Δ $\vdash$ $e_1 \Rightarrow \tau_1 \Longrightarrow$
Θ ; Φ ; $\mathcal{B}$; *GNil* ; Δ $\vdash$ $e_2 \Rightarrow \tau_2 \Longrightarrow \Theta$; $\mathcal{B}$; *GNil* $\vdash$ $\tau_2 \lesssim \tau_1 \Longrightarrow \Theta$; Φ ; $\mathcal{B}$; *GNil* ; Δ $\vdash$ *AS-let*
x e_2 *s* $\Leftarrow$ τ **and**
check-branch-s Θ Φ $\{||\}$ *GNil* Δ *tid dc const v cs* $\tau \Longrightarrow$ *True* **and**
check-branch-list Θ Φ $\{||\}$ Γ Δ *tid dclist v css* $\tau \Longrightarrow$ *True*
proof(*nominal-induct GNil* Δ *AS-let x* e_1 *s* τ **and** τ **and** τ *avoiding*: e_2 τ_1 τ_2

rule: *check-s-check-branch-s-check-branch-list.strong-induct*)
case (*check-letI x1* Θ Φ $\mathcal{B}$ Δ $\tau 1$ *z1 s1 b1 c1*)
obtain *z2* **and** *b2* **and** *c2* **where** *t2*:$\tau_2 = \{\!|\, z2 : b2 \mid c2 \,|\!\}$ $\wedge$ *atom z2* $\sharp$ (*x1*, Θ, Φ, $\mathcal{B}$, *GNil*, Δ, e_2, $\tau 1$, *s1*)
using *obtain-fresh-z* **by** *metis*

obtain *z1a* **and** *b1a* **and** *c1a* **where** *t1*:$\tau_1 = \{\!|\, z1a : b1a \mid c1a \,|\!\}$ $\wedge$ *atom z1a* $\sharp$ *x1* **using** *infer-e-uniqueness check-letI* **by** *metis*
hence *t3*: $\{\!|\, z1a : b1a \mid c1a \,|\!\} = \{\!|\, z1 : b1 \mid c1 \,|\!\}$ **using** *infer-e-uniqueness check-letI* **by** *metis*

have *beq*: *b1a* = *b2* $\wedge$ *b2* = *b1* **using** *check-letI subtype-eq-base t1 t2 t3* **by** *metis*

have Θ ; Φ ; $\mathcal{B}$; *GNil* ; Δ $\vdash$ *AS-let x1* e_2 *s1* $\Leftarrow$ $\tau 1$ **proof**
show ‹*atom x1* $\sharp$ (Θ, Φ, $\mathcal{B}$, *GNil*, Δ, e_2, $\tau 1$)› **using** *check-letI t2 fresh-prodN* **by** *metis*
show ‹*atom z2* $\sharp$ (*x1*, Θ, Φ, $\mathcal{B}$, *GNil*, Δ, e_2, $\tau 1$, *s1*)› **using** *check-letI t2* **using** *check-letI t2 fresh-prodN* **by** *metis*
show ‹Θ ; Φ ; $\mathcal{B}$; *GNil* ; Δ $\vdash$ $e_2 \Rightarrow \{\!|\, z2 : b2 \mid c2 \,|\!\}$› **using** *check-letI t2* **by** *metis*

have ‹ Θ ; Φ ; $\mathcal{B}$; *GNil*@(*x1*, *b2*, *c2*[*z2*::=[*x1*]^v]$_{cv}$) $\#_\Gamma$ *GNil* ; Δ $\vdash$ *s1* $\Leftarrow$ $\tau 1$›
proof(*rule ctx-subtype-s*)
have *c1a*[*z1a*::=[*x1*]^v]$_{cv}$ = *c1*[*z1*::=[*x1*]^v]$_{cv}$ **using** *subst-v-flip-eq-two subst-v-c-def t3* τ*.eq-iff* **by** *metis*
thus ‹ Θ ; Φ ; $\mathcal{B}$; *GNil* @ (*x1*, *b2*, *c1a*[*z1a*::=[*x1*]^v]$_{cv}$) $\#_\Gamma$ *GNil* ; Δ $\vdash$ *s1* $\Leftarrow$ $\tau 1$› **using** *check-letI beq append-g.simps subst-defs* **by** *metis*
show ‹Θ ; $\mathcal{B}$; *GNil* $\vdash$ $\{\!|\, z2 : b2 \mid c2 \,|\!\} \lesssim \{\!|\, z1a : b2 \mid c1a \,|\!\}$› **using** *check-letI beq t1 t2* **by** *metis*
have *atom x1* $\sharp$ *c2* **using** *t2 check-letI* τ*-fresh-c fresh-prodN* **by** *blast*
moreover have *atom x1* $\sharp$ *c1a* **using** *t1 check-letI* τ*-fresh-c fresh-prodN* **by** *blast*
ultimately show ‹*atom x1* $\sharp$ (*z1a*, *z2*, *c1a*, *c2*)› **using** *t1 t2 fresh-prodN fresh-x-neq* **by** *metis*
qed

thus ‹ Θ ; Φ ; $\mathcal{B}$; (*x1*, *b2*, *c2*[*z2*::=[*x1*]^v]$_v$) $\#_\Gamma$ *GNil* ; Δ $\vdash$ *s1* $\Leftarrow$ $\tau 1$› **using** *append-g.simps subst-defs* **by** *metis*
qed

moreover have *AS-let x1* e_2 *s1* = *AS-let x* e_2 *s* **using** *check-letI s-branch-s-branch-list.eq-iff* **by** *metis*

ultimately show *?case* **by** *metis*

qed(*auto+*)

end

Chapter 15

Base Type Variable Substitition Lemmas

lemma *subst-vv-subst-bb-commute*:
  fixes *bv*::*bv* **and** *b*::*b* **and** *x*::*x* **and** *v*::*v*
  assumes *atom bv* ♯ *v*
  shows $(v'[x::=v]_{vv})[bv::=b]_{vb} = (v'[bv::=b]_{vb})[x::=v]_{vv}$
using *assms* **proof**(*nominal-induct v' rule*: *v.strong-induct*)
  case (*V-lit x*)
  then show *?case* **using** *subst-vb.simps subst-vv.simps* **by** *simp*
next
  case (*V-var y*)
  hence $v[bv::=b]_{vb}=v$ **using** *forget-subst subst-b-v-def* **by** *metis*
  then show *?case* **unfolding** *subst-vb.simps*(*2*) *subst-vv.simps*(*2*) **using** *V-var* **by** *auto*
next
  case (*V-pair x1a x2a*)
  then show *?case* **using** *subst-vb.simps subst-vv.simps* **by** *simp*
next
  case (*V-cons x1a x2a x3*)
  then show *?case* **using** *subst-vb.simps subst-vv.simps* **by** *simp*
next
  case (*V-consp x1a x2a x3 x4*)
  then show *?case* **using** *subst-vb.simps subst-vv.simps* **by** *simp*
qed

lemma *subst-cev-subst-bb-commute*:
  fixes *bv*::*bv* **and** *b*::*b* **and** *x*::*x* **and** *v*::*v*
  assumes *atom bv* ♯ *v*
  shows $(ce[x::=v]_v)[bv::=b]_{ceb} = (ce[bv::=b]_{ceb})[x::=v]_v$
  using *assms* **apply** (*nominal-induct ce rule*: *ce.strong-induct*, (*simp add*: *subst-vv-subst-bb-commute subst-ceb.simps subst-cv.simps*))
  using *assms subst-vv-subst-bb-commute subst-ceb.simps subst-cv.simps*
  apply (*simp add*: *subst-v-ce-def*)+
  done

lemma *subst-cv-subst-bb-commute*:

fixes $bv::bv$ **and** $b::b$ **and** $x::x$ **and** $v::v$
assumes *atom bv* $\sharp$ *v*
shows $c[x::=v]_{cv}[bv::=b]_{cb} = (c[bv::=b]_{cb})[x::=v]_{cv}$
using *assms* **apply** (*nominal-induct c rule*: *c.strong-induct*)
using *assms subst-vv-subst-bb-commute subst-ceb.simps subst-cv.simps*
subst-v-c-def subst-b-c-def **apply** *auto*
using *subst-cev-subst-bb-commute subst-v-ce-def* **apply** *auto+*
done

thm *subst-cv-subst-bb-commute*

lemma *subst-b-c-of*:
$(c\text{-}of\ \tau\ z)[bv::=b]_{cb} =$ *c-of* $(\tau[bv::=b]_{\tau b})\ z$
proof(*nominal-induct* τ *avoiding*: *z rule*:τ.*strong-induct*)
case (*T-refined-type z' b' c'*)
moreover have *atom bv* $\sharp$ [*z*]^v **using** *fresh-at-base v.fresh* **by** *auto*
ultimately show *?case* **using** *subst-cv-subst-bb-commute*[*of bv V-var z c' z' b*] *c-of.simps subst-tb.simps*
by *metis*
qed

lemma *subst-b-b-of*:
$(b\text{-}of\ \tau)[bv::=b]_{bb} =$ *b-of* $(\tau[bv::=b]_{\tau b})$
by(*nominal-induct* τ *rule*:τ.*strong-induct*, *simp add*: *b-of.simps subst-tb.simps*)

lemma *subst-b-if*:
$\{\!|\ z : b\text{-}of\ \tau[bv::=b]_{\tau b}\ |\ CE\text{-}val\ (v[bv::=b]_{vb})\ ==\ CE\text{-}val\ (V\text{-}lit\ ll)\ \ IMP\ \ c\text{-}of\ \tau[bv::=b]_{\tau b}\ z\ |\!\} =$
$\{\!|\ z : b\text{-}of\ \tau\ |\ CE\text{-}val\ (v)\ ==\ CE\text{-}val\ (V\text{-}lit\ ll)\ \ IMP\ \ c\text{-}of\ \tau\ z\ |\!\}[bv::=b]_{\tau b}$
unfolding *subst-tb.simps subst-cb.simps subst-ceb.simps subst-vb.simps* **using** *subst-b-b-of subst-b-c-of*
by *auto*

lemma *subst-b-top-eq*:
$\{\!|\ z : B\text{-}unit\ |\ TRUE\ |\!\}[bv::=b]_{\tau b} = \{\!|\ z : B\text{-}unit\ |\ TRUE\ |\!\}$ **and** $\{\!|\ z : B\text{-}bool\ |\ TRUE\ |\!\}[bv::=b]_{\tau b} =$
$\{\!|\ z : B\text{-}bool\ |\ TRUE\ |\!\}$ **and** $\{\!|\ z : B\text{-}id\ tid\ |\ TRUE\ |\!\} = \{\!|\ z : B\text{-}id\ tid\ |\ TRUE\ |\!\}[bv::=b]_{\tau b}$
unfolding *subst-tb.simps subst-bb.simps subst-cb.simps* **by** *auto*

lemmas *subst-b-eq = subst-b-c-of subst-b-b-of subst-b-if subst-b-top-eq*

lemma *subst-cx-subst-bb-commute*[*simp*]:
fixes $bv::bv$ **and** $b::b$ **and** $x::x$ **and** $v'::c$
shows $(v'[x::=V\text{-}var\ y]_{cv})[bv::=b]_{cb} = (v'[bv::=b]_{cb})[x::=V\text{-}var\ y]_{cv}$
using *subst-cv-subst-bb-commute fresh-at-base v.fresh* **by** *auto*

lemma *subst-b-infer-b*:
fixes $l::l$ **and** $b::b$
assumes $\vdash l \Rightarrow \tau$ **and** $\Theta\ ;\ \{||\} \vdash_{wf} b$ **and** $B = \{|bv|\}$
shows $\vdash l \Rightarrow (\tau[bv::=b]_{\tau b})$
using *assms infer-l-form3 infer-l-form4 wf-b-subst infer-l-wf subst-tb.simps base-for-lit.simps subst-tb.simps*
subst-b-base-for-lit subst-cb.simps(6) *subst-ceb.simps*(1) *subst-vb.simps*(1) *subst-vb.simps*(2) *type-l-eq*
by *metis*

lemma *subst-b-subtype*:
fixes $s::s$ **and** $b'::b$
assumes $\Theta\ ;\ \{|bv|\}\ ;\ \Gamma\ \vdash \tau 1 \lesssim \tau 2$ **and** $\Theta\ ;\ \{||\} \vdash_{wf}\ b'$ **and** $B = \{|bv|\}$
shows $\Theta\ ;\ \{||\}\ ;\ \Gamma[bv::=b']_{\Gamma b}\ \vdash \tau 1[bv::=b']_{\tau b} \lesssim \tau 2[bv::=b']_{\tau b}$
using *assms* **proof**(*nominal-induct* $\{|bv|\}\ \Gamma\ \tau 1\ \tau 2$ *rule*:*subtype.strong-induct*)
case (*subtype-baseI* $x\ \Theta\ \Gamma\ z\ c\ z'\ c'\ b$)

hence $**$: $\Theta\ ;\ \{|bv|\}\ ;\ (x,\ b,\ c[z::=V\text{-}var\ x]_{cv})\ \#_\Gamma\ \Gamma\ \models c'[z'::=V\text{-}var\ x]_{cv}$ **using** *validI subst-defs*
by *metis*

thm *Typing.subtype-baseI*
have $\Theta\ ;\ \{||\}\ ;\ \Gamma[bv::=b']_{\Gamma b}\ \vdash \{\!|\ z : b[bv::=b']_{bb}\ |\ c[bv::=b']_{cb}\ |\!\} \lesssim \{\!|\ z' : b[bv::=b']_{bb}\ |\ c'[bv::=b']_{cb}\ |\!\}$ **proof**(*rule Typing.subtype-baseI*)
show $\Theta\ ;\ \{||\}\ ;\ \Gamma[bv::=b']_{\Gamma b}\ \vdash_{wf} \{\!|\ z : b[bv::=b']_{bb}\ |\ c[bv::=b']_{cb}\ |\!\}$
using *subtype-baseI assms wf-b-subst*(*4*) *subst-tb.simps subst-defs* **by** *metis*
show $\Theta\ ;\ \{||\}\ ;\ \Gamma[bv::=b']_{\Gamma b}\ \vdash_{wf} \{\!|\ z' : b[bv::=b']_{bb}\ |\ c'[bv::=b']_{cb}\ |\!\}$
using *subtype-baseI assms wf-b-subst*(*4*) *subst-tb.simps* **by** *metis*
show *atom* $x \sharp (\Theta,\ \{||\}::bv\ fset,\ \Gamma[bv::=b']_{\Gamma b},\ z\ ,\ c[bv::=b']_{cb}\ ,\ z'\ ,\ c'[bv::=b']_{cb}\ )$
apply(*unfold fresh-prodN*,*auto simp add*: $*$ *fresh-prodN fresh-empty-fset*)
using *subst-b-fresh-x* $*$ *fresh-prodN* ‹*atom* $x \sharp c$› ‹*atom* $x \sharp c'$› *subst-defs subtype-baseI* **by** *metis*+
have $\Theta\ ;\ \{||\}\ ;\ (x,\ b[bv::=b']_{bb},\ c[z::=V\text{-}var\ x]_v[bv::=b']_{cb})\ \#_\Gamma\ \Gamma[bv::=b']_{\Gamma b}\ \models c'[z'::=V\text{-}var\ x]_v[bv::=b']_{cb}$
using $**$ *subst-b-valid subst-gb.simps assms subtype-baseI* **by** *metis*
thus $\Theta\ ;\ \{||\}\ ;\ (x,\ b[bv::=b']_{bb},\ (c[bv::=b']_{cb})[z::=V\text{-}var\ x]_v)\ \#_\Gamma\ \Gamma[bv::=b']_{\Gamma b}\ \models (c'[bv::=b']_{cb})[z'::=V\text{-}var\ x]_v$
using *subst-defs subst-cv-subst-bb-commute* **by** (*metis subst-cx-subst-bb-commute*)
qed
thus *?case* **using** *subtype-baseI subst-tb.simps subst-defs* **by** *metis*
qed

lemma *subst-b-infer-v*:
fixes $v::v$ **and** $b::b$
assumes $\Theta\ ;\ B\ ;\ G \vdash v \Rightarrow \tau$ **and** $\Theta\ ;\ \{||\} \vdash_{wf}\ b$ **and** $B = \{|bv|\}$
shows $\Theta\ ;\ \{||\}\ ;\ G[bv::=b]_{\Gamma b} \vdash v[bv::=b]_{vb} \Rightarrow (\tau[bv::=b]_{\tau b})$
using *assms* **proof**(*nominal-induct avoiding*: *b rule*: *infer-v.strong-induct*)
case (*infer-v-varI* $\Theta\ \mathcal{B}\ \Gamma\ b'\ c\ x\ z$)
show *?case* **unfolding** *subst-b-simps* **proof**
show $\Theta\ ;\ \{||\}\ \vdash_{wf} \Gamma[bv::=b]_{\Gamma b}$ **using** *infer-v-varI wf-b-subst* **by** *metis*
show *Some* $(b'[bv::=b]_{bb},\ c[bv::=b]_{cb}) =$ *lookup* $\Gamma[bv::=b]_{\Gamma b}\ x$ **using** *subst-b-lookup infer-v-varI* **by** *metis*
show *atom* $z \sharp x$ **using** *infer-v-varI* **by** *auto*
show *atom* $z \sharp \Gamma[bv::=b]_{\Gamma b}$ **using** *infer-v-varI subst-b-fresh-x subst-b-*Γ*-def* **by** *metis*
qed
next
case (*infer-v-litI* $\Theta\ \mathcal{B}\ \Gamma\ l\ \tau$)
then show *?case* **using** *Typing.infer-v-litI subst-b-infer-b*
using *wf-b-subst1*(*3*) **by** *auto*
next
case (*infer-v-pairI z v1 v2* $\Gamma\ \Theta\ \mathcal{B}$ *z1 b1 c1 z2 b2 c2*)

show *?case* **unfolding** *subst-b-simps* **apply**(*rule Typing.infer-v-pairI*)
apply(*simp add*: *subst-b-fresh-x infer-v-pairI*)+
proof(*goal-cases*)
show ‹ Θ ; {||} ; $\Gamma[bv::=b]_{\Gamma b} \vdash v1[bv::=b]_{vb} \Rightarrow \{\!|\ z1 : b1[bv::=b]_{bb} \mid c1[bv::=b]_{cb}\ |\!\}$ › **using** *subst-tb.simps infer-v-pairI* **by** *metis*
show ‹ Θ ; {||} ; $\Gamma[bv::=b]_{\Gamma b} \vdash v2[bv::=b]_{vb} \Rightarrow \{\!|\ z2 : b2[bv::=b]_{bb} \mid c2[bv::=b]_{cb}\ |\!\}$ › **using** *subst-tb.simps infer-v-pairI* **by** *metis*
qed

next
case (*infer-v-consI s dclist* Θ *dc x b' c* $\mathcal{B}$ Γ *v z' c' z*)
show *?case* **unfolding** *subst-b-simps* **proof**

show *AF-typedef s dclist* $\in$ *set* Θ **using** *infer-v-consI* **by** *auto*
show (*dc*, $\{\!|\ x : b' \mid c\ |\!\}$) $\in$ *set dclist* **using** *infer-v-consI* **by** *auto*
have $\vdash_{wf} \Theta$ **using** *infer-v-consI wfX-wfY infer-v-wf* **by** *metis*
hence $**$:*supp* $\{\!|\ x : b' \mid c\ |\!\} = \{\}$ **using** *wfTh-wfT wfT-nil-supp infer-v-consI* **by** *metis*
hence *atom bv* $\sharp$ *b'* **using** *infer-v-consI wfTh-wfT* τ*.fresh fresh-def wfT-supp* τ*.supp* **by** *fastforce*
hence $*$: $b'[bv::=b]_{bb} = b'$ **using** *forget-subst*[*of bv b' b*] *subst-b-b-def* **by** *simp*

hence *teq2*: $\{\!|\ x : b' \mid c\ |\!\}[bv::=b]_{\tau b} = \{\!|\ x : b' \mid c\ |\!\}$ **using** *forget-subst subst-b-*τ*-def fresh-def* $**$
by (*metis empty-iff*)
thus Θ ; {||} ; $\Gamma[bv::=b]_{\Gamma b} \vdash v[bv::=b]_{vb} \Rightarrow \{\!|\ z' : b' \mid c'[bv::=b]_{cb}\ |\!\}$ **using** *infer-v-consI* $*$ *subst-tb.simps* **by** *metis*
show Θ ; {||} ; $\Gamma[bv::=b]_{\Gamma b} \vdash \{\!|\ z' : b' \mid c'[bv::=b]_{cb}\ |\!\} \lesssim \{\!|\ x : b' \mid c\ |\!\}$
using $*$ *teq2 subst-b-subtype subst-tb.simps*
by (*metis infer-v-consI.hyps*(5) *infer-v-consI.prems*(1) *infer-v-consI.prems*(2))

show *atom z* $\sharp$ $v[bv::=b]_{vb}$ **using** *infer-v-consI* **using** *subst-b-fresh-x subst-b-v-def* **by** *metis*
show *atom z* $\sharp$ $\Gamma[bv::=b]_{\Gamma b}$ **using** *infer-v-consI subst-g-b-x-fresh* **by** *auto*
qed
next
case (*infer-v-conspI s bv2 dclist2* Θ *dc tc* $\mathcal{B}$ Γ *v tv ba z*)
thm *Typing.infer-v-conspI*
have Θ ; {||} ; $\Gamma[bv::=b]_{\Gamma b} \vdash$ *V-consp s dc* $(ba[bv::=b]_{bb})$ $(v[bv::=b]_{vb}) \Rightarrow \{\!|\ z : B\text{-}app\ s\ (ba[bv::=b]_{bb}) \mid [\ [\ z\ ]^v\ ]^{ce} == [\ V\text{-}consp\ s\ dc\ (ba[bv::=b]_{bb})\ (v[bv::=b]_{vb})\ ]^{ce}\ |\!\}$
proof(*rule Typing.infer-v-conspI*)

show *AF-typedef-poly s bv2 dclist2* $\in$ *set* Θ **using** *infer-v-conspI* **by** *auto*
show (*dc*, *tc*) $\in$ *set dclist2* **using** *infer-v-conspI* **by** *auto*
show Θ ; {||} ; $\Gamma[bv::=b]_{\Gamma b} \vdash v[bv::=b]_{vb} \Rightarrow tv[bv::=b]_{\tau b}$
using *infer-v-conspI subst-tb.simps* **by** *metis*
find-theorems *fresh*
show Θ ; {||} ; $\Gamma[bv::=b]_{\Gamma b} \vdash tv[bv::=b]_{\tau b} \lesssim tc[bv2::=ba[bv::=b]_{bb}]_{\tau b}$ **proof** −
have *supp tc* $\subseteq$ { *atom bv2* } **using** *infer-v-conspI wfTh-poly-lookup-supp wfX-wfY* **by** *metis*
moreover have *bv2* $\neq$ *bv* **using** ‹*atom bv2* $\sharp$ $\mathcal{B}$› ‹$\mathcal{B} = \{|bv|\}$ › *fresh-at-base fresh-def*
using *fresh-finsert* **by** *fastforce*
ultimately have *atom bv* $\sharp$ *tc* **unfolding** *fresh-def* **by** *auto*
hence $tc[bv2::=ba[bv::=b]_{bb}]_{\tau b} = tc[bv2::=ba]_{\tau b}[bv::=b]_{\tau b}$
using *subst-tb-commute* **by** *metis*
moreover have Θ ; {||} ; $\Gamma[bv::=b]_{\Gamma b} \vdash tv[bv::=b]_{\tau b} \lesssim tc[bv2::=ba]_{\tau b}[bv::=b]_{\tau b}$

using *infer-v-conspI*(7) *subst-b-subtype infer-v-conspI* **by** *metis*
ultimately show *?thesis* **by** *auto*
qed

show *atom z* ♯ (Θ, {||}, Γ[*bv*::=*b*]$_{\Gamma b}$, *v*[*bv*::=*b*]$_{vb}$, *ba*[*bv*::=*b*]$_{bb}$)
apply(*unfold fresh-prodN*, *intro conjI*, *auto simp add*: *infer-v-conspI fresh-empty-fset*)
using ‹*atom z* ♯ Γ› *fresh-subst-if* *subst-b-*Γ*-def x-fresh-b* **apply** *metis*
using ‹*atom z* ♯ *v*› *fresh-subst-if* *subst-b-v-def x-fresh-b* **by** *metis*
show *atom bv2* ♯ (Θ, {||}, Γ[*bv*::=*b*]$_{\Gamma b}$, *v*[*bv*::=*b*]$_{vb}$, *ba*[*bv*::=*b*]$_{bb}$)
apply(*unfold fresh-prodN*, *intro conjI*, *auto simp add*: *infer-v-conspI fresh-empty-fset*)
using ‹*atom bv2* ♯ *b*› ‹*atom bv2* ♯ Γ› *fresh-subst-if* *subst-b-*Γ*-def* **apply** *metis*
using ‹*atom bv2* ♯ *b*› ‹*atom bv2* ♯ *v*› *fresh-subst-if* *subst-b-v-def* **apply** *metis*
using ‹*atom bv2* ♯ *b*› ‹*atom bv2* ♯ *ba*› *fresh-subst-if* *subst-b-b-def* **by** *metis*
show Θ ; {||} ⊢$_{wf}$ *ba*[*bv*::=*b*]$_{bb}$
using *infer-v-conspI* *wf-b-subst* **by** *metis*
qed
thus *?case* **using** *subst-vb.simps subst-tb.simps subst-bb.simps* **by** *simp*

qed

lemma *subst-b-check-v*:
fixes *v*::*v* **and** *b*::*b*
assumes Θ ; *B* ; *G* ⊢ *v* ⇐ τ **and** Θ ; {||} ⊢$_{wf}$ *b* **and** *B* = {|*bv*|}
shows Θ ; {||} ; *G*[*bv*::=*b*]$_{\Gamma b}$ ⊢ *v*[*bv*::=*b*]$_{vb}$ ⇐ (τ[*bv*::=*b*]$_{\tau b}$)
proof −
obtain τ′ **where** Θ ; *B* ; *G* ⊢ *v* ⇒ τ′ ∧ Θ ; *B* ; *G* ⊢ τ′ ≲ τ **using** *check-v-elims*[*OF assms*(*1*)] **by** *metis*
thus *?thesis* **using** *subst-b-subtype subst-b-infer-v assms*
by (*metis* (*no-types*) *check-v-subtypeI subst-b-infer-v subst-b-subtype*)
qed

lemma *subst-vv-subst-vb-switch*:
shows (*v*′[*bv*::=*b*′]$_{vb}$)[*x*::=*v*[*bv*::=*b*′]$_{vb}$]$_{vv}$ = *v*′[*x*::=*v*]$_{vv}$[*bv*::=*b*′]$_{vb}$
proof(*nominal-induct v*′ *rule*:*v.strong-induct*)
case (*V-lit x*)
then show *?case* **using** *subst-vv.simps subst-vb.simps* **by** *auto*
next
case (*V-var x*)
then show *?case* **using** *subst-vv.simps subst-vb.simps* **by** *auto*
next
case (*V-pair x1a x2a*)
then show *?case* **using** *subst-vv.simps subst-vb.simps v.fresh* **by** *auto*
next
case (*V-cons x1a x2a x3*)
then show *?case* **using** *subst-vv.simps subst-vb.simps v.fresh* **by** *auto*
next
case (*V-consp x1a x2a x3 x4*)
then show *?case* **using** *subst-vv.simps subst-vb.simps v.fresh pure-fresh*
by (*metis forget-subst subst-b-b-def*)
qed

lemma *subst-cev-subst-vb-switch*:

shows $(ce[bv::=b']_{ceb})[x::=v[bv::=b']_{vb}]_{cev} = (ce[x::=v]_{cev})[bv::=b']_{ceb}$
by(*nominal-induct ce rule*:*ce.strong-induct*, *auto simp add*: *subst-vv-subst-vb-switch ce.fresh*)

lemma *subst-cv-subst-vb-switch*:
shows $(c[bv::=b']_{cb})[x::=v[bv::=b']_{vb}]_{cv} = c[x::=v]_{cv}[bv::=b']_{cb}$
by(*nominal-induct c rule*:*c.strong-induct*, *auto simp add*: *subst-cev-subst-vb-switch c.fresh*)

lemma *subst-tv-subst-vb-switch*:
shows $(\tau[bv::=b']_{\tau b})[x::=v[bv::=b']_{vb}]_{\tau v} = \tau[x::=v]_{\tau v}[bv::=b']_{\tau b}$
proof(*nominal-induct* τ *avoiding*: *x v rule*:τ.*strong-induct*)
case (*T-refined-type z b c*)
hence *ceq*: $(c[bv::=b']_{cb})[x::=v[bv::=b']_{vb}]_{cv} = c[x::=v]_{cv}[bv::=b']_{cb}$ **using** *subst-cv-subst-vb-switch* **by** *auto*

moreover have *atom z* $\sharp$ $v[bv::=b']_{vb}$ **using** *x-fresh-b fresh-subst-if subst-b-v-def T-refined-type* **by** *metis*

hence $\{\!|\ z : b\ |\ c\ |\!\}[bv::=b']_{\tau b}[x::=v[bv::=b']_{vb}]_{\tau v} = \{\!|\ z : b[bv::=b']_{bb}\ |\ (c[bv::=b']_{cb})[x::=v[bv::=b']_{vb}]_{cv}\ |\!\}$
using *subst-tv.simps subst-tb.simps T-refined-type fresh-Pair* **by** *metis*

moreover have $\{\!|\ z : b[bv::=b']_{bb}\ |\ (c[bv::=b']_{cb})[x::=v[bv::=b']_{vb}]_{cv}\ |\!\} = \{\!|\ z : b\ |\ c[x::=v]_{cv}\ |\!\}[bv::=b']_{\tau b}$
using *subst-tv.simps subst-tb.simps ceq* τ.*fresh forget-subst*[*of bv b b'*] *subst-b-b-def T-refined-type* **by** *metis*

ultimately show *?case* **using** *subst-tv.simps subst-tb.simps ceq T-refined-type* **by** *auto*
qed

lemma *subst-tb-triple*:
assumes *atom bv* $\sharp$ τ'
shows $\tau'[bv'::=b'[bv::=b]_{bb}]_{\tau b}[x'::=v'[bv::=b]_{vb}]_{\tau v} = \tau'[bv'::=b']_{\tau b}[x'::=v']_{\tau v}[bv::=b]_{\tau b}$
proof −
have $\tau'[bv'::=b'[bv::=b]_{bb}]_{\tau b}[x'::=v'[bv::=b]_{vb}]_{\tau v} = \tau'[bv'::=b']_{\tau b}[bv::=b]_{\tau b}\ [x'::=v'[bv::=b]_{vb}]_{\tau v}$
using *subst-tb-commute* ‹*atom bv* $\sharp$ τ'› **by** *auto*
also have ... $= \tau'[bv'::=b']_{\tau b}\ [x'::=v']_{\tau v}[bv::=b]_{\tau b}$
using *subst-tv-subst-vb-switch* **by** *auto*
finally show *?thesis* **using** *fresh-subst-if forget-subst* **by** *auto*
qed

lemma *subst-b-infer-e*:
fixes *s*::*s* **and** *b*::*b*
assumes Θ ; Φ ; B ; G; $D \vdash e \Rightarrow \tau$ **and** Θ ; $\{||\} \vdash_{wf} b$ **and** $B = \{|bv|\}$
shows Θ ; Φ ; $\{||\}$; $G[bv::=b]_{\Gamma b}$; $D[bv::=b]_{\Delta b} \vdash (e[bv::=b]_{eb}) \Rightarrow (\tau[bv::=b]_{\tau b})$
using *assms* **proof**(*nominal-induct avoiding*: *b rule*: *infer-e.strong-induct*)
case (*infer-e-valI* Θ $\mathcal{B}$ Γ Δ Φ *v* τ)
thus *?case* **using** *subst-eb.simps infer-e.intros wf-b-subst subst-db.simps wf-b-subst infer-v-wf subst-b-infer-v*

by (*metis forget-subst ms-fresh-all*(*1*) *wfV-b-fresh*)
next
case (*infer-e-plusI* Θ $\mathcal{B}$ Γ Δ Φ *v1 z1 c1 v2 z2 c2 z3*)

thm *wf-b-subst(15)*
show *?case* **unfolding** *subst-b-simps subst-eb.simps* **proof**(*rule Typing.infer-e-plusI*)
show Θ ; $\{||\}$; $\Gamma[bv::=b]_{\Gamma b} \vdash_{wf} \Delta[bv::=b]_{\Delta b}$ **using** *wf-b-subst(10) subst-db.simps infer-e-plusI wfX-wfY*
by (*metis wf-b-subst(15)*)
show $\Theta \vdash_{wf} \Phi$ **using** *infer-e-plusI* **by** *auto*
show Θ ; $\{||\}$; $\Gamma[bv::=b]_{\Gamma b} \vdash v1[bv::=b]_{vb} \Rightarrow \{\!|\ z1 : B\text{-}int \mid c1[bv::=b]_{cb}\ |\!\}$ **using** *subst-b-infer-v infer-e-plusI subst-b-simps* **by** *force*
show Θ ; $\{||\}$; $\Gamma[bv::=b]_{\Gamma b} \vdash v2[bv::=b]_{vb} \Rightarrow \{\!|\ z2 : B\text{-}int \mid c2[bv::=b]_{cb}\ |\!\}$ **using** *subst-b-infer-v infer-e-plusI subst-b-simps* **by** *force*
show *atom z3* $\sharp$ *AE-op Plus* $(v1[bv::=b]_{vb})$ $(v2[bv::=b]_{vb})$ **using** *subst-b-simps infer-e-plusI subst-b-fresh-x subst-b-e-def* **by** *metis*
show *atom z3* $\sharp$ $\Gamma[bv::=b]_{\Gamma b}$ **using** *subst-g-b-x-fresh infer-e-plusI* **by** *auto*
qed
next
case (*infer-e-leqI* Θ $\mathcal{B}$ Γ Δ Φ *v1 z1 c1 v2 z2 c2 z3*)
show *?case* **unfolding** *subst-b-simps* **proof**(*rule Typing.infer-e-leqI*)
show Θ ; $\{||\}$; $\Gamma[bv::=b]_{\Gamma b} \vdash_{wf} \Delta[bv::=b]_{\Delta b}$ **using** *wf-b-subst(10) subst-db.simps infer-e-leqI wfX-wfY*
by (*metis wf-b-subst(15)*)
show $\Theta \vdash_{wf} \Phi$ **using** *infer-e-leqI* **by** *auto*
show Θ ; $\{||\}$; $\Gamma[bv::=b]_{\Gamma b} \vdash v1[bv::=b]_{vb} \Rightarrow \{\!|\ z1 : B\text{-}int \mid c1[bv::=b]_{cb}\ |\!\}$ **using** *subst-b-infer-v infer-e-leqI subst-b-simps* **by** *force*
show Θ ; $\{||\}$; $\Gamma[bv::=b]_{\Gamma b} \vdash v2[bv::=b]_{vb} \Rightarrow \{\!|\ z2 : B\text{-}int \mid c2[bv::=b]_{cb}\ |\!\}$ **using** *subst-b-infer-v infer-e-leqI subst-b-simps* **by** *force*
show *atom z3* $\sharp$ *AE-op LEq* $(v1[bv::=b]_{vb})$ $(v2[bv::=b]_{vb})$ **using** *subst-b-simps infer-e-leqI subst-b-fresh-x subst-b-e-def* **by** *metis*
show *atom z3* $\sharp$ $\Gamma[bv::=b]_{\Gamma b}$ **using** *subst-g-b-x-fresh infer-e-leqI* **by** *auto*
qed
next
case (*infer-e-appI* Θ $\mathcal{B}$ Γ Δ Φ *f x b' c* τ' *s' v* τ)
show *?case* **proof**(*subst subst-eb.simps*, *rule Typing.infer-e-appI*)
show Θ ; $\{||\}$; $\Gamma[bv::=b]_{\Gamma b} \vdash_{wf} \Delta[bv::=b]_{\Delta b}$ **using** *wf-b-subst(10) subst-db.simps infer-e-appI wfX-wfY* **by** (*metis wf-b-subst(15)*)
show $\Theta \vdash_{wf} \Phi$ **using** *infer-e-appI* **by** *auto*
show *Some* (*AF-fundef f* (*AF-fun-typ-none* (*AF-fun-typ x b' c* τ' *s'*))) = *lookup-fun* Φ *f* **using** *infer-e-appI* **by** *auto*

have *atom bv* $\sharp$ *b'* **using** $\langle \Theta \vdash_{wf} \Phi \rangle$ *infer-e-appI wfPhi-f-supp fresh-def*[*of atom bv b'*] **by** *simp*
hence $b' = b'[bv::=b]_{bb}$ **using** *subst-b-simps*
using *has-subst-b-class.forget-subst subst-b-b-def* **by** *force*
moreover have *ceq*:$c = c[bv::=b]_{cb}$ **using** *subst-b-simps* **proof** −
have *atom bv* $\sharp$ *c* **using** *infer-e-appI wfPhi-f-supp-c*[*OF infer-e-appI*(*3*) $\langle \Theta \vdash_{wf} \Phi \rangle$] *fresh-def*[*of atom bv c*]
using *fresh-def fresh-finsert insert-absorb insert-subset ms-fresh-all supp-at-base x-not-in-b-set* **by** *metis*
thus *?thesis*
using *forget-subst subst-b-c-def fresh-def*[*of atom bv c*] **by** *metis*
qed
show Θ ; $\{||\}$; $\Gamma[bv::=b]_{\Gamma b} \vdash v[bv::=b]_{vb} \Leftarrow \{\!|\ x : b' \mid c\ |\!\}$ **using** *subst-b-check-v subst-tb.simps subst-vb.simps infer-e-appI*
proof −

```
    have Θ ; {|bv|} ; Γ ⊢ v ⇐ {| x : b' | c |}
      by (metis ‹B = {|bv|}› ‹Θ ; B ; Γ ⊢ v ⇐ {| x : b' | c |}›)
    then show ?thesis
      by (metis (no-types) ‹Θ ; {||} ⊢wf b› ‹b' = b'[bv::=b]bb› subst-b-check-v subst-tb.simps ceq)
  qed
  show atom x ♯ Γ[bv::=b]Γb using subst-g-b-x-fresh infer-e-appI by auto
  have supp τ' ⊆ { atom x } using wfPhi-f-simple-supp-t infer-e-appI by auto
  hence atom bv ♯ τ' using fresh-def fresh-at-base by force
  then show τ'[x::=v[bv::=b]vb]v = τ[bv::=b]τb using infer-e-appI (6) forget-subst subst-b-τ-def
subst-tv-subst-vb-switch subst-defs by metis
 qed
next
 case (infer-e-appPI Θ' B Γ' Δ Φ' b' f' bv' x' b1 c τ' s' v' τ1)

 have beq: b1[bv'::=b']bb[bv::=b]bb = b1[bv'::=b'[bv::=b]bb]bb
 proof −
  have supp b1 ⊆ { atom bv' } using wfPhi-f-poly-supp-b infer-e-appPI
    using supp-at-base by blast
  moreover have bv ≠ bv' using infer-e-appPI fresh-def supp-at-base
    by (simp add: fresh-def supp-at-base)
  ultimately have atom bv ♯ b1 using fresh-def fresh-at-base by force
  thus ?thesis by simp
 qed

 have ceq: (c[bv'::=b']cb)[bv::=b]cb = c[bv'::=b'[bv::=b]bb]cb proof −
  have supp c ⊆ { atom bv', atom x' } using wfPhi-f-poly-supp-c infer-e-appPI
    using supp-at-base by blast
  moreover have bv ≠ bv' using infer-e-appPI fresh-def supp-at-base
    by (simp add: fresh-def supp-at-base)
  moreover have atom x' ≠ atom bv by auto
  ultimately have atom bv ♯ c using fresh-def[of atom bv c] fresh-at-base by auto
  thus ?thesis by simp
 qed

 show ?case proof(subst subst-eb.simps, rule Typing.infer-e-appPI)
  show Θ' ; {||} ; Γ'[bv::=b]Γb ⊢wf Δ[bv::=b]Δb using wf-b-subst subst-db.simps infer-e-appPI wfX-wfY
by metis
  show Θ' ⊢wf Φ' using infer-e-appPI by auto
  show Some (AF-fundef f' (AF-fun-typ-some bv' (AF-fun-typ x' b1 c τ' s'))) = lookup-fun Φ' f'
using infer-e-appPI by auto
  thus Θ' ; {||} ; Γ'[bv::=b]Γb ⊢ v'[bv::=b]vb ⇐ {| x' : b1[bv'::=b'[bv::=b]bb]b | c[bv'::=b'[bv::=b]bb]b
|}
    using subst-b-check-v subst-tb.simps subst-b-simps infer-e-appPI
  proof −
   have Θ' ; {||} ; Γ'[bv::=b]Γb ⊢ v'[bv::=b]vb ⇐ {| x' : b1[bv'::=b']b[bv::=b]bb | (c[bv'::=b']b)[bv::=b]cb
|}
     using infer-e-appPI subst-b-check-v subst-tb.simps by metis
   thus ?thesis using beq ceq subst-defs by metis
  qed
  show atom x' ♯ Γ'[bv::=b]Γb using subst-g-b-x-fresh infer-e-appPI by auto
  show τ'[bv'::=b'[bv::=b]bb]b[x'::=v'[bv::=b]vb]v = τ1[bv::=b]τb proof −
```

have *supp* $\tau' \subseteq \{$ *atom* x', *atom* bv' $\}$ **using** *wfPhi-f-poly-supp-t infer-e-appPI* **by** *auto*
moreover hence $bv \neq bv'$ **using** *infer-e-appPI fresh-def supp-at-base*
by (*simp add*: *fresh-def supp-at-base*)
ultimately have *atom* $bv \sharp \tau'$ **using** *fresh-def* **by** *force*
hence $\tau'[bv'::=b'[bv::=b]_{bb}]_b[x'::=v'[bv::=b]_{vb}]_v = \tau'[bv'::=b']_b[x'::=v']_v[bv::=b]_{\tau b}$ **using** *subst-tb-triple subst-defs* **by** *auto*
thus *?thesis* **using** *infer-e-appPI* **by** *metis*
qed
show *atom* $bv' \sharp (\Theta', \Phi', \{||\}, \Gamma'[bv::=b]_{\Gamma b}, \Delta[bv::=b]_{\Delta b}, b'[bv::=b]_{bb}, v'[bv::=b]_{vb}, \tau 1[bv::=b]_{\tau b})$
unfolding *fresh-prodN* **apply**(*auto simp add*: *infer-e-appPI fresh-empty-fset*)
using *fresh-subst-if subst-b-Γ-def subst-b-Δ-def subst-b-b-def subst-b-v-def subst-b-τ-def infer-e-appPI* **by** *metis+*
show Θ' ; $\{||\} \vdash_{wf} b'[bv::=b]_{bb}$ **using** *infer-e-appPI wf-b-subst* **by** *simp*
qed
next
case (*infer-e-fstI* Θ $\mathcal{B}$ Γ Δ Φ v z' *b1 b2 c z*)
show *?case* **unfolding** *subst-b-simps* **proof**(*rule Typing.infer-e-fstI*)
show Θ ; $\{||\}$; $\Gamma[bv::=b]_{\Gamma b} \vdash_{wf} \Delta[bv::=b]_{\Delta b}$ **using** *wf-b-subst(10) subst-db.simps infer-e-fstI wfX-wfY*
by (*metis wf-b-subst(15)*)
show $\Theta \vdash_{wf} \Phi$ **using** *infer-e-fstI* **by** *auto*
show Θ ; $\{||\}$; $\Gamma[bv::=b]_{\Gamma b} \vdash v[bv::=b]_{vb} \Rightarrow \{\!|\ z' : B\text{-}pair\ b1[bv::=b]_{bb}\ b2[bv::=b]_{bb}\ |\ c[bv::=b]_{cb}\ |\!\}$
using *subst-b-infer-v subst-tb.simps subst-b-simps infer-e-fstI* **by** *force*
show *atom* $z \sharp$ *AE-fst* $(v[bv::=b]_{vb})$ **using** *infer-e-fstI subst-b-fresh-x subst-b-v-def e.fresh* **by** *metis*
show *atom* $z \sharp \Gamma[bv::=b]_{\Gamma b}$ **using** *subst-g-b-x-fresh infer-e-fstI* **by** *auto*
qed
next
case (*infer-e-sndI* Θ $\mathcal{B}$ Γ Δ Φ v z' *b1 b2 c z*)
show *?case* **unfolding** *subst-b-simps* **proof**(*rule Typing.infer-e-sndI*)
show Θ ; $\{||\}$; $\Gamma[bv::=b]_{\Gamma b} \vdash_{wf} \Delta[bv::=b]_{\Delta b}$ **using** *wf-b-subst(10) subst-db.simps infer-e-sndI wfX-wfY*
by (*metis wf-b-subst(15)*)
show $\Theta \vdash_{wf} \Phi$ **using** *infer-e-sndI* **by** *auto*
show Θ ; $\{||\}$; $\Gamma[bv::=b]_{\Gamma b} \vdash v[bv::=b]_{vb} \Rightarrow \{\!|\ z' : B\text{-}pair\ b1[bv::=b]_{bb}\ b2[bv::=b]_{bb}\ |\ c[bv::=b]_{cb}\ |\!\}$
using *subst-b-infer-v subst-tb.simps subst-b-simps infer-e-sndI* **by** *force*
show *atom* $z \sharp$ *AE-snd* $(v[bv::=b]_{vb})$ **using** *infer-e-sndI subst-b-fresh-x subst-b-v-def e.fresh* **by** *metis*
show *atom* $z \sharp \Gamma[bv::=b]_{\Gamma b}$ **using** *subst-g-b-x-fresh infer-e-sndI* **by** *auto*
qed
next
case (*infer-e-lenI* Θ $\mathcal{B}$ Γ Δ Φ v z' c z)
show *?case* **unfolding** *subst-b-simps* **proof**(*rule Typing.infer-e-lenI*)
show Θ ; $\{||\}$; $\Gamma[bv::=b]_{\Gamma b} \vdash_{wf} \Delta[bv::=b]_{\Delta b}$ **using** *wf-b-subst(10) subst-db.simps infer-e-lenI wfX-wfY*
by (*metis wf-b-subst(15)*)
show $\Theta \vdash_{wf} \Phi$ **using** *infer-e-lenI* **by** *auto*
show Θ ; $\{||\}$; $\Gamma[bv::=b]_{\Gamma b} \vdash v[bv::=b]_{vb} \Rightarrow \{\!|\ z' : B\text{-}bitvec\ |\ c[bv::=b]_{cb}\ |\!\}$
using *subst-b-infer-v subst-tb.simps subst-b-simps infer-e-lenI* **by** *force*
show *atom* $z \sharp$ *AE-len* $(v[bv::=b]_{vb})$ **using** *infer-e-lenI subst-b-fresh-x subst-b-v-def e.fresh* **by** *metis*
show *atom* $z \sharp \Gamma[bv::=b]_{\Gamma b}$ **using** *subst-g-b-x-fresh infer-e-lenI* **by** *auto*
qed
next
case (*infer-e-mvarI* Θ $\mathcal{B}$ Γ Φ Δ u τ)

show *?case* **proof**(*subst subst subst-eb.simps*, *rule Typing.infer-e-mvarI*)
show Θ ; $\{||\}$ $\vdash_{wf}$ $\Gamma[bv::=b]_{\Gamma b}$ **using** *infer-e-mvarI wf-b-subst* **by** *auto*
show Θ $\vdash_{wf}$ Φ **using** *infer-e-mvarI* **by** *auto*
show Θ ; $\{||\}$; $\Gamma[bv::=b]_{\Gamma b}$ $\vdash_{wf}$ $\Delta[bv::=b]_{\Delta b}$ **using** *infer-e-mvarI* **using** *wf-b-subst*(*10*) *subst-db.simps infer-e-sndI wfX-wfY*
by (*metis wf-b-subst*(*15*))
show $(u, \tau[bv::=b]_{\tau b}) \in$ *setD* $\Delta[bv::=b]_{\Delta b}$ **using** *infer-e-mvarI subst-db.simps set-insert subst-d-b-member* **by** *simp*
qed
next
case (*infer-e-concatI* Θ $\mathcal{B}$ Γ Δ Φ *v1 z1 c1 v2 z2 c2 z3*)
show *?case* **unfolding** *subst-b-simps* **proof**(*rule Typing.infer-e-concatI*)
show Θ ; $\{||\}$; $\Gamma[bv::=b]_{\Gamma b}$ $\vdash_{wf}$ $\Delta[bv::=b]_{\Delta b}$ **using** *wf-b-subst*(*10*) *subst-db.simps infer-e-concatI wfX-wfY*
by (*metis wf-b-subst*(*15*))
show Θ $\vdash_{wf}$ Φ **using** *infer-e-concatI* **by** *auto*
show Θ ; $\{||\}$; $\Gamma[bv::=b]_{\Gamma b}$ $\vdash$ *v1*$[bv::=b]_{vb}$ $\Rightarrow$ $\{\!|$ *z1* : *B-bitvec* | *c1*$[bv::=b]_{cb}$ $|\!\}$
using *subst-b-infer-v subst-tb.simps subst-b-simps infer-e-concatI* **by** *force*
show Θ ; $\{||\}$; $\Gamma[bv::=b]_{\Gamma b}$ $\vdash$ *v2*$[bv::=b]_{vb}$ $\Rightarrow$ $\{\!|$ *z2* : *B-bitvec* | *c2*$[bv::=b]_{cb}$ $|\!\}$
using *subst-b-infer-v subst-tb.simps subst-b-simps infer-e-concatI* **by** *force*
show *atom z3* $\sharp$ *AE-concat* (*v1*$[bv::=b]_{vb}$) (*v2*$[bv::=b]_{vb}$) **using** *infer-e-concatI subst-b-fresh-x subst-b-v-def e.fresh* **by** *metis*
show *atom z3* $\sharp$ $\Gamma[bv::=b]_{\Gamma b}$ **using** *subst-g-b-x-fresh infer-e-concatI* **by** *auto*
qed
next
case (*infer-e-splitI* Θ $\mathcal{B}$ Γ Δ Φ *v1 z1 c1 v2 z2 z3*)
show *?case* **unfolding** *subst-b-simps* **proof**(*rule Typing.infer-e-splitI*)
show ‹ Θ ; $\{||\}$; $\Gamma[bv::=b]_{\Gamma b}$ $\vdash_{wf}$ $\Delta[bv::=b]_{\Delta b}$ › **using** *wf-b-subst*(*10*) *subst-db.simps infer-e-splitI wfX-wfY*
by (*metis wf-b-subst*(*15*))
show ‹ Θ $\vdash_{wf}$ Φ › **using** *infer-e-splitI* **by** *auto*
show ‹ Θ ; $\{||\}$; $\Gamma[bv::=b]_{\Gamma b}$ $\vdash$ *v1*$[bv::=b]_{vb}$ $\Rightarrow$ $\{\!|$ *z1* : *B-bitvec* | *c1*$[bv::=b]_{cb}$ $|\!\}$ ›
using *subst-b-infer-v subst-tb.simps subst-b-simps infer-e-splitI* **by** *force*
show ‹Θ ; $\{||\}$; $\Gamma[bv::=b]_{\Gamma b}$ $\vdash$ *v2*$[bv::=b]_{vb}$ $\Leftarrow$ $\{\!|$ *z2* : *B-int* | [*leq* [[*L-num 0*]^v]^{ce} [[*z2*]^v]^{ce}]^{ce} == [[*L-true*]^v]^{ce} *AND* [*leq* [[*z2*]^v]^{ce} [| [*v1*$[bv::=b]_{vb}$]^{ce} |]^{ce}]^{ce} == [[*L-true*]^v]^{ce} $|\!\}$ ›
using *subst-b-check-v subst-tb.simps subst-b-simps infer-e-splitI*
proof −
have Θ ; $\{||\}$; $\Gamma[bv::=b]_{\Gamma b}$ $\vdash$ *v2*$[bv::=b]_{vb}$ $\Leftarrow$ $\{\!|$ *z2* : *B-int* | [*leq* [[*L-num 0*]^v]^{ce} [[*z2*]^v]^{ce}]^{ce} == [[*L-true*]^v]^{ce} *AND* [*leq* [[*z2*]^v]^{ce} [| [*v1*]^{ce} |]^{ce}]^{ce} == [[*L-true*]^v]^{ce} $|\!\}[bv::=b]_{\tau b}$
using *infer-e-splitI.hyps*(*7*) *infer-e-splitI.prems*(*1*) *infer-e-splitI.prems*(*2*) *subst-b-check-v* **by** *presburger*
then show *?thesis*
by *simp*
qed
show ‹*atom z1* $\sharp$ *AE-split* (*v1*$[bv::=b]_{vb}$) (*v2*$[bv::=b]_{vb}$)› **using** *infer-e-splitI subst-b-fresh-x subst-b-v-def e.fresh* **by** *metis*
show ‹*atom z1* $\sharp$ $\Gamma[bv::=b]_{\Gamma b}$› **using** *subst-g-b-x-fresh infer-e-splitI* **by** *auto*

show ‹*atom z2* $\sharp$ *AE-split* (*v1*$[bv::=b]_{vb}$) (*v2*$[bv::=b]_{vb}$)› **using** *infer-e-splitI subst-b-fresh-x subst-b-v-def e.fresh* **by** *metis*

show ‹*atom z2* ♯ Γ[*bv*::=*b*]$_{\Gamma b}$› **using** *subst-g-b-x-fresh infer-e-splitI* **by** *auto*
show ‹*atom z3* ♯ *AE-split* (*v1*[*bv*::=*b*]$_{vb}$) (*v2*[*bv*::=*b*]$_{vb}$)› **using** *infer-e-splitI subst-b-fresh-x subst-b-v-def e.fresh* **by** *metis*
show ‹*atom z3* ♯ Γ[*bv*::=*b*]$_{\Gamma b}$› **using** *subst-g-b-x-fresh infer-e-splitI* **by** *auto*
qed
qed

lemma *subst-b-c-of-forget*:
assumes *atom bv* ♯ *const*
shows (*c-of const x*)[*bv*::=*b*]$_{cb}$ = *c-of const x*
using *assms* **proof**(*nominal-induct const avoiding*: *x rule*:τ.*strong-induct*)
case (*T-refined-type x' b' c'*)
hence *c-of* {| *x'* : *b'* | *c'* |} *x* = *c'*[*x'*::=*V-var x*]$_{cv}$ **using** *c-of.simps* **by** *metis*
moreover have *atom bv* ♯ *c'*[*x'*::=*V-var x*]$_{cv}$ **proof** −
have *atom bv* ♯ *c'* **using** *T-refined-type* τ.*fresh* **by** *simp*
moreover have *atom bv* ♯ *V-var x* **using** *v.fresh* **by** *simp*
ultimately show *?thesis*
using *T-refined-type* τ.*fresh subst-b-c-def fresh-subst-if* τ*-fresh-c fresh-subst-cv-if has-subst-b-class.subst-b-fresh-x ms-fresh-all*(*37*) *ms-fresh-all assms* **by** *metis*
qed
ultimately show *?case* **using** *forget-subst subst-b-c-def* **by** *metis*
qed

lemma *subst-b-check-s*:
fixes *s*::*s* **and** *b*::*b* **and** *cs*::*branch-s* **and** *css*::*branch-list* **and** *v*::*v* **and** τ::τ
assumes Θ ; {||} ⊢$_{wf}$ *b* **and** *B* = {|*bv*|}
shows Θ ; Φ ; *B* ; *G*; *D* ⊢ *s* ⇐ τ ⟹ Θ ; Φ ; {||} ; *G*[*bv*::=*b*]$_{\Gamma b}$; *D*[*bv*::=*b*]$_{\Delta b}$ ⊢ (*s*[*bv*::=*b*]$_{sb}$) ⇐ (τ[*bv*::=*b*]$_{\tau b}$) **and**
Θ ; Φ ; *B* ; *G*; *D* ; *tid* ; *cons* ; *const* ; *v* ⊢ *cs* ⇐ τ ⟹ Θ ; Φ ; {||} ; *G*[*bv*::=*b*]$_{\Gamma b}$; *D*[*bv*::=*b*]$_{\Delta b}$; *tid* ; *cons* ; *const* ; *v*[*bv*::=*b*]$_{vb}$ ⊢ (*subst-branchb cs bv b*) ⇐ (τ[*bv*::=*b*]$_{\tau b}$) **and**
Θ ; Φ ; *B* ; *G*; *D* ; *tid* ; *dclist* ; *v* ⊢ *css* ⇐ τ ⟹ Θ ; Φ ; {||} ; *G*[*bv*::=*b*]$_{\Gamma b}$; *D*[*bv*::=*b*]$_{\Delta b}$; *tid* ; *dclist* ; *v*[*bv*::=*b*]$_{vb}$ ⊢ (*subst-branchlb css bv b*) ⇐ (τ[*bv*::=*b*]$_{\tau b}$)
using *assms* **proof**(*induct rule*: *check-s-check-branch-s-check-branch-list.inducts*)
note *facts* = *wfD-emptyI wfX-wfY wf-b-subst subst-b-subtype subst-b-infer-v*
case (*check-valI* Θ $\mathcal{B}$ Γ Δ Φ *v* τ' τ)
show *?case*
apply(*subst subst-sb.simps*, *rule Typing.check-valI*)
using *facts check-valI* **apply** *metis*
using *check-valI subst-b-infer-v wf-b-subst subst-b-subtype* **apply** *blast*
using *check-valI subst-b-infer-v wf-b-subst subst-b-subtype* **apply** *blast*
using *check-valI subst-b-infer-v wf-b-subst subst-b-subtype* **by** *metis*
next

case (*check-letI x* Θ Φ $\mathcal{B}$ Γ Δ *e* τ *z s b' c*)
show *?case* **proof**(*subst subst-sb.simps*, *rule Typing.check-letI*)

show *atom x* ♯(Θ, Φ, {||}, Γ[*bv*::=*b*]$_{\Gamma b}$, Δ[*bv*::=*b*]$_{\Delta b}$, *e*[*bv*::=*b*]$_{eb}$, τ[*bv*::=*b*]$_{\tau b}$)
apply(*unfold fresh-prodN*,*auto*)
apply(*simp add*: *check-letI fresh-empty-fset*)+

apply(*metis* ∗ *subst-b-fresh-x check-letI fresh-prodN*)+ **done**
show *atom z* ♯ (*x*, Θ, Φ, {||}, Γ[*bv*::=*b*]$_{\Gamma b}$, Δ[*bv*::=*b*]$_{\Delta b}$, *e*[*bv*::=*b*]$_{eb}$, τ[*bv*::=*b*]$_{\tau b}$, *s*[*bv*::=*b*]$_{sb}$)
apply(*unfold fresh-prodN*,*auto*)
apply(*simp add*: *check-letI fresh-empty-fset*)+
apply(*metis* ∗ *subst-b-fresh-x check-letI fresh-prodN*)+ **done**
show Θ ; Φ ; {||} ; Γ[*bv*::=*b*]$_{\Gamma b}$; Δ[*bv*::=*b*]$_{\Delta b}$ ⊢ *e*[*bv*::=*b*]$_{eb}$ ⇒ {‖ *z* : *b*′[*bv*::=*b*]$_{bb}$ | *c*[*bv*::=*b*]$_{cb}$ ‖}
using *check-letI subst-b-infer-e subst-tb.simps* **by** *metis*
have *c*[*z*::=[*x*]^{v}]$_{cv}$[*bv*::=*b*]$_{cb}$ = (*c*[*bv*::=*b*]$_{cb}$)[*z*::=*V-var x*]$_{cv}$
using *subst-cv-subst-bb-commute*[*of bv V-var x c z b*] *fresh-at-base* **by** *simp*
thus Θ ; Φ ; {||} ; ((*x*, *b*′[*bv*::=*b*]$_{bb}$, (*c*[*bv*::=*b*]$_{cb}$)[*z*::=*V-var x*]$_{v}$) #$_{\Gamma}$ Γ[*bv*::=*b*]$_{\Gamma b}$) ; Δ[*bv*::=*b*]$_{\Delta b}$ ⊢ *s*[*bv*::=*b*]$_{sb}$ ⇐ τ[*bv*::=*b*]$_{\tau b}$
using *check-letI subst-gb.simps subst-defs* **by** *metis*
qed
next
case (*check-assertI x* Θ Φ $\mathcal{B}$ Γ Δ *c* τ *s*)
show *?case* **proof**(*subst subst-sb.simps*, *rule Typing.check-assertI*)
show *atom x* ♯ (Θ, Φ, {||}, Γ[*bv*::=*b*]$_{\Gamma b}$, Δ[*bv*::=*b*]$_{\Delta b}$, *c*[*bv*::=*b*]$_{cb}$, τ[*bv*::=*b*]$_{\tau b}$, *s*[*bv*::=*b*]$_{sb}$)
apply(*unfold fresh-prodN*,*auto*)
apply(*simp add*: *check-assertI fresh-empty-fset*)+
apply(*metis* ∗ *subst-b-fresh-x check-assertI fresh-prodN*)+ **done**

have Θ ; Φ ; {||} ; ((*x*, *B-bool*, *c*) #$_{\Gamma}$ Γ)[*bv*::=*b*]$_{\Gamma b}$; Δ[*bv*::=*b*]$_{\Delta b}$ ⊢ *s*[*bv*::=*b*]$_{sb}$ ⇐ τ[*bv*::=*b*]$_{\tau b}$ **using** *check-assertI*
by *metis*
thus Θ ; Φ ; {||} ; (*x*, *B-bool*, *c*[*bv*::=*b*]$_{cb}$) #$_{\Gamma}$ Γ[*bv*::=*b*]$_{\Gamma b}$; Δ[*bv*::=*b*]$_{\Delta b}$ ⊢ *s*[*bv*::=*b*]$_{sb}$ ⇐ τ[*bv*::=*b*]$_{\tau b}$ **using** *subst-gb.simps* **by** *auto*
show Θ ; {||} ; Γ[*bv*::=*b*]$_{\Gamma b}$ ⊨ *c*[*bv*::=*b*]$_{cb}$ **using** *subst-b-valid check-assertI* **by** *simp*
show Θ ; {||} ; Γ[*bv*::=*b*]$_{\Gamma b}$ ⊢$_{wf}$ Δ[*bv*::=*b*]$_{\Delta b}$ **using** *wf-b-subst2*(*6*) *check-assertI* **by** *simp*
qed
next
case (*check-branch-list-consI* Θ Φ $\mathcal{B}$ Γ Δ *tid dclist v cs* τ *css*)
then show *?case* **unfolding** *subst-branchlb.simps* **using** *Typing.check-branch-list-consI* **by** *simp*
next
case (*check-branch-list-finalI* Θ Φ $\mathcal{B}$ Γ Δ *tid dclist v cs* τ)
then show *?case* **unfolding** *subst-branchlb.simps* **using** *Typing.check-branch-list-finalI* **by** *simp*
next
case (*check-branch-s-branchI* Θ $\mathcal{B}$ Γ Δ τ *const x* Φ *tid cons v s*)

show *?case* **unfolding** *subst-b-simps* **proof**(*rule Typing.check-branch-s-branchI*)
show Θ ; {||} ; Γ[*bv*::=*b*]$_{\Gamma b}$ ⊢$_{wf}$ Δ[*bv*::=*b*]$_{\Delta b}$ **using** *check-branch-s-branchI wf-b-subst subst-db.simps* **by** *metis*
show ⊢$_{wf}$ Θ **using** *check-branch-s-branchI* **by** *auto*
show Θ ; {||} ; Γ[*bv*::=*b*]$_{\Gamma b}$ ⊢$_{wf}$ τ[*bv*::=*b*]$_{\tau b}$ **using** *check-branch-s-branchI wf-b-subst* **by** *metis*

show *atom x* ♯(Θ, Φ, {||}, Γ[*bv*::=*b*]$_{\Gamma b}$, Δ[*bv*::=*b*]$_{\Delta b}$, *tid*, *cons* , *const*, *v*[*bv*::=*b*]$_{vb}$, τ[*bv*::=*b*]$_{\tau b}$)
apply(*unfold fresh-prodN*,*auto*)
apply(*simp add*: *check-branch-s-branchI fresh-empty-fset*)+
apply(*metis* ∗ *subst-b-fresh-x check-branch-s-branchI fresh-prodN*)+
done
show *wft*:Θ ; {||} ; *GNil* ⊢$_{wf}$ *const* **using** *check-branch-s-branchI* **by** *auto*
hence (*b-of const*) = (*b-of const*)[*bv*::=*b*]$_{bb}$
using *wfT-nil-supp fresh-def*[*of atom bv*] *forget-subst subst-b-b-def* τ.*supp*

bot.extremum-uniqueI ex-in-conv fresh-def supp-empty-fset
by (*metis b-of-supp*)
moreover have (*c-of const x*)$[bv::=b]_{cb}$ = *c-of const x*
using *wft wfT-nil-supp fresh-def*[*of atom bv*] *forget-subst subst-b-c-def* τ*.supp*
bot.extremum-uniqueI ex-in-conv fresh-def supp-empty-fset subst-b-c-of-forget **by** *metis*
ultimately show Θ ; Φ ; {||} ; (*x*, *b-of const*, *CE-val* ($v[bv::=b]_{vb}$) == *CE-val*(*V-cons tid cons* (*V-var x*)) *AND c-of const x*) $\#_\Gamma$ $\Gamma[bv::=b]_{\Gamma b}$; $\Delta[bv::=b]_{\Delta b}$ $\vdash$ $s[bv::=b]_{sb}$ $\Leftarrow$ $\tau[bv::=b]_{\tau b}$
using *check-branch-s-branchI subst-gb.simps* **by** *auto*

qed

next
case (*check-ifI z* Θ Φ $\mathcal{B}$ Γ Δ *v s1 s2* τ)
show *?case* **unfolding** *subst-b-simps* **proof**(*rule Typing.check-ifI*)
show ‹*atom z* $\sharp$ (Θ, Φ, {||}, $\Gamma[bv::=b]_{\Gamma b}$, $\Delta[bv::=b]_{\Delta b}$, $v[bv::=b]_{vb}$, *s1*$[bv::=b]_{sb}$, *s2*$[bv::=b]_{sb}$, $\tau[bv::=b]_{\tau b}$)›
by(*unfold fresh-prodN*,*auto*, *auto simp add*: *check-ifI fresh-empty-fset subst-b-fresh-x*)
have {| *z* : *B-bool* | *TRUE* |}$[bv::=b]_{\tau b}$ = {| *z* : *B-bool* | *TRUE* |} **by** *auto*
thus ‹Θ ; {||} ; $\Gamma[bv::=b]_{\Gamma b}$ $\vdash$ $v[bv::=b]_{vb}$ $\Leftarrow$ {| *z* : *B-bool* | *TRUE* |}› **using** *check-ifI subst-b-check-v* **by** *metis*
show ‹ Θ ; Φ ; {||} ; $\Gamma[bv::=b]_{\Gamma b}$; $\Delta[bv::=b]_{\Delta b}$ $\vdash$ *s1*$[bv::=b]_{sb}$ $\Leftarrow$ {| *z* : *b-of* $\tau[bv::=b]_{\tau b}$ | *CE-val* ($v[bv::=b]_{vb}$) == *CE-val* (*V-lit L-true*) *IMP c-of* $\tau[bv::=b]_{\tau b}$ *z* |}›
using *subst-b-if check-ifI* **by** *metis*
show ‹ Θ ; Φ ; {||} ; $\Gamma[bv::=b]_{\Gamma b}$; $\Delta[bv::=b]_{\Delta b}$ $\vdash$ *s2*$[bv::=b]_{sb}$ $\Leftarrow$ {| *z* : *b-of* $\tau[bv::=b]_{\tau b}$ | *CE-val* ($v[bv::=b]_{vb}$) == *CE-val* (*V-lit L-false*) *IMP c-of* $\tau[bv::=b]_{\tau b}$ *z* |}›
using *subst-b-if check-ifI* **by** *metis*
qed

next

case (*check-let2I x* Θ Φ $\mathcal{B}$ *G* Δ *t s1* τ *s2*)
show *?case* **unfolding** *subst-b-simps* **proof** (*rule Typing.check-let2I*)
have *atom x* $\sharp$ *b* **using** *x-fresh-b* **by** *auto*
show ‹*atom x* $\sharp$ (Θ, Φ, {||}, *G*$[bv::=b]_{\Gamma b}$, $\Delta[bv::=b]_{\Delta b}$, *t*$[bv::=b]_{\tau b}$, *s1*$[bv::=b]_{sb}$, $\tau[bv::=b]_{\tau b}$)›
apply(*unfold fresh-prodN*, *auto*, *auto simp add*: *check-let2I fresh-prodN fresh-empty-fset*)
apply(*metis subst-b-fresh-x check-let2I fresh-prodN*)+
done

show ‹ Θ ; Φ ; {||} ; *G*$[bv::=b]_{\Gamma b}$; $\Delta[bv::=b]_{\Delta b}$ $\vdash$ *s1*$[bv::=b]_{sb}$ $\Leftarrow$ *t*$[bv::=b]_{\tau b}$ › **using** *check-let2I subst-tb.simps* **by** *auto*
show ‹ Θ ; Φ ; {||} ; (*x*, *b-of t*$[bv::=b]_{\tau b}$, *c-of t*$[bv::=b]_{\tau b}$ *x*) $\#_\Gamma$ *G*$[bv::=b]_{\Gamma b}$; $\Delta[bv::=b]_{\Delta b}$ $\vdash$ *s2*$[bv::=b]_{sb}$ $\Leftarrow$ $\tau[bv::=b]_{\tau b}$›
using *check-let2I subst-tb.simps subst-gb.simps b-of.simps subst-b-c-of subst-b-b-of* **by** *auto*
qed
next
case (*check-varI u* Θ Φ $\mathcal{B}$ Γ Δ τ' *v* τ *s*)
show *?case* **unfolding** *subst-b-simps* **proof**(*rule Typing.check-varI*)
show *atom u* $\sharp$ (Θ, Φ, {||}, $\Gamma[bv::=b]_{\Gamma b}$, $\Delta[bv::=b]_{\Delta b}$, $\tau'[bv::=b]_{\tau b}$, $v[bv::=b]_{vb}$, $\tau[bv::=b]_{\tau b}$)
by(*unfold fresh-prodN*,*auto simp add*: *check-varI fresh-empty-fset subst-b-fresh-u*)
show Θ ; {||} ; $\Gamma[bv::=b]_{\Gamma b}$ $\vdash$ $v[bv::=b]_{vb}$ $\Leftarrow$ $\tau'[bv::=b]_{\tau b}$ **using** *check-varI subst-b-check-v* **by** *auto*
show Θ ; Φ ; {||} ; (*subst-gb* Γ *bv b*) ; (*u*, ($\tau'[bv::=b]_{\tau b}$)) $\#_\Delta$ (*subst-db* Δ *bv b*) $\vdash$ ($s[bv::=b]_{sb}$) $\Leftarrow$ ($\tau[bv::=b]_{\tau b}$) **using** *check-varI* **by** *auto*

qed
next
case (*check-assignI* Θ Φ $\mathcal{B}$ Γ Δ u τ v z τ')
show *?case* **unfolding** *subst-b-simps* **proof**(*rule Typing.check-assignI*)
show Θ $\vdash_{wf}$ Φ **using** *check-assignI* **by** *auto*
show Θ ; $\{||\}$; $\Gamma[bv::=b]_{\Gamma b}$ $\vdash_{wf}$ $\Delta[bv::=b]_{\Delta b}$ **using** *wf-b-subst check-assignI* **by** *auto*
show (u, $\tau[bv::=b]_{\tau b}$) $\in$ *setD* $\Delta[bv::=b]_{\Delta b}$ **using** *check-assignI subst-d-b-member* **by** *simp*
show Θ ; $\{||\}$; $\Gamma[bv::=b]_{\Gamma b}$ $\vdash$ $v[bv::=b]_{vb}$ $\Leftarrow$ $\tau[bv::=b]_{\tau b}$ **using** *check-assignI subst-b-check-v* **by** *auto*
show Θ ; $\{||\}$; $\Gamma[bv::=b]_{\Gamma b}$ $\vdash$ $\{\!|$ z : *B-unit* | *TRUE* $|\!\}$ $\lesssim$ $\tau'[bv::=b]_{\tau b}$ **using** *check-assignI subst-b-subtype subst-b-simps subst-tb.simps* **by** *fastforce*
qed
next
case (*check-whileI* Θ Φ $\mathcal{B}$ Γ Δ *s1* z *s2* τ')
show *?case* **unfolding** *subst-b-simps* **proof**(*rule Typing.check-whileI*)
show Θ ; Φ ; $\{||\}$; $\Gamma[bv::=b]_{\Gamma b}$; $\Delta[bv::=b]_{\Delta b}$ $\vdash$ *s1*$[bv::=b]_{sb}$ $\Leftarrow$ $\{\!|$ z : *B-bool* | *TRUE* $|\!\}$ **using** *check-whileI* **by** *auto*
show Θ ; Φ ; $\{||\}$; $\Gamma[bv::=b]_{\Gamma b}$; $\Delta[bv::=b]_{\Delta b}$ $\vdash$ *s2*$[bv::=b]_{sb}$ $\Leftarrow$ $\{\!|$ z : *B-unit* | *TRUE* $|\!\}$ **using** *check-whileI* **by** *auto*
show Θ ; $\{||\}$; $\Gamma[bv::=b]_{\Gamma b}$ $\vdash$ $\{\!|$ z : *B-unit* | *TRUE* $|\!\}$ $\lesssim$ $\tau'[bv::=b]_{\tau b}$ **using** *subst-b-subtype check-whileI* **by** *fastforce*
qed
next
case (*check-seqI* Θ Φ $\mathcal{B}$ Γ Δ *s1* z *s2* τ)
then show *?case* **unfolding** *subst-sb.simps* **using** *check-seqI Typing.check-seqI subst-b-eq* **by** *metis*
next
case (*check-caseI* Θ Φ $\mathcal{B}$ Γ Δ *tid dclist v cs* τ z)
show *?case* **unfolding** *subst-b-simps* **proof**(*rule Typing.check-caseI*)
show ‹ Θ ; Φ ; $\{||\}$; $\Gamma[bv::=b]_{\Gamma b}$; $\Delta[bv::=b]_{\Delta b}$; *tid* ; *dclist* ; $v[bv::=b]_{vb}$ $\vdash$ *subst-branchlb cs bv b* $\Leftarrow$ $\tau[bv::=b]_{\tau b}$› **using** *check-caseI* **by** *auto*
show ‹*AF-typedef tid dclist* $\in$ *set* Θ› **using** *check-caseI* **by** *auto*
show ‹Θ ; $\{||\}$; $\Gamma[bv::=b]_{\Gamma b}$ $\vdash$ $v[bv::=b]_{vb}$ $\Leftarrow$ $\{\!|$ z : *B-id tid* | *TRUE* $|\!\}$› **using** *check-caseI subst-b-check-v subst-b-simps subst-tb.simps subst-b-simps*
proof −
have $\{\!|$ z : *B-id tid* | *TRUE* $|\!\}$ = $\{\!|$ z : *B-id tid* | *TRUE* $|\!\}[bv::=b]_{\tau b}$ **using** *subst-b-eq* **by** *auto*
then show *?thesis*
by (*metis* (*no-types*) *check-caseI.hyps*(*4*) *check-caseI.prems*(*1*) *check-caseI.prems*(*2*) *subst-b-check-v*)
qed
show ‹ $\vdash_{wf}$ Θ › **using** *check-caseI* **by** *auto*
qed
qed

end

method *supp-calc* = (*metis* (*mono-tags*, *hide-lams*) *pure-supp* *c.supp e.supp v.supp supp-l-empty opp.supp sup-bot.right-neutral supp-at-base*)
declare *infer-e.intros*[*simp*]
declare *infer-e.intros*[*intro*]

Chapter 16

Safety

16.1 Operational Semantics

lemma *dclist-distinct-unique*:
assumes (*dc*, *const*) ∈ *set dclist2* **and** (*cons*, *const1*) ∈ *set dclist2* **and** *dc*=*cons* **and** *distinct* (*List.map fst dclist2*)
shows (*const*) = *const1*
proof −
have (*cons*, *const*) = (*dc*, *const1*)
using *assms* **by** (*metis* (*no-types*, *lifting*) *assms*(*3*) *assms*(*4*) *distinct.simps*(*1*) *distinct.simps*(*2*) *empty-iff insert-iff list.set*(*1*) *list.simps*(*15*) *list.simps*(*8*) *list.simps*(*9*) *map-of-eq-Some-iff*)
thus *?thesis* **by** *auto*
qed

lemma *fresh-d-fst-d*:
assumes *atom u* ♯ *δ*
shows *u* ∉ *fst* ‘ *set δ*
using *assms* **proof**(*induct δ*)
case *Nil*
then show *?case* **by** *auto*
next
case (*Cons ut δ′*)
obtain *u′* **and** *t′* **where** ∗:*ut* = (*u′*,*t′*) **by** *fastforce*
hence *atom u* ♯ *ut* ∧ *atom u* ♯ *δ′* **using** *fresh-Cons Cons* **by** *auto*
moreover hence *atom u* ♯ *fst ut* **using** ∗ *fresh-Pair*[*of atom u u′ t′*] *Cons* **by** *auto*
ultimately show *?case* **using** *Cons* **by** *auto*
qed

nominal-function *dc-of* :: *branch-s* ⇒ *string* **where**
dc-of (*AS-branch dc - -*) = *dc*
apply(*auto*,*simp add*: *eqvt-def dc-of-graph-aux-def*)
using *s-branch-s-branch-list.exhaust* **by** *metis*
nominal-termination (*eqvt*) **by** *lexicographic-order*

lemma *delta-sim-fresh*:
assumes Θ ⊢ *δ* ∼ Δ **and** *atom u* ♯ *δ*

shows *atom u* ♯ Δ
using *assms* **proof**(*induct rule* : *delta-sim.inducts*)
case (*delta-sim-nilI* Θ)
then show *?case* **using** *fresh-def supp-DNil* **by** *blast*
next
case (*delta-sim-consI* Θ δ Δ *v* τ *u'*)
hence Θ ; {||} ; *GNil* ⊢$_{wf}$ τ **using** *check-v-wf* **by** *meson*
hence *supp* τ = {} **using** *wfT-supp* **by** *fastforce*
moreover have *atom u* ♯ *u'* **using** *delta-sim-consI fresh-Cons fresh-Pair* **by** *blast*
moreover have *atom u* ♯ Δ **using** *delta-sim-consI fresh-Cons* **by** *blast*
ultimately show *?case* **using** *fresh-Pair fresh-DCons fresh-def* **by** *blast*
qed

lemma *delta-sim-v*:
fixes Δ::Δ
assumes Θ ⊢ δ ∼ Δ **and** (*u*,*v*) ∈ *set* δ **and** (*u*,τ) ∈ *setD* Δ **and** Θ ; {||} ; *GNil* ⊢$_{wf}$ Δ
shows Θ ; {||} ; *GNil* ⊢ *v* ⇐ τ
using *assms* **proof**(*induct* δ *arbitrary*: Δ)
case *Nil*
then show *?case* **by** *auto*
next
case (*Cons uv* δ)
obtain *u'* **and** *v'* **where** *uv* : *uv*=(*u'*,*v'*) **by** *fastforce*
show *?case* **proof**(*cases u'*=*u*)
case *True*
hence ∗:Θ ⊢ ((*u*,*v'*)#δ) ∼ Δ **using** *uv Cons* **by** *blast*
then obtain τ' **and** Δ' **where** *tt*: Θ ; {||} ; *GNil* ⊢ *v'* ⇐ τ' ∧ *u* ∉ *fst* ' *set* δ ∧ Δ = (*u*,τ')#$_{\Delta}$Δ'
using *delta-sim-elims*(*3*)[*OF* ∗] **by** *metis*
moreover hence *v'*=*v* **using** *Cons True*
by (*metis Pair-inject fst-conv image-eqI set-ConsD uv*)
moreover have τ=τ' **using** *wfD-unique tt Cons*
setD.simps list.set-intros **by** *blast*
ultimately show *?thesis* **by** *metis*
next
case *False*
hence ∗:Θ ⊢ ((*u'*,*v'*)#δ) ∼ Δ **using** *uv Cons* **by** *blast*
then obtain τ' **and** Δ' **where** *tt*: Θ ⊢ δ ∼ Δ' ∧ Θ ; {||} ; *GNil* ⊢ *v'* ⇐ τ' ∧ *u'* ∉ *fst* ' *set* δ ∧ Δ
= (*u'*,τ')#$_{\Delta}$Δ' **using** *delta-sim-elims*(*3*)[*OF* ∗] **by** *metis*

moreover hence Θ ; {||} ; *GNil* ⊢$_{wf}$ Δ' **using** *wfD-elims Cons delta-sim-elims* **by** *metis*
ultimately show *?thesis* **using** *Cons*
using *False* **by** *auto*
qed
qed

lemma *delta-sim-delta-lookup*:
assumes Θ ⊢ δ ∼ Δ **and** (*u*, {| *z* : *b* | *c* |}) ∈ *setD* Δ
shows ∃ *v*. (*u*,*v*) ∈ *set* δ
using *assms* **by**(*induct rule*: *delta-sim.inducts*,*auto*+)

lemma *update-d-stable*:

fst ‘ set δ = fst ‘ set (update-d δ u v)
proof(*induct δ*)
case *Nil*
then show *?case* **by** *auto*
next
case (*Cons a δ*)
then show *?case* **using** *update-d.simps*
by (*metis* (*no-types*, *lifting*) *eq-fst-iff image-cong image-insert list.simps*(*15*) *prod.exhaust-sel*)
qed

lemma *update-d-sim*:
fixes Δ::Δ
assumes Θ ⊢ δ ∼ Δ **and** Θ ; {||} ; *GNil* ⊢ *v* ⇐ τ **and** (*u*,τ) ∈ *setD* Δ **and** Θ ; {||} ; *GNil* $\vdash_{wf}$ Δ
shows Θ ⊢ (*update-d δ u v*) ∼ Δ
using *assms* **proof**(*induct δ arbitrary*: Δ)
case *Nil*
then show *?case* **using** *delta-sim-consI* **by** *simp*
next
case (*Cons uv δ*)
obtain *u′* **and** *v′* **where** *uv* : *uv*=(*u′*,*v′*) **by** *fastforce*

hence ∗:Θ ⊢ ((*u′*,*v′*)#δ) ∼ Δ **using** *uv Cons* **by** *blast*
then obtain τ′ **and** Δ′ **where** *tt*: Θ ⊢ δ ∼ Δ′ ∧ Θ ; {||} ; *GNil* ⊢ *v′* ⇐ τ′ ∧ *u′* ∉ *fst ‘ set δ* ∧ Δ = (*u′*,τ′)#$_{\Delta}$Δ′ **using** *delta-sim-elims* ∗ **by** *metis*

show *?case* **proof**(*cases u*=*u′*)
case *True*
then have (*u*,τ′) ∈ *setD* Δ **using** *tt* **by** *auto*
then have τ = τ′ **using** *Cons wfD-unique* **by** *metis*
moreover have *update-d* ((*u′*,*v′*)#δ) *u v* = ((*u′*,*v*)#δ) **using** *update-d.simps True* **by** *presburger*
ultimately show *?thesis* **using** *delta-sim-consI tt Cons True*
by (*simp add*: *tt uv*)
next
case *False*
have Θ ⊢ (*u′*,*v′*) # (*update-d δ u v*) ∼ (*u′*,τ′)#$_{\Delta}$Δ′
proof(*rule delta-sim-consI*)
show Θ ⊢ *update-d δ u v* ∼ Δ′ **using** *Cons* **using** *delta-sim-consI*
delta-sim.simps update-d.simps Cons delta-sim-elims uv tt
False fst-conv set-ConsD wfG-elims wfD-elims **by** (*metis setD-ConsD*)
show Θ ; {||} ; *GNil* ⊢ *v′* ⇐ τ′ **using** *tt* **by** *auto*
show *u′* ∉ *fst ‘ set* (*update-d δ u v*) **using** *update-d.simps Cons update-d-stable tt* **by** *auto*
qed
thus *?thesis* **using** *False update-d.simps uv*
by (*simp add*: *tt*)
qed
qed

16.2 Preservation

Types are preserved under reduction step

lemma *check-if*:

fixes $s'::s$ **and** $cs::branch\text{-}s$ **and** $css::branch\text{-}list$ **and** $v::v$
shows $\Theta\ ;\ \Phi\ ;\ B\ \ ;\ G\ ;\ \Delta\ \vdash\ s' \Leftarrow \tau \Longrightarrow s' =\ IF\ (V\text{-}lit\ ll)\ THEN\ s1\ ELSE\ s2 \Longrightarrow$
$\Theta\ ;\ \{||\}\ ;\ GNil \vdash_{wf} \tau \Longrightarrow G = GNil \Longrightarrow B = \{||\} \Longrightarrow ll = L\text{-}true \wedge s = s1 \vee ll = L\text{-}false \wedge s = s2 \Longrightarrow$
$\Theta\ ;\ \Phi\ ;\ \{||\}\ ;\ GNil\ ;\ \Delta\ \vdash\ s \Leftarrow \tau$ **and**
$check\text{-}branch\text{-}s\ \Theta\ \Phi\ \ \{||\}\ GNil\ \Delta\ tid\ dc\ const\ v\ cs\ \tau \Longrightarrow True$ **and**
$check\text{-}branch\text{-}list\ \Theta\ \Phi\ \ \{||\}\ \Gamma\ \Delta\ tid\ dclist\ v\ css\ \tau \Longrightarrow True$
proof($nominal\text{-}induct\ \tau$ **and** τ **and** τ *rule*: $check\text{-}s\text{-}check\text{-}branch\text{-}s\text{-}check\text{-}branch\text{-}list.strong\text{-}induct$)
case ($check\text{-}ifI\ z\ \Theta\ \Phi\ \mathcal{B}\ \Gamma\ \Delta\ v\ s1\ s2\ \tau$)
obtain z' **where** teq: $\tau = \{\!|\ z' : b\text{-}of\ \tau\ |\ c\text{-}of\ \tau\ z'\ |\!\} \wedge atom\ z' \sharp (z,\tau)$ **using** $obtain\text{-}fresh\text{-}z\text{-}c\text{-}of$ **by** *metis*
hence ceq: $(c\text{-}of\ \tau\ z')[z'::=[\ z\ ]^v]_{cv} = (c\text{-}of\ \tau\ z)$ **using** $c\text{-}of\text{-}switch\ fresh\text{-}Pair$ **by** *metis*
have zf: $atom\ z \sharp c\text{-}of\ \tau\ z'$ **by**(*rule* $c\text{-}of\text{-}fresh$, *auto simp add*: *freshers* $check\text{-}ifI$, *insert* $fresh\text{-}Pair\ teq\ fresh\text{-}at\text{-}base$, *simp add*: *freshers*)

hence *1*:$\Theta\ ;\ \Phi\ ;\ \{||\}\ ;\ GNil\ ;\ \Delta\ \vdash s \Leftarrow \{\!|\ z : b\text{-}of\ \tau\ |\ CE\text{-}val\ (V\text{-}lit\ ll) == CE\text{-}val\ (V\text{-}lit\ ll)\ IMP\ c\text{-}of\ \tau\ z\ |\!\}$ **using** $check\text{-}ifI$ **by** *auto*
moreover have *2*:$\Theta\ ;\ \{||\}\ ;\ GNil \vdash (\{\!|\ z : b\text{-}of\ \tau\ |\ CE\text{-}val\ (V\text{-}lit\ ll) == CE\text{-}val\ (V\text{-}lit\ ll)\ IMP\ c\text{-}of\ \tau\ z\ |\!\}) \lesssim \tau$
proof −
have $\Theta\ ;\ \{||\}\ ;\ GNil \vdash_{wf} (\{\!|\ z : b\text{-}of\ \tau\ |\ CE\text{-}val\ (V\text{-}lit\ ll\ ) == CE\text{-}val\ (V\text{-}lit\ ll)\ \ IMP\ c\text{-}of\ \tau\ z\ |\!\})$ **using** $check\text{-}ifI\ check\text{-}s\text{-}wf$ **by** *auto*
moreover have $\Theta\ ;\ \{||\}\ ;\ GNil\ \vdash_{wf} \tau$ **using** $check\text{-}s\text{-}wf\ check\text{-}ifI$ **by** *auto*
ultimately show *?thesis* **using** $subtype\text{-}if\text{-}simp[of\ \Theta\ \ \{||\}\ z\ b\text{-}of\ \tau\ ll\ c\text{-}of\ \tau\ z'\ z']$ **using** $teq\ ceq\ zf\ subst\text{-}defs$ **by** *metis*
qed
ultimately show *?case* **using** $check\text{-}s\text{-}supertype(1)\ check\text{-}ifI$ **by** *metis*

qed(*auto*+)

lemma $preservation\text{-}if$:
assumes $\Theta\ ;\ \Phi\ ;\ \Delta \vdash \langle\ \delta\ ,\ IF\ (V\text{-}lit\ ll)\ THEN\ s1\ ELSE\ s2\ \rangle \Leftarrow \tau$ **and**
$ll = L\text{-}true \wedge s = s1 \vee ll = L\text{-}false \wedge s = s2$
shows $\Theta\ ;\ \Phi\ ;\ \Delta\ \vdash \langle\ \delta\ ,\ s\ \rangle \Leftarrow \tau\ \wedge\ setD\ \Delta \subseteq setD\ \Delta$
proof −
have $*$: $\Theta\ ;\ \Phi\ ;\ \{||\}\ ;\ GNil\ ;\ \Delta\ \vdash\ AS\text{-}if\ (V\text{-}lit\ ll)\ s1\ s2 \Leftarrow \tau \wedge (\forall fd{\in}set\ \Phi.\ check\text{-}fundef\ \Theta\ \Phi\ fd)$
using $assms\ config\text{-}type\text{-}elims$ **by** *metis*
hence $\Theta\ ;\ \Phi\ ;\ \{||\}\ ;\ GNil\ ;\ \Delta\ \vdash\ s \Leftarrow \tau$ **using** $check\text{-}s\text{-}wf\ check\text{-}if\ assms$ **by** *metis*
hence $\Theta\ ;\ \Phi\ ;\ \Delta \vdash \langle\ \delta\ ,\ s\ \rangle \Leftarrow \tau\ \wedge\ setD\ \Delta \subseteq setD\ \Delta$ **using** $config\text{-}typeI\ *$
using $assms(1)$ **by** *blast*
thus *?thesis* **by** *blast*
qed

lemma $check\text{-}s\text{-}x\text{-}fresh$:
fixes $x::x$ **and** $s::s$
assumes $\Theta\ ;\ \Phi\ ;\ B\ ;\ \ GNil\ ;\ D\ \vdash s \Leftarrow \tau$
shows $atom\ x \sharp s \wedge atom\ x \sharp \tau \wedge atom\ x \sharp D$
proof −
have $\Theta\ ;\ \Phi\ ;\ B\ ;\ GNil\ ;\ D \vdash_{wf} s : b\text{-}of\ \tau$ **using** $check\text{-}s\text{-}wf[OF\ assms]$ **by** *auto*
moreover have $\Theta\ ;\ B\ ;\ GNil\ \ \vdash_{wf} \tau$ **using** $check\text{-}s\text{-}wf\ assms$ **by** *auto*
moreover have $\Theta\ ;\ B\ ;\ GNil\ \ \vdash_{wf} D$ **using** $check\text{-}s\text{-}wf\ assms$ **by** *auto*
ultimately show *?thesis* **using** $wf\text{-}supp\ x\text{-}fresh\text{-}u$

by (*meson fresh-GNil wfS-x-fresh wfT-x-fresh wfD-x-fresh*)
qed

lemma *check-funtyp-subst-b*:
fixes $b'::b$
assumes *check-funtyp* $\Theta\ \Phi\ \{|bv|\}\ (AF\text{-}fun\text{-}typ\ x\ b\ c\ \tau\ s)$ **and** $\langle\ \Theta\ ;\ \{||\}\ \vdash_{wf}\ b'\ \rangle$
shows *check-funtyp* $\Theta\ \Phi\ \ \{||\}\ (AF\text{-}fun\text{-}typ\ x\ b[bv::=b']_{bb}\ (c[bv::=b']_{cb})\ \tau[bv::=b']_{\tau b}\ s[bv::=b']_{sb})$
using *assms* **proof** (*nominal-induct* $\{|bv|\}$ *AF-fun-typ x b c* τ *s rule*: *check-funtyp.strong-induct*)
case (*check-funtypI* $x'\ \Theta\ \Phi\ c'\ s'\ \tau'$)
have *check-funtyp* $\Theta\ \Phi\ \{||\}\ (AF\text{-}fun\text{-}typ\ x'\ b[bv::=b']_{bb}\ (c'[bv::=b']_{cb})\ \tau'[bv::=b']_{\tau b}\ s'[bv::=b']_{sb})$ **proof**
show $\langle atom\ x'\ \sharp\ (\Theta, \Phi, \{||\}::bv\ fset, b[bv::=b']_{bb})\rangle$ **using** *check-funtypI fresh-prodN x-fresh-b fresh-empty-fset*
by *metis*

have $\langle\ \Theta\ ;\ \Phi\ ;\ \{||\}\ ;\ ((x',\ b,\ c')\ \#_\Gamma\ GNil)[bv::=b']_{\Gamma b}\ ;\ []_\Delta[bv::=b']_{\Delta b}\ \vdash\ s'[bv::=b']_{sb} \Leftarrow \tau'[bv::=b']_{\tau b}\rangle$
proof(*rule subst-b-check-s*)
show $\langle\ \Theta\ ;\ \{||\}\ \vdash_{wf}\ b'\ \rangle$ **using** *check-funtypI* **by** *metis*
show $\langle\{|bv|\} = \{|bv|\}\rangle$ **by** *auto*
show $\langle\ \Theta\ ;\ \Phi\ ;\ \{|bv|\}\ ;\ (x',\ b,\ c')\ \#_\Gamma\ GNil\ ;\ []_\Delta\ \vdash\ s' \Leftarrow \tau'\rangle$ **using** *check-funtypI* **by** *metis*
qed

thus $\langle\ \Theta\ ;\ \Phi\ ;\ \{||\}\ ;\ (x',\ b[bv::=b']_{bb},\ c'[bv::=b']_{cb})\ \#_\Gamma\ GNil\ ;\ []_\Delta\ \vdash\ s'[bv::=b']_{sb} \Leftarrow \tau'[bv::=b']_{\tau b}\rangle$
using *subst-gb.simps subst-db.simps* **by** *simp*
qed

moreover have $(AF\text{-}fun\text{-}typ\ x\ b\ c\ \tau\ s) = (AF\text{-}fun\text{-}typ\ x'\ b\ c'\ \tau'\ s')$ **using** *fun-typ.eq-iff check-funtypI*
by *metis*
moreover hence $(AF\text{-}fun\text{-}typ\ x\ b[bv::=b']_{bb}\ (c[bv::=b']_{cb})\ \tau[bv::=b']_{\tau b}\ s[bv::=b']_{sb}) = (AF\text{-}fun\text{-}typ\ x'\ b[bv::=b']_{bb}\ (c'[bv::=b']_{cb})\ \tau'[bv::=b']_{\tau b}\ s'[bv::=b']_{sb})$
using *subst-ft-b.simps* **by** *metis*
ultimately show *?case* **by** *metis*
qed

lemma *funtyp-simple-check*:
fixes $s::s$ **and** $\Delta::\Delta$ **and** $\tau::\tau$ **and** $v::v$
assumes *check-funtyp* $\Theta\ \Phi\ \ (\{||\}::bv\ fset)\ (AF\text{-}fun\text{-}typ\ x\ b\ c\ \tau\ s)$ **and**
$\Theta\ ;\ \{||\}\ ;\ GNil \vdash v \Leftarrow \{\!|\ x : b\ |\ c\ |\!\}$
shows $\Theta\ ;\ \Phi\ ;\ \ \{||\}\ ;\ GNil\ ;\ DNil \vdash s[x::=v]_{sv} \Leftarrow \tau[x::=v]_{\tau v}$
using *assms* **proof**(*nominal-induct* $(\{||\}::bv\ fset)$ *AF-fun-typ x b c* τ *s avoiding*: *v x rule*: *check-funtyp.strong-induct*)
case (*check-funtypI* $x'\ \Theta\ \Phi\ c'\ s'\ \tau'$)

hence *eq1*: $\{\!|\ x' : b\ \ |\ c'\ |\!\} = \{\!|\ x : b\ \ |\ c\ |\!\}$ **using** *funtyp-eq-iff-equalities* **by** *metis*

obtain x'' **and** c'' **where** *xf*:$\{\!|\ x : b\ \ |\ c\ |\!\} = \{\!|\ x'' : b\ |\ c''\ |\!\} \wedge atom\ x''\ \sharp\ (x',v) \wedge atom\ x''\ \sharp\ (x,c)$
using *obtain-fresh-z3* **by** *metis*
moreover have *atom* $x'\ \sharp\ c''$ **proof** −
have *supp* $\{\!|\ x'' : b\ |\ c''\ |\!\} = \{\}$ **using** *eq1 check-funtypI xf check-v-wf wfT-nil-supp* **by** *metis*
hence *supp* $c'' \subseteq \{\ atom\ x''\ \}$ **using** τ*.supp eq1 xf* **by** (*auto simp add*: *freshers*)
moreover have *atom* $x' \neq atom\ x''$ **using** *xf fresh-Pair fresh-x-neq* **by** *metis*
ultimately show *?thesis* **using** *xf fresh-Pair fresh-x-neq fresh-def fresh-at-base* **by** *blast*
qed
ultimately have *eq2*: $c''[x''::=[\ x'\ ]^v]_{cv} = c'$ **using** *eq1 type-eq-subst-eq3*(*1*)[*of x' b c' x'' b c'*] **by**
metis

have *atom* $x' \sharp c$ **proof** −
have *supp* $\{\!|\ x : b \mid c\ |\!\} = \{\}$ **using** *eq1 check-funtypI xf check-v-wf wfT-nil-supp* **by** *metis*
hence *supp* $c \subseteq \{\ atom\ x\ \}$ **using** τ*.supp* **by** *auto*
moreover have *atom* $x \neq atom\ x'$ **using** *check-funtypI fresh-Pair fresh-x-neq* **by** *metis*
ultimately show *?thesis* **using** *fresh-def* **by** *force*
qed
hence *eq*: $c[x::=[\ x'\ ]^v]_{cv} = c' \wedge s'[x'::=v]_{sv} = s[x::=v]_{sv} \wedge \tau'[x'::=v]_{\tau v} = \tau[x::=v]_{\tau v}$
using *funtyp-eq-iff-equalities type-eq-subst-eq3 check-funtypI* **by** *metis*

have $\Theta\ ;\ \Phi\ ;\ \{||\}\ ;\ ((x',\ b,\ c'[x''::=[\ x'\ ]^v]_{cv})\ \#_\Gamma\ GNil)[x'::=v]_{\Gamma v}\ ;\ []_\Delta[x'::=v]_{\Delta v}\ \vdash s'[x'::=v]_{sv} \Leftarrow \tau'[x'::=v]_{\tau v}$
proof(*rule subst-check-check-s*)
show $\langle \Theta\ ;\ \{||\}\ ;\ GNil\ \vdash v \Leftarrow \{\!|\ x'' : b\ \mid c''\ |\!\}\rangle$ **using** *check-funtypI eq1 xf* **by** *metis*
show $\langle atom\ x'' \sharp (x',\ v)\rangle$ **using** *check-funtypI fresh-x-neq fresh-Pair xf* **by** *metis*
show $\langle\ \Theta\ ;\ \Phi\ ;\ \{||\}\ ;\ (x',\ b,\ c''[x''::=[\ x'\ ]^v]_{cv})\ \#_\Gamma\ GNil\ ;\ []_\Delta\ \vdash s' \Leftarrow \tau'\rangle$ **using** *check-funtypI eq2* **by** *metis*
show $\langle\ (x',\ b,\ c''[x''::=[\ x'\ ]^v]_{cv})\ \#_\Gamma\ GNil = GNil\ @\ (x',\ b,\ c''[x''::=[\ x'\ ]^v]_{cv})\ \#_\Gamma\ GNil\rangle$ **using** *append-g.simps* **by** *auto*
qed
hence $\Theta\ ;\ \Phi\ ;\ \{||\}\ ;\ GNil\ ;\ []_\Delta\ \vdash s'[x'::=v]_{sv} \Leftarrow \tau'[x'::=v]_{\tau v}$ **using** *subst-gv.simps subst-dv.simps* **by** *auto*
thus *?case* **using** *eq* **by** *auto*
qed

lemma *funtypq-simple-check*:
fixes $s::s$ **and** $\Delta::\Delta$ **and** $\tau::\tau$ **and** $v::v$
assumes *check-funtypq* $\Theta\ \Phi$ (*AF-fun-typ-none* (*AF-fun-typ x b c t s*)) **and**
$\Theta\ ;\ \{||\}\ ;\ GNil \vdash v \Leftarrow \{\!|\ x : b \mid c\ |\!\}$
shows $\Theta\ ;\ \Phi\ ;\ \{||\}\ ;\ GNil\ ;\ DNil \vdash s[x::=v]_{sv} \Leftarrow t[x::=v]_{\tau v}$
using *assms* **proof**(*nominal-induct* (*AF-fun-typ-none* (*AF-fun-typ x b c t s*)) *avoiding*: *v rule*: *check-funtypq.strong-induc*
case (*check-fundefq-simpleI* $\Theta\ \Phi\ x'\ c'\ t'\ s'$)
hence *eq*: $\{\!|\ x : b\ \mid c\ |\!\} = \{\!|\ x' : b\ \mid c'\ |\!\} \wedge s'[x'::=v]_{sv} = s[x::=v]_{sv} \wedge t[x::=v]_{\tau v} = t'[x'::=v]_{\tau v}$
using *funtyp-eq-iff-equalities* **by** *metis*
hence $\Theta\ ;\ \Phi\ ;\ \{||\}\ ;\ GNil\ ;\ []_\Delta\ \vdash s'[x'::=v]_{sv} \Leftarrow t'[x'::=v]_{\tau v}$
using *funtyp-simple-check*[*OF check-fundefq-simpleI*(*1*)] *check-fundefq-simpleI* **by** *metis*
thus *?case* **using** *eq* **by** *metis*
qed

lemma *funtyp-poly-eq-iff-equalities*:
assumes $[[atom\ bv']]lst.\ AF\text{-}fun\text{-}typ\ x'\ b''\ c'\ t'\ s' = [[atom\ bv]]lst.\ AF\text{-}fun\text{-}typ\ x\ b\ c\ t\ s$
shows $\{\!|\ x' : b''[bv'::=b']_{bb}\ \mid c'[bv'::=b']_{cb}\ |\!\} = \{\!|\ x : b[bv::=b']_{bb}\ \mid c[bv::=b']_{cb}\ |\!\} \wedge$
$s'[bv'::=b']_{sb}[x'::=v]_{sv} = s[bv::=b']_{sb}[x::=v]_{sv} \wedge t'[bv'::=b']_{\tau b}[x'::=v]_{\tau v} = t[bv::=b']_{\tau b}[x::=v]_{\tau v}$

proof −
have *subst-ft-b* (*AF-fun-typ x' b'' c' t' s'*) *bv' b'* = *subst-ft-b* (*AF-fun-typ x b c t s*) *bv b'*
using *subst-b-flip-eq-two subst-b-fun-typ-def assms* **by** *metis*
thus *?thesis* **using** *fun-typ.eq-iff subst-ft-b.simps funtyp-eq-iff-equalities subst-tb.simps*
by (*metis* (*full-types*) *assms fun-poly-arg-unique*)

qed

lemma *funtypq-poly-check*:
fixes $s::s$ **and** $\Delta::\Delta$ **and** $\tau::\tau$ **and** $v::v$ **and** $b'::b$
assumes *check-funtypq* $\Theta\ \Phi$ (*AF-fun-typ-some bv* (*AF-fun-typ x b c t s*)) **and**
$\Theta\ ;\ \{||\}\ ;\ GNil \vdash v \Leftarrow \{\!|\ x : b[bv::=b']_{bb}\ |\ c[bv::=b']_{cb}\ |\!\}$ **and**
$\Theta\ ;\ \{||\} \vdash_{wf} b'$
shows $\Theta\ ;\ \Phi\ ;\ \{||\}\ ;\ GNil\ ;\ DNil \vdash s[bv::=b']_{sb}[x::=v]_{sv} \Leftarrow t[bv::=b']_{\tau b}[x::=v]_{\tau v}$
using *assms* **proof**(*nominal-induct* (*AF-fun-typ-some bv* (*AF-fun-typ x b c t s*)) *avoiding*: *v rule*: *check-funtypq.strong-induct*)
case (*check-funtypq-polyI bv' Θ Φ x' b'' c' t' s'*)

hence $**$:$\{\!|\ x' : b''[bv'::=b']_{bb}\ |\ c'[bv'::=b']_{cb}\ |\!\} = \{\!|\ x : b[bv::=b']_{bb}\ |\ c[bv::=b']_{cb}\ |\!\} \wedge$
$s'[bv'::=b']_{sb}[x'::=v]_{sv} = s[bv::=b']_{sb}[x::=v]_{sv} \wedge t'[bv'::=b']_{\tau b}[x'::=v]_{\tau v} = t[bv::=b']_{\tau b}[x::=v]_{\tau v}$

using *funtyp-poly-eq-iff-equalities* **by** *metis*

have $*$:*check-funtyp* $\Theta\ \Phi\ \{||\}$ (*AF-fun-typ* $x'\ b''[bv'::=b']_{bb}$ $(c'[bv'::=b']_{cb})$ $(t'[bv'::=b']_{\tau b})$ $s'[bv'::=b']_{sb}$)
using *check-funtyp-subst-b*[*OF check-funtypq-polyI*(*5*) *check-funtypq-polyI*(*8*)] **by** *metis*
moreover have $\Theta\ ;\ \{||\}\ ;\ GNil \vdash v \Leftarrow \{\!|\ x' : b''[bv'::=b']_{bb}\ |\ c'[bv'::=b']_{cb}\ |\!\}$ **using** $**$ *check-funtypq-polyI* **by** *metis*
ultimately have $\Theta\ ;\ \Phi\ ;\ \{||\}\ ;\ GNil\ ;\ []_\Delta \vdash s'[bv'::=b']_{sb}[x'::=v]_{sv} \Leftarrow t'[bv'::=b']_{\tau b}[x'::=v]_{\tau v}$
using *funtyp-simple-check*[*OF* $*$] *check-funtypq-polyI* **by** *metis*
thus *?case* **using** $**$ **by** *metis*

qed

lemma *fundef-simple-check*:
fixes $s::s$ **and** $\Delta::\Delta$ **and** $\tau::\tau$ **and** $v::v$
assumes *check-fundef* $\Theta\ \Phi$ (*AF-fundef f* (*AF-fun-typ-none* (*AF-fun-typ x b c t s*))) **and**
$\Theta\ ;\ \{||\}\ ;\ GNil \vdash v \Leftarrow \{\!|\ x : b\ |\ c\ |\!\}$ **and** $\Theta\ ;\ \{||\}\ ;\ GNil \vdash_{wf} \Delta$
shows $\Theta\ ;\ \Phi\ ;\ \{||\}\ ;\ GNil\ ;\ \Delta \vdash s[x::=v]_{sv} \Leftarrow t[x::=v]_{\tau v}$
using *assms* **proof**(*nominal-induct* (*AF-fundef f* (*AF-fun-typ-none* (*AF-fun-typ x b c t s*))) *avoiding*: *v rule*: *check-fundef.strong-induct*)
case (*check-fundefI* $\Theta\ \Phi$)
then show *?case* **using** *funtypq-simple-check*[*THEN check-s-d-weakening*(*1*)] *setD.simps* **by** *auto*
qed

lemma *fundef-poly-check*:
fixes $s::s$ **and** $\Delta::\Delta$ **and** $\tau::\tau$ **and** $v::v$ **and** $b'::b$
assumes *check-fundef* $\Theta\ \Phi$ (*AF-fundef f* (*AF-fun-typ-some bv* (*AF-fun-typ x b c t s*))) **and**
$\Theta\ ;\ \{||\}\ ;\ GNil \vdash v \Leftarrow \{\!|\ x : b[bv::=b']_{bb}\ |\ c[bv::=b']_{cb}\ |\!\}$ **and** $\Theta\ ;\ \{||\}\ ;\ GNil \vdash_{wf} \Delta$ **and** $\Theta\ ;\ \{||\} \vdash_{wf} b'$
shows $\Theta\ ;\ \Phi\ ;\ \{||\}\ ;\ GNil\ ;\ \Delta \vdash s[bv::=b']_{sb}[x::=v]_{sv} \Leftarrow t[bv::=b']_{\tau b}[x::=v]_{\tau v}$
using *assms* **proof**(*nominal-induct* (*AF-fundef f* (*AF-fun-typ-some bv* (*AF-fun-typ x b c t s*))) *avoiding*: *v rule*: *check-fundef.strong-induct*)
case (*check-fundefI* $\Theta\ \Phi$)
then show *?case* **using** *funtypq-poly-check*[*THEN check-s-d-weakening*(*1*)] *setD.simps* **by** *auto*
qed

lemma *preservation-app*:

assumes
Some (*AF-fundef f* (*AF-fun-typ-none* (*AF-fun-typ x1 b1 c1 τ1′ s1′*))) = *lookup-fun* Φ *f* **and**
(∀ *fd*∈*set* Φ. *check-fundef* Θ Φ *fd*)
shows Θ ; Φ ; *B* ; *G* ; Δ ⊢ *ss* ⇐ τ ⟹ *B* = {||} ⟹ *G* = *GNil* ⟹ *ss* = *LET x* = (*AE-app f v*) *IN s* ⟹
Θ ; Φ ; {||} ; *GNil* ; Δ ⊢ *LET x* : (*τ1′*[*x1*::=*v*]$_{\tau v}$) = (*s1′*[*x1*::=*v*]$_{sv}$) *IN s* ⇐ τ **and**
check-branch-s Θ Φ *B GNil* Δ *tid dc const v cs* τ ⟹ *True* **and**
check-branch-list Θ Φ *B* Γ Δ *tid dclist v css* τ ⟹ *True*
using *assms* **proof**(*nominal-induct* τ **and** τ **and** τ *avoiding*: *v rule*: *check-s-check-branch-s-check-branch-list.strong-indu*
case (*check-letI x2* Θ Φ *B* Γ Δ *e* τ *z s2 b c*)

hence *eq*: *e* = (*AE-app f v*) **by** *simp*
hence ∗:Θ ; Φ ; {||} ;*GNil* ; Δ ⊢ (*AE-app f v*) ⇒ {| *z* : *b* | *c* |} **using** *check-letI* **by** *auto*

then obtain *x3 b3 c3 τ3 s3* **where**
∗∗:Θ ; {||} ; *GNil* ⊢$_{wf}$ Δ ∧ Θ ⊢$_{wf}$ Φ ∧ *Some* (*AF-fundef f* (*AF-fun-typ-none* (*AF-fun-typ x3 b3 c3 τ3 s3*))) = *lookup-fun* Φ *f* ∧
Θ ; {||} ; *GNil* ⊢ *v* ⇐ {| *x3* : *b3* | *c3* |} ∧ *atom x3* ♯ *GNil* ∧ *τ3*[*x3*::=*v*]$_{\tau v}$ = {| *z* : *b* | *c* |}
using *infer-e-elims*(*6*)[*OF* ∗] *subst-defs* **by** *metis*

obtain *z3* **where** *z3*:{| *x3* : *b3* | *c3* |} = {| *z3* : *b3* | *c3*[*x3*::=*V-var z3*]$_{cv}$ |} ∧ *atom z3* ♯ (*x3*, *v*,*c3*,*x1*,*c1*) **using** *obtain-fresh-z3* **by** *metis*

have *seq*:[[*atom x3*]]*lst*. *s3* = [[*atom x1*]]*lst*. *s1′* **using** *fun-def-eq check-letI* ∗∗ *option.inject* **by** *metis*

let *?ft* = *AF-fun-typ x3 b3 c3 τ3 s3*
thm *check-fundef-elims*

have *sup*: *supp τ3* ⊆ { *atom x3*} ∧ *supp s3* ⊆ { *atom x3*} **using** *wfPhi-f-supp* ∗∗ **by** *force*

have Θ ; Φ ; {||} ; *GNil* ; Δ ⊢ *AS-let2 x2 τ3*[*x3*::=*v*]$_{\tau v}$ (*s3*[*x3*::=*v*]$_{sv}$) *s2* ⇐ τ **proof**
show ‹*atom x2* ♯ (Θ, Φ, {||}::*bv fset*, *GNil*, Δ, *τ3*[*x3*::=*v*]$_{\tau v}$, *s3*[*x3*::=*v*]$_{sv}$, τ)›
unfolding *fresh-prodN* **using** *check-letI fresh-subst-v-if subst-v-τ-def sup*
by (*metis all-not-in-conv fresh-def fresh-empty-fset fresh-subst-sv-if fresh-subst-tv-if singleton-iff subset-singleton-iff*)

show ‹ Θ ; Φ ; {||} ; *GNil* ; Δ ⊢ *s3*[*x3*::=*v*]$_{sv}$ ⇐ *τ3*[*x3*::=*v*]$_{\tau v}$› **proof**(*rule fundef-simple-check*)
show ‹*check-fundef* Θ Φ (*AF-fundef f* (*AF-fun-typ-none* (*AF-fun-typ x3 b3 c3 τ3 s3*)))› **using**
∗∗ *check-letI lookup-fun-member* **by** *metis*
show ‹Θ ; {||} ; *GNil* ⊢ *v* ⇐ {| *x3* : *b3* | *c3* |}› **using** ∗∗ **by** *auto*
show ‹ Θ ; {||} ; *GNil* ⊢$_{wf}$ Δ › **using** ∗∗ **by** *auto*
qed
show ‹ Θ ; Φ ; {||} ; (*x2*, *b-of τ3*[*x3*::=*v*]$_{\tau v}$, *c-of τ3*[*x3*::=*v*]$_{\tau v}$ *x2*) #$_\Gamma$ *GNil* ; Δ ⊢ *s2* ⇐ τ›
using *check-letI* ∗∗ *b-of.simps c-of.simps subst-defs* **by** *metis*
qed

moreover have *AS-let2 x2 τ3*[*x3*::=*v*]$_{\tau v}$ (*s3*[*x3*::=*v*]$_{sv}$) *s2* = *AS-let2 x* (*τ1′*[*x1*::=*v*]$_{\tau v}$) (*s1′*[*x1*::=*v*]$_{sv}$) *s* **proof** −
have ∗: [[*atom x2*]]*lst*. *s2* = [[*atom x*]]*lst*. *s* **using** *check-letI s-branch-s-branch-list.eq-iff* **by** *auto*
moreover have *τ3*[*x3*::=*v*]$_{\tau v}$ = *τ1′*[*x1*::=*v*]$_{\tau v}$ **using** *fun-ret-unique* ∗∗ *check-letI* **by** *metis*
moreover have *s3*[*x3*::=*v*]$_{sv}$ = (*s1′*[*x1*::=*v*]$_{sv}$) **using** *subst-v-flip-eq-two subst-v-s-def seq* **by** *metis*
ultimately show *?thesis* **using** *s-branch-s-branch-list.eq-iff* **by** *metis*

qed

ultimately show *?case* **using** *check-letI* **by** *auto*
qed(*auto+*)

lemma *fresh-subst-v-subst-b*:
fixes *x2*::*x* **and** *tm*::'*a*::{*has-subst-v*,*has-subst-b*} **and** *x*::*x*
assumes *supp tm* ⊆ { *atom bv*, *atom x* } **and** *atom x2* ♯ *v*
shows *atom x2* ♯ *tm*[*bv*::=*b*]$_b$[*x*::=*v*]$_v$
using *assms* **proof**(*cases x2=x*)
case *True*
then show *?thesis* **using** *fresh-subst-v-if assms* **by** *blast*
next
case *False*
hence *atom x2* ♯ *tm* **using** *assms fresh-def fresh-at-base* **by** *force*
hence *atom x2* ♯ *tm*[*bv*::=*b*]$_b$ **using** *assms fresh-subst-if x-fresh-b False* **by** *force*
then show *?thesis* **using** *fresh-subst-v-if assms* **by** *auto*
qed

lemma *preservation-poly-app*:
assumes
Some (*AF-fundef f* (*AF-fun-typ-some bv1* (*AF-fun-typ x1 b1 c1* τ*1'* *s1'*))) = *lookup-fun* Φ *f*
and (∀ *fd*∈*set* Φ. *check-fundef* Θ Φ *fd*)
shows Θ ; Φ ; *B* ; *G* ; Δ ⊢ *ss* ⇐ τ ⟹ *B* = {||} ⟹ *G* = *GNil* ⟹ *ss* = *LET x* = (*AE-appP f b' v*) *IN s* ⟹ Θ ; {||} ⊢$_{wf}$ *b'* ⟹
Θ ; Φ ; {||} ; *GNil* ; Δ ⊢ *LET x* : (τ*1'*[*bv1*::=*b'*]$_{\tau b}$[*x1*::=*v*]$_{\tau v}$) = (*s1'*[*bv1*::=*b'*]$_{sb}$[*x1*::=*v*]$_{sv}$) *IN s* ⇐ τ **and**
check-branch-s Θ Φ *B GNil* Δ *tid dc const v cs* τ ⟹ *True* **and**
check-branch-list Θ Φ *B* Γ Δ *tid dclist v css* τ ⟹ *True*
using *assms* **proof**(*nominal-induct* τ **and** τ **and** τ *avoiding*: *v x1 rule*: *check-s-check-branch-s-check-branch-list.strong-i*
case (*check-letI x2* Θ Φ *B* Γ Δ *e* τ *z s2 b c*)

hence *eq*: *e* = (*AE-appP f b' v*) **by** *simp*
hence ∗:Θ ; Φ ; {||} ;*GNil* ; Δ ⊢ (*AE-appP f b' v*) ⇒ {| *z* : *b* | *c* |} **using** *check-letI* **by** *auto*

then obtain *x3 b3 c3* τ*3 s3 bv3* **where**
∗∗:Θ ; {||} ; *GNil* ⊢$_{wf}$ Δ ∧ Θ ⊢$_{wf}$ Φ ∧ *Some* (*AF-fundef f* (*AF-fun-typ-some bv3* (*AF-fun-typ x3 b3 c3* τ*3 s3*))) = *lookup-fun* Φ *f* ∧
Θ ; {||} ; *GNil* ⊢ *v* ⇐ {| *x3* : *b3*[*bv3*::=*b'*]$_{bb}$ | *c3*[*bv3*::=*b'*]$_{cb}$ |} ∧ *atom x3* ♯ *GNil* ∧ τ*3*[*bv3*::=*b'*]$_{\tau b}$[*x3*::=*v*]$_{\tau v}$ = {| *z* : *b* | *c* |}
∧ Θ ; {||} ⊢$_{wf}$ *b'*
using *infer-e-elims*(*21*)[*OF* ∗] *subst-defs* **by** *metis*

obtain *z3* **where** *z3*:{| *x3* : *b3* | *c3* |} = {| *z3* : *b3* | *c3*[*x3*::=*V-var z3*]$_{cv}$ |} ∧ *atom z3* ♯ (*x3*, *v*,*c3*,*x1*,*c1*) **using** *obtain-fresh-z3* **by** *metis*

let *?ft* = (*AF-fun-typ x3* (*b3*[*bv3*::=*b'*]$_{bb}$) (*c3*[*bv3*::=*b'*]$_{cb}$) (τ*3*[*bv3*::=*b'*]$_{\tau b}$) (*s3*[*bv3*::=*b'*]$_{sb}$))

have ∗:*check-fundef* Θ Φ (*AF-fundef f* (*AF-fun-typ-some bv3* (*AF-fun-typ x3 b3 c3* τ*3 s3*))) **using** ∗∗ *check-letI lookup-fun-member* **by** *metis*

hence *ftq*:*check-funtypq* Θ Φ (*AF-fun-typ-some bv3* (*AF-fun-typ x3 b3 c3* τ*3 s3*)) **using** *check-fundef-elims*

by *auto*

let *?ft* = *AF-fun-typ-some bv3* (*AF-fun-typ x3 b3 c3 τ3 s3*)

have *sup*: *supp τ3* ⊆ { *atom x3*, *atom bv3*} ∧ *supp s3* ⊆ { *atom x3*, *atom bv3* } **using** *wfPhi-f-poly-supp-s wfPhi-f-poly-supp-t* ∗∗ **by** *force*

have Θ ; Φ ; {||} ; *GNil* ; Δ ⊢ *AS-let2 x2* $\tau 3[bv3::=b']_{\tau b}[x3::=v]_{\tau v}$ $(s3[bv3::=b']_{sb}[x3::=v]_{sv})$ *s2* ⇐ *τ*
proof
show ‹*atom x2* ♯ (Θ, Φ, {||}::*bv fset*, *GNil*, Δ, $\tau 3[bv3::=b']_{\tau b}[x3::=v]_{\tau v}$, $s3[bv3::=b']_{sb}[x3::=v]_{sv}$, *τ*)›
proof −

thm *fresh-subst-v-subst-b*
have *atom x2* ♯ $\tau 3[bv3::=b']_{\tau b}[x3::=v]_{\tau v}$
using *fresh-subst-v-subst-b subst-v-τ-def subst-b-τ-def* ‹ *atom x2* ♯ *v*› *sup* **by** *fastforce*
moreover have *atom x2* ♯ $s3[bv3::=b']_{sb}[x3::=v]_{sv}$
using *fresh-subst-v-subst-b subst-v-s-def subst-b-s-def* ‹ *atom x2* ♯ *v*› *sup*
proof −
have ∀ *b*. *atom x2* = *atom x3* ∨ *atom x2* ♯ $s3[bv3::=b]_b$
by (*metis* (*no-types*) *check-letI.hyps*(*1*) *fresh-subst-sv-if*(*1*) *fresh-subst-v-subst-b insert-commute subst-v-s-def sup*)
then show *?thesis*
by (*metis check-letI.hyps*(*1*) *fresh-subst-sb-if fresh-subst-sv-if*(*1*) *has-subst-b-class.subst-b-fresh-x x-fresh-b*)
qed
ultimately show *?thesis* **using** *fresh-prodN check-letI* **by** *metis*
qed

show ‹ Θ ; Φ ; {||} ; *GNil* ; Δ ⊢ $s3[bv3::=b']_{sb}[x3::=v]_{sv}$ ⇐ $\tau 3[bv3::=b']_{\tau b}[x3::=v]_{\tau v}$› **proof**(*rule fundef-poly-check*)
show ‹*check-fundef* Θ Φ (*AF-fundef f* (*AF-fun-typ-some bv3* (*AF-fun-typ x3 b3 c3 τ3 s3*)))›
using ∗∗ *lookup-fun-member check-letI* **by** *metis*
show ‹Θ ; {||} ; *GNil* ⊢ *v* ⇐ {‖ *x3* : *b3*$[bv3::=b']_{bb}$ | *c3*$[bv3::=b']_{cb}$ ‖}› **using** ∗∗ **by** *metis*
show ‹ Θ ; {||} ; *GNil* $\vdash_{wf}$ Δ › **using** ∗∗ **by** *metis*
show ‹ Θ ; {||} $\vdash_{wf}$ *b'* › **using** ∗∗ **by** *metis*
qed
show ‹ Θ ; Φ ; {||} ; (*x2*, *b-of* $\tau 3[bv3::=b']_{\tau b}[x3::=v]_{\tau v}$, *c-of* $\tau 3[bv3::=b']_{\tau b}[x3::=v]_{\tau v}$ *x2*) #Γ *GNil* ; Δ ⊢ *s2* ⇐ *τ*›
using *check-letI* ∗∗ *b-of.simps c-of.simps subst-defs* **by** *metis*
qed

moreover have *AS-let2 x2* $\tau 3[bv3::=b']_{\tau b}[x3::=v]_{\tau v}$ $(s3[bv3::=b']_{sb}[x3::=v]_{sv})$ *s2* = *AS-let2 x* $(\tau 1'[bv1::=b']_{\tau b}[x1::=v]_{\tau v})$ $(s1'[bv1::=b']_{sb}[x1::=v]_{sv})$ *s* **proof** −
have ∗: [[*atom x2*]]*lst*. *s2* = [[*atom x*]]*lst*. *s* **using** *check-letI s-branch-s-branch-list.eq-iff* **by** *auto*
moreover have $\tau 3[bv3::=b']_{\tau b}[x3::=v]_{\tau v}$ = $\tau 1'[bv1::=b']_{\tau b}[x1::=v]_{\tau v}$ **using** *fun-poly-ret-unique* ∗∗ *check-letI* **by** *metis*
moreover have $s3[bv3::=b']_{sb}[x3::=v]_{sv}$ = $(s1'[bv1::=b']_{sb}[x1::=v]_{sv})$ **using** *subst-v-flip-eq-two subst-v-s-def fun-poly-body-unique* ∗∗ *check-letI* **by** *metis*
ultimately show *?thesis* **using** *s-branch-s-branch-list.eq-iff* **by** *metis*
qed

ultimately show *?case* **using** *check-letI* **by** *auto*
qed(*auto+*)

lemma *check-s-plus*:
assumes Θ ; Φ ; {||} ; *GNil* ; Δ ⊢ *LET x* = (*AE-op Plus* (*V-lit* (*L-num n1*)) (*V-lit* (*L-num n2*))) *IN s′* ⇐ τ
shows Θ ; Φ ; {||} ; *GNil* ; Δ ⊢ *LET x* = (*AE-val* (*V-lit* (*L-num* (*n1*+*n2*)))) *IN s′* ⇐ τ
proof −
obtain *t1* **where** *1*: Θ ; Φ ; {||} ; *GNil* ; Δ ⊢ *AE-op Plus* (*V-lit* (*L-num n1*)) (*V-lit* (*L-num n2*)) ⇒ *t1*
using *assms check-s-elims* **by** *metis*
then obtain *z1* **where** *2*: *t1* = {| *z1* : *B-int* | *CE-val* (*V-var z1*) == *CE-op Plus* ([*V-lit* (*L-num n1*)]^{ce}) ([*V-lit* (*L-num n2*)]^{ce}) |}
using *infer-e-plus* **by** *metis*

obtain *z2* **where** *3*: ‹Θ ; Φ ; {||} ; *GNil* ; Δ ⊢ *AE-val* (*V-lit* (*L-num* (*n1*+*n2*))) ⇒ {| *z2* : *B-int* | *CE-val* (*V-var z2*) == *CE-val* (*V-lit* (*L-num* (*n1*+*n2*))) |}›
using *infer-v-form infer-e-valI infer-v-litI infer-l.intros infer-e-wf 1*
by (*simp add*: *fresh-GNil*)

thus *?thesis* **using** *subtype-let 1 2 subtype-bop infer-e-wf type-for-lit.simps*
by (*metis assms opp.distinct*(*1*) *type-l-eq*)
qed

lemma *check-s-leq*:
assumes Θ ; Φ ; {||} ; *GNil* ; Δ ⊢ *LET x* = (*AE-op LEq* (*V-lit* (*L-num n1*)) (*V-lit* (*L-num n2*))) *IN s′* ⇐ τ
shows Θ ; Φ ; {||} ; *GNil* ; Δ ⊢ *LET x* = (*AE-val* (*V-lit* (*if* (*n1* ≤ *n2*) *then L-true else L-false*))) *IN s′* ⇐ τ
proof −
obtain *t1* **where** *1*: Θ ; Φ ; {||} ; *GNil* ; Δ ⊢ *AE-op LEq* (*V-lit* (*L-num n1*)) (*V-lit* (*L-num n2*)) ⇒ *t1*
using *assms check-s-elims* **by** *metis*
then obtain *z1* **where** *2*: *t1* = {| *z1* : *B-bool* | *CE-val* (*V-var z1*) == *CE-op LEq* ([*V-lit* (*L-num n1*)]^{ce}) ([*V-lit* (*L-num n2*)]^{ce}) |}
using *infer-e-leq* **by** *auto*

obtain *z2* **where** *3*: ‹Θ ; Φ ; {||} ; *GNil* ; Δ ⊢ *AE-val* (*V-lit* ((*if* (*n1* ≤ *n2*) *then L-true else L-false*))) ⇒ {| *z2* : *B-bool* | *CE-val* (*V-var z2*) == *CE-val* (*V-lit* ((*if* (*n1* ≤ *n2*) *then L-true else L-false*))) |}›
using *infer-v-form infer-e-valI infer-v-litI infer-l.intros infer-e-wf 1*
fresh-GNil
by *simp*
thm *subtype-let*
show *?thesis* **proof**(*rule subtype-let*)
show ‹ Θ ; Φ ; {||} ; *GNil* ; Δ ⊢ *AS-let x* (*AE-op LEq* [*L-num n1*]^{v} [*L-num n2*]^{v}) *s′* ⇐ τ› **using** *assms* **by** *auto*
show ‹Θ ; Φ ; {||} ; *GNil* ; Δ ⊢ *AE-op LEq* [*L-num n1*]^{v} [*L-num n2*]^{v} ⇒ *t1*› **using** *1* **by** *auto*
show ‹Θ ; Φ ; {||} ; *GNil* ; Δ ⊢ [[*if n1* ≤ *n2 then L-true else L-false*]^{v}]^{e} ⇒ {| *z2* : *B-bool* | *CE-val* (*V-var z2*) == *CE-val* (*V-lit* ((*if* (*n1* ≤ *n2*) *then L-true else L-false*))) |}› **using** *3* **by** *auto*
show ‹Θ ; {||} ; *GNil* ⊢ {| *z2* : *B-bool* | *CE-val* (*V-var z2*) == *CE-val* (*V-lit* ((*if* (*n1* ≤ *n2*)

then L-true else L-false))) ⟧ ≲ *t1*›
using *subtype-bop*[**where** *opp=LEq*] *check-s-wf assms 2*
by (*metis opp.distinct*(*1*) *subtype-bop type-l-eq*)
qed

qed

lemma *preservation-plus*:
assumes Θ ; Φ ; Δ ⊢ ⟨ δ , *LET x* = (*AE-op Plus* (*V-lit* (*L-num n1*)) (*V-lit* (*L-num n2*))) *IN s'* ⟩ ⇐ τ
shows Θ ; Φ ; Δ ⊢ ⟨ δ , *LET x* = (*AE-val* (*V-lit* (*L-num* (*n1*+*n2*)))) *IN s'* ⟩ ⇐ τ
proof −

have *tt*: Θ ; Φ ; {||} ; *GNil* ; Δ ⊢ *AS-let x* (*AE-op Plus* (*V-lit* (*L-num n1*)) (*V-lit* (*L-num n2*))) *s'* ⇐ τ **and** *dsim*: Θ ⊢ δ ∼ Δ **and** *fd*:(∀ *fd*∈*set* Φ. *check-fundef* Θ Φ *fd*)
using *assms config-type-elims* **by** *blast*+

hence Θ ; Φ ; {||} ; *GNil* ; Δ ⊢*AS-let x* (*AE-val* (*V-lit* (*L-num* (*n1*+*n2*)))) *s'* ⇐ τ **using** *check-s-plus assms* **by** *auto*

hence Θ ; Φ ; Δ ⊢ ⟨ δ , *AS-let x* (*AE-val* (*V-lit* ((*L-num* (*n1*+*n2*))))) *s'* ⟩ ⇐ τ **using** *dsim config-typeI fd* **by** *presburger*
then show *?thesis* **using** *dsim config-typeI*
by (*meson order-refl*)
qed

lemma *preservation-leq*:
assumes Θ ; Φ ; Δ ⊢ ⟨ δ , *AS-let x* (*AE-op LEq* (*V-lit* (*L-num n1*)) (*V-lit* (*L-num n2*))) *s'* ⟩ ⇐ τ

shows Θ ; Φ ; Δ ⊢ ⟨ δ , *AS-let x* (*AE-val* (*V-lit* (((*if* (*n1* ≤ *n2*) *then L-true else L-false*))))) *s'* ⟩ ⇐ τ
proof −

have *tt*: Θ ; Φ ; {||} ; *GNil* ; Δ ⊢ *AS-let x* (*AE-op LEq* (*V-lit* (*L-num n1*)) (*V-lit* (*L-num n2*))) *s'* ⇐ τ **and** *dsim*: Θ ⊢ δ ∼ Δ **and** *fd*:(∀ *fd*∈*set* Φ. *check-fundef* Θ Φ *fd*)
using *assms config-type-elims* **by** *blast*+

hence Θ ; Φ ; {||} ; *GNil* ; Δ ⊢*AS-let x* (*AE-val* (*V-lit* (((*if* (*n1* ≤ *n2*) *then L-true else L-false*))))) *s'* ⇐ τ **using** *check-s-leq assms* **by** *auto*

hence Θ ; Φ ; Δ ⊢ ⟨ δ , *AS-let x* (*AE-val* (*V-lit* ((((*if* (*n1* ≤ *n2*) *then L-true else L-false*)))))) *s'* ⟩ ⇐ τ **using** *dsim config-typeI fd* **by** *presburger*
then show *?thesis* **using** *dsim config-typeI*
by (*meson order-refl*)
qed

lemma *subst-s-abs-lst*:
fixes *s*::*s* **and** *sa*::*s* **and** *v'*::*v*
assumes [[*atom x*]]*lst*. *s* = [[*atom xa*]]*lst*. *sa* **and** *atom xa* ♯ *v* ∧ *atom x* ♯ *v*
shows *s*[*x*::=*v*]$_{sv}$ = *sa*[*xa*::=*v*]$_{sv}$
proof −

obtain $c'::x$ **where** *cdash*: *atom* $c' \sharp (v, x, xa, s, sa)$ **using** *obtain-fresh* **by** *blast*
moreover have $(x \leftrightarrow c') \cdot s = (xa \leftrightarrow c') \cdot sa$ **proof** −
have *atom* $c' \sharp (s, sa) \wedge$ *atom* $c' \sharp (x, xa, s, sa)$ **using** *cdash* **by** *auto*
thus *?thesis* **using** *assms* **by** *auto*
qed
ultimately show *?thesis* **using** *assms*
using *subst-sv-flip* **by** *auto*
qed

lemma *check-let-val*:
fixes $v::v$ **and** $s::s$
shows $\Theta\ ;\ \Phi\ ;\ B\ ;\ G\ ;\ \Delta\ \vdash ss \Leftarrow \tau \Longrightarrow\ B = \{||\} \Longrightarrow G = GNil \Longrightarrow$
$ss = AS\text{-}let\ x\ (AE\text{-}val\ v)\ s \vee\ ss = AS\text{-}let2\ x\ t\ (AS\text{-}val\ v)\ s \Longrightarrow\ \Theta\ ;\ \Phi\ ;\ \{||\}\ ;\ GNil\ ;\ \Delta\ \vdash (s[x::=v]_{sv}) \Leftarrow \tau$ **and**
check-branch-s $\Theta\ \Phi\ \mathcal{B}\ GNil\ \Delta\ tid\ dc\ const\ v\ cs\ \tau \Longrightarrow True$ **and**
check-branch-list $\Theta\ \Phi\ \mathcal{B}\ \Gamma\ \Delta\ tid\ dclist\ v\ css\ \tau \Longrightarrow True$
proof(*nominal-induct* τ **and** τ **and** τ *avoiding*: *v rule*: *check-s-check-branch-s-check-branch-list.strong-induct*)
case (*check-letI x1* $\Theta\ \Phi\ \mathcal{B}\ \Gamma\ \Delta\ e\ \tau\ z\ s1\ b\ c$)
hence $*:e = AE\text{-}val\ v$ **by** *auto*
let *?G* $= (x1,\ b,\ c[z::=V\text{-}var\ x1]_{cv})\ \#_\Gamma\ \Gamma$
have $\Theta\ ;\ \Phi\ ;\ \mathcal{B}\ ;\ \ ?G[x1::=v]_{\Gamma v}\ ;\ \Delta[x1::=v]_{\Delta v}\ \vdash s1[x1::=v]_{sv} \Leftarrow \tau[x1::=v]_{\tau v}$
proof(*rule subst-infer-check-s*(*1*))
show $**:\langle\ \Theta\ ;\ \mathcal{B}\ ;\ \Gamma \vdash v \Rightarrow\ \{\!|\ z : b\ |\ c\ |\!\}\rangle$ **using** *infer-e-elims check-letI* $*$ **by** *fast*
thus $\langle \Theta\ ;\ \mathcal{B}\ ;\ \Gamma\ \vdash\ \{\!|\ z : b\ |\ c\ |\!\} \lesssim \{\!|\ z : b\ |\ c\ |\!\}\rangle$ **using** *subtype-reflI infer-v-wf* **by** *metis*
show $\langle atom\ z \sharp (x1,\ v)\rangle$ **using** *check-letI fresh-Pair* **by** *auto*
show $\langle \Theta\ ;\ \Phi\ ;\ \mathcal{B}\ ;\ (x1,\ b,\ c[z::=V\text{-}var\ x1]_{cv})\ \#_\Gamma\ \Gamma\ ;\ \Delta\ \vdash s1 \Leftarrow \tau\rangle$ **using** *check-letI subst-defs* **by** *auto*
show $(x1,\ b,\ c[z::=V\text{-}var\ x1]_{cv})\ \#_\Gamma\ \Gamma = GNil\ @\ (x1,\ b,\ c[z::=V\text{-}var\ x1]_{cv})\ \#_\Gamma\ \Gamma$ **by** *auto*
qed

hence $\Theta\ ;\ \Phi\ ;\ \mathcal{B}\ ;\ \ \Gamma\ ;\ \Delta\ \vdash s1[x1::=v]_{sv} \Leftarrow \tau$ **using** *check-letI* **by** *auto*
moreover have $s1[x1::=v]_{sv} = s[x::=v]_{sv}$
by (*metis* (*full-types*) *check-letI fresh-GNil infer-e-elims*(*7*) *s-branch-s-branch-list.distinct s-branch-s-branch-list.eq-iff*(

subst-s-abs-lst wfG-x-fresh-in-v-simple)

ultimately show *?case* **using** *check-letI* **by** *simp*
next
case (*check-let2I x1* $\Theta\ \Phi\ \mathcal{B}\ \Gamma\ \Delta\ t\ s1\ \tau\ s2$)

hence *s1eq*:$s1 = AS\text{-}val\ v$ **by** *auto*
let *?G* $= (x1,\ b\text{-}of\ t,\ c\text{-}of\ t\ x1)\ \#_\Gamma\ \Gamma$
obtain $z::x$ **where** $*:atom\ z \sharp (x1\ ,\ v,t)$ **using** *obtain-fresh-z* **by** *metis*
hence *teq*:$t = \{\!|\ z : b\text{-}of\ t\ |\ c\text{-}of\ t\ z\ |\!\}$ **using** *b-of-c-of-eq* **by** *auto*
have $\Theta\ ;\ \Phi\ ;\ \mathcal{B}\ ;\ \ ?G[x1::=v]_{\Gamma v}\ ;\ \Delta[x1::=v]_{\Delta v}\ \vdash s2[x1::=v]_{sv} \Leftarrow \tau[x1::=v]_{\tau v}$
proof(*rule subst-check-check-s*(*1*))
obtain t' **where** $\Theta\ ;\ \mathcal{B}\ ;\ \Gamma \vdash v \Rightarrow t' \wedge\ \Theta\ ;\ \mathcal{B}\ ;\ \Gamma \vdash t' \lesssim t$ **using** *check-s-elims*(*1*) *check-let2I*(*10*) *s1eq* **by** *auto*
thus $**:\langle\ \Theta\ ;\ \mathcal{B}\ ;\ \Gamma \vdash v \Leftarrow \{\!|\ z : b\text{-}of\ t\ |\ c\text{-}of\ t\ z\ |\!\}\rangle$ **using** *check-v.intros teq* **by** *auto*
show *atom* $z \sharp (x1,\ v)$ **using** $*$ **by** *auto*
show $\langle\ \Theta\ ;\ \Phi\ ;\ \mathcal{B}\ ;\ (x1,\ b\text{-}of\ t,\ c\text{-}of\ t\ x1)\ \#_\Gamma\ \Gamma\ ;\ \Delta\ \vdash s2 \Leftarrow \tau\rangle$ **using** *check-let2I* **by** *auto*

show (*x1*, *b-of t* , *c-of t x1*) $\#_\Gamma$ Γ = *GNil* @ (*x1*, *b-of t*, (*c-of t z*)[*z*::=*V-var x1*]$_{cv}$) $\#_\Gamma$ Γ **using** *append-g.simps c-of-switch* ∗ *fresh-prodN* **by** *metis*
qed

hence Θ ; Φ ; $\mathcal{B}$; Γ ; Δ ⊢ *s2*[*x1*::=*v*]$_{sv}$ ⇐ τ **using** *check-let2I* **by** *auto*
moreover have *s2*[*x1*::=*v*]$_{sv}$ = *s*[*x*::=*v*]$_{sv}$ **using**
check-let2I fresh-GNil check-s-elims s-branch-s-branch-list.distinct s-branch-s-branch-list.eq-iff subst-s-abs-lst wfG-x-fresh-in-v-simple
proof −
have *AS-let2 x t* (*AS-val v*) *s* = *AS-let2 x1 t s1 s2*
by (*metis check-let2I.prems*(*3*) *s-branch-s-branch-list.distinct s-branch-s-branch-list.eq-iff*(*3*))
then show *?thesis*
by (*metis* (*no-types*) *check-let2I check-let2I.prems*(*2*) *check-s-elims*(*1*) *fresh-GNil s-branch-s-branch-list.eq-iff*(*3*) *subst-s-abs-lst wfG-x-fresh-in-v-simple*)
qed

ultimately show *?case* **using** *check-let2I* **by** *simp*

qed(*auto*+)

lemma *preservation-let-val*:
assumes Θ ; Φ ; Δ ⊢ ⟨ δ , *AS-let x* (*AE-val v*) *s* ⟩ ⇐ τ ∨ Θ ; Φ ; Δ ⊢ ⟨ δ , *AS-let2 x t* (*AS-val v*) *s* ⟩ ⇐ τ (**is** *?A* ∨ *?B*)
shows ∃Δ'. Θ ; Φ ; Δ' ⊢ ⟨ δ , *s*[*x*::=*v*]$_{sv}$ ⟩ ⇐ τ ∧ *setD* Δ ⊆ *setD* Δ'
proof −
have *tt*: Θ ⊢ δ ∼ Δ **and** *fd*: (∀*fd*∈*set* Φ. *check-fundef* Θ Φ *fd*)
using *assms* **by** *blast*+

have *?A* ∨ *?B* **using** *assms* **by** *auto*
then have Θ ; Φ ; {||} ; *GNil* ; Δ ⊢ *s*[*x*::=*v*]$_{sv}$ ⇐ τ
proof
assume Θ ; Φ ; Δ ⊢ ⟨ δ , *AS-let x* (*AE-val v*) *s* ⟩ ⇐ τ
hence ∗ : Θ ; Φ ; {||} ; *GNil* ; Δ ⊢ *AS-let x* (*AE-val v*) *s* ⇐ τ **by** *blast*
thus *?thesis* **using** *check-let-val* **by** *simp*
next
assume Θ ; Φ ; Δ ⊢ ⟨ δ , *AS-let2 x t* (*AS-val v*) *s* ⟩ ⇐ τ
hence ∗ : Θ ; Φ ; {||} ; *GNil* ; Δ ⊢ *AS-let2 x t* (*AS-val v*) *s* ⇐ τ **by** *blast*
thus *?thesis* **using** *check-let-val* **by** *simp*
qed

thus *?thesis* **using** *tt config-typeI fd*
order-refl **by** *metis*
qed

lemma *check-s-fst-snd*:
assumes *fst-snd* = *AE-fst* ∧ *v*=*v1* ∨ *fst-snd* = *AE-snd* ∧ *v*=*v2*
and Θ ; Φ ; {||} ; *GNil* ; Δ ⊢ *AS-let x* (*fst-snd* (*V-pair v1 v2*)) *s'* ⇐ τ
shows Θ ; Φ ; {||} ; *GNil* ; Δ ⊢ *AS-let x* (*AE-val v*) *s'* ⇐ τ
proof −
have *1*: ‹ Θ ; Φ ; {||} ; *GNil* ; Δ ⊢ *AS-let x* (*fst-snd* (*V-pair v1 v2*)) *s'* ⇐ τ› **using** *assms* **by** *auto*

then obtain *t1* **where** *2*:Θ ; Φ ; $\{||\}$; *GNil* ; Δ $\vdash$ (*fst-snd* (*V-pair v1 v2*)) $\Rightarrow$ *t1* **using** *check-s-elims*
by *auto*

show *?thesis* **using** *subtype-let 1 2 assms*
by (*meson infer-e-fst-pair infer-e-snd-pair*)
qed

lemma *preservation-fst-snd*:
assumes Θ ; Φ ; $\Delta \vdash \langle\ \delta\ ,\ LET\ x = (fst\text{-}snd\ (V\text{-}pair\ v1\ v2))\ IN\ s'\ \rangle \Leftarrow \tau$ **and**
fst-snd = *AE-fst* $\wedge$ *v*=*v1* $\vee$ *fst-snd* = *AE-snd* $\wedge$ *v*=*v2*
shows $\exists \Delta'.\ \Theta$; Φ ; $\Delta \vdash \langle\ \delta\ ,\ LET\ x = (AE\text{-}val\ v)\ IN\ s'\ \rangle \Leftarrow \tau\ \wedge\ setD\ \Delta \subseteq setD\ \Delta'$
proof −
have *tt*: Θ ; Φ ; $\{||\}$; *GNil* ; Δ $\vdash$ *AS-let x* (*fst-snd* (*V-pair v1 v2*)) *s'* $\Leftarrow \tau \wedge \Theta \vdash \delta \sim \Delta$ **using**
assms **by** *blast*
hence *t2*: Θ ; Φ ; $\{||\}$; *GNil* ; Δ $\vdash$ *AS-let x* (*fst-snd* (*V-pair v1 v2*)) *s'* $\Leftarrow \tau$ **by** *auto*

moreover have $\forall$ *fd*$\in$*set* Φ. *check-fundef* Θ Φ *fd* **using** *assms config-type-elims* **by** *auto*
ultimately show *?thesis* **using** *config-typeI order-refl tt assms check-s-fst-snd* **by** *metis*
qed

inductive-cases *check-branch-s-elims2*[*elim*!]:
Θ ; Φ ; $\mathcal{B}$; Γ ; Δ; *tid* ; *cons* ; *const* ; *v* $\vdash$ *cs* $\Leftarrow \tau$

lemmas *freshers* = *freshers atom-dom.simps setG.simps fresh-def x-not-in-b-set*
declare *freshers* [*simp*]

lemma *subtype-eq-if*:
fixes *t*::τ **and** *va*::*v*
assumes Θ ; $\mathcal{B}$; $\Gamma \vdash_{wf}$ $\{\!|$ *z* : *b-of t* | *c-of t z* $|\!\}$ **and** Θ ; $\mathcal{B}$; $\Gamma \vdash_{wf} \{\!|$ *z* : *b-of t* | *c IMP c-of t z* $|\!\}$
shows Θ ; $\mathcal{B}$; $\Gamma \vdash \{\!|$ *z* : *b-of t* | *c-of t z* $|\!\} \lesssim \{\!|$ *z* : *b-of t* | *c IMP c-of t z* $|\!\}$
proof −
obtain *x*::*x* **where** *xf*:*atom x* $\sharp$ ((Θ, $\mathcal{B}$, Γ, *z*, *c-of t z*, *z*, *c IMP c-of t z*),*c*) **using** *obtain-fresh* **by**
metis

moreover have Θ ; $\mathcal{B}$; (*x*, *b-of t*, (*c-of t z*)[*z*::=[*x*]^v]$_{cv}$) #$_\Gamma$ Γ $\models$ (*c IMP c-of t z*)[*z*::=[*x*]^v]$_{cv}$
unfolding *subst-cv.simps*
proof(*rule valid-eq-imp*)

have Θ ; $\mathcal{B}$; (*x*, *b-of t*, (*c-of t z*)[*z*::=[*x*]^v]$_v$) #$_\Gamma$ Γ $\vdash_{wf}$ (*c IMP* (*c-of t z*))[*z*::=[*x*]^v]$_v$
apply(*rule wfT-wfC-cons*)
apply(*simp add*: *assms*, *simp add*: *assms*, *unfold fresh-prodN*)
using *xf fresh-prodN* **by** *metis*+
thus Θ ; $\mathcal{B}$; (*x*, *b-of t*, (*c-of t z*)[*z*::=[*x*]^v]$_{cv}$) #$_\Gamma$ Γ $\vdash_{wf}$ *c*[*z*::=[*x*]^v]$_{cv}$ *IMP* (*c-of t z*)[*z*::=[*x*]^v]$_{cv}$
using *subst-cv.simps subst-defs* **by** *auto*
qed

ultimately show *?thesis* **using** *subtype-baseI assms fresh-Pair subst-defs* **by** *metis*
qed

lemma *subtype-eq-if-τ*:
fixes $t::\tau$ **and** $va::v$
assumes $\Theta\ ;\ \mathcal{B}\ ;\ \Gamma \vdash_{wf}\ t$ **and** $\Theta\ ;\ \mathcal{B}\ ;\ \Gamma \vdash_{wf} \{\!|\ z : b\text{-}of\ t\ \mid c\ \ IMP\ c\text{-}of\ t\ z\ |\!\}$ **and** $atom\ z \sharp t$
shows $\Theta\ ;\ \mathcal{B}\ ;\ \Gamma \vdash t \lesssim \{\!|\ z : b\text{-}of\ t\ \mid c\ \ IMP\ c\text{-}of\ t\ z\ |\!\}$
proof −
have $t = \{\!|\ z : b\text{-}of\ t \mid c\text{-}of\ t\ z\ |\!\}$ **using** *b-of-c-of-eq assms* **by** *auto*
thus *?thesis* **using** *subtype-eq-if assms c-of.simps b-of.simps* **by** *metis*
qed

lemma *valid-conj*:
assumes $\Theta\ ;\ \mathcal{B}\ ;\ \Gamma \models c1$ **and** $\Theta\ ;\ \mathcal{B}\ ;\ \Gamma \models c2$
shows $\Theta\ ;\ \mathcal{B}\ ;\ \Gamma \models c1\ AND\ c2$
proof
show ‹ $\Theta\ ;\ \mathcal{B}\ ;\ \Gamma\ \ \vdash_{wf} c1\ \ AND\ \ c2$ › **using** *valid.simps wfC-conjI assms* **by** *auto*
show ‹$\forall i.\ \ \Theta\ ;\ \Gamma \vdash i \wedge\ \ i \models \Gamma\ \longrightarrow\ \ i \models c1\ \ AND\ \ c2$ ›
proof(*rule*+)
fix i
assume $*{:}\Theta\ ;\ \Gamma \vdash i \wedge i \models \Gamma$
thus $i\ [\![\ c1\ ]\!]\ \sim\ True$ **using** *assms valid.simps*
using *is-satis.cases* **by** *blast*
show $i\ [\![\ c2\ ]\!]\ \sim\ True$ **using** *assms valid.simps*
using *is-satis.cases* $*$ **by** *blast*
qed
qed

lemma *wfT-conj*:
assumes $\Theta\ ;\ \mathcal{B}\ ;\ GNil \vdash_{wf} \{\!|\ z : b \mid c1\ |\!\}$ **and** $\Theta\ ;\ \mathcal{B}\ ;\ GNil \vdash_{wf} \{\!|\ z : b \mid c2\ |\!\}$
shows $\Theta\ ;\ \mathcal{B}\ ;\ GNil \vdash_{wf} \{\!|\ z : b \mid c1\ AND\ c2|\!\}$
proof
show ‹$atom\ z \sharp (\Theta,\ \mathcal{B},\ GNil)$›
apply(*unfold fresh-prodN*, *intro conjI*)
using *wfTh-fresh wfT-wf assms* **apply** *metis*
using *fresh-GNil x-not-in-b-set fresh-def* **by** *metis*+
show ‹ $\Theta\ ;\ \mathcal{B}\ \vdash_{wf} b$ › **using** *wfT-elims assms* **by** *metis*
have $\Theta\ ;\ \mathcal{B}\ ;\ (z,\ b,\ TRUE)\ \#_\Gamma\ GNil\ \ \vdash_{wf} c1$ **using** *wfT-wfC fresh-GNil assms* **by** *auto*
moreover have $\Theta\ ;\ \mathcal{B}\ ;\ (z,\ b,\ TRUE)\ \#_\Gamma\ GNil\ \ \vdash_{wf} c2$ **using** *wfT-wfC fresh-GNil assms* **by** *auto*
ultimately show ‹ $\Theta\ ;\ \mathcal{B}\ ;\ (z,\ b,\ TRUE)\ \#_\Gamma\ GNil\ \ \vdash_{wf} c1\ \ AND\ \ c2$ › **using** *wfC-conjI* **by** *auto*
qed

lemma *subtype-conj*:
assumes $\Theta\ ;\ \mathcal{B}\ ;\ GNil\ \vdash t \lesssim \{\!|\ z : b \mid c1\ |\!\}$ **and** $\Theta\ ;\ \mathcal{B}\ ;\ GNil\ \vdash t \lesssim \{\!|\ z : b \mid c2\ |\!\}$
shows $\Theta\ ;\ \mathcal{B}\ ;\ GNil\ \vdash \{\!|\ z : b \mid c\text{-}of\ t\ z\ |\!\}\ \lesssim \{\!|\ z : b \mid c1\ AND\ c2\ |\!\}$
proof −
have *beq*: $b\text{-}of\ t = b$ **using** *subtype-eq-base2 b-of.simps assms* **by** *metis*
obtain $x::x$ **where** $x{:}‹atom\ x \sharp (\Theta,\ \mathcal{B},\ GNil,\ z,\ c\text{-}of\ t\ z,\ z,\ c1\ \ AND\ \ c2\ )›$ **using** *obtain-fresh* **by** *metis*
thus *?thesis* **proof**
have $atom\ z \sharp t$ **using** *subtype-wf wfT-supp fresh-def x-not-in-b-set atom-dom.simps setG.simps assms* **by** *blast*
hence $t{:}t = \{\!|\ z : b\text{-}of\ t\ \mid c\text{-}of\ t\ z\ |\!\}$ **using** *b-of-c-of-eq* **by** *auto*

```
    thus ‹ Θ ; B ; GNil ⊢wf {| z : b | c-of t z |} › using subtype-wf beq assms by auto

    show ‹Θ ; B ; (x, b, (c-of t z)[z::=[ x ]v]v) #Γ GNil ⊨ (c1 AND c2 )[z::=[ x ]v]v ›
    proof −
      have ‹Θ ; B ; (x, b, (c-of t z)[z::=[ x ]v]v) #Γ GNil ⊨ c1[z::=[ x ]v]v ›
      proof(rule subtype-valid)
        show ‹Θ ; B ; GNil ⊢ t ≲ {| z : b | c1 |}› using assms by auto
        show ‹atom x ♯ GNil› using fresh-GNil by auto
        show ‹t = {| z : b | c-of t z |}› using t beq by auto
        show ‹{| z : b | c1 |} = {| z : b | c1 |}› by auto
      qed
      moreover have ‹Θ ; B ; (x, b, (c-of t z)[z::=[ x ]v]v) #Γ GNil ⊨ c2[z::=[ x ]v]v ›
      proof(rule subtype-valid)
        show ‹Θ ; B ; GNil ⊢ t ≲ {| z : b | c2 |}› using assms by auto
        show ‹atom x ♯ GNil› using fresh-GNil by auto
        show ‹t = {| z : b | c-of t z |}› using t beq by auto
        show ‹{| z : b | c2 |} = {| z : b | c2 |}› by auto
      qed
      ultimately show ?thesis unfolding subst-cv.simps subst-v-c-def using valid-conj by metis
    qed
    thus ‹ Θ ; B ; GNil ⊢wf {| z : b | c1 AND c2 |} › using subtype-wf wfT-conj assms by auto
  qed
qed


lemma infer-v-conj:
  assumes Θ ; B ; GNil ⊢ v ⇐ {| z : b | c1 |} and Θ ; B ; GNil ⊢ v ⇐ {| z : b | c2 |}
  shows Θ ; B ; GNil ⊢ v ⇐ {| z : b | c1 AND c2 |}
proof −
  obtain t1 where t1: Θ ; B ; GNil ⊢ v ⇒ t1 ∧ Θ ; B ; GNil ⊢ t1 ≲ {| z : b | c1 |}
    using assms check-v-elims by metis
  obtain t2 where t2: Θ ; B ; GNil ⊢ v ⇒ t2 ∧ Θ ; B ; GNil ⊢ t2 ≲ {| z : b | c2 |}
    using assms check-v-elims by metis
  have teq: t1 = {| z : b | c-of t1 z |} using subtype-eq-base2 b-of.simps
    by (metis (full-types) b-of-c-of-eq fresh-GNil infer-v-t-wf t1 wfT-x-fresh)
  have t1 = t2 using infer-v-uniqueness t1 t2 by auto
  hence Θ ; B ; GNil ⊢ {| z : b | c-of t1 z |} ≲ {| z : b | c1 AND c2 |} using subtype-conj t1 t2 by simp
  hence Θ ; B ; GNil ⊢ t1 ≲ {| z : b | c1 AND c2 |} using teq by auto
  thus ?thesis using t1 using check-v.intros by auto
qed

lemma wfG-conj:
  fixes c1::c
  assumes Θ ; B ⊢wf (x, b, c1 AND c2) #Γ Γ
  shows Θ ; B ⊢wf (x, b, c1) #Γ Γ
proof(cases c1 ∈ {TRUE, FALSE})
  case True
  then show ?thesis using assms wfG-cons2I wfG-elims wfX-wfY by metis
next
  case False
  then show ?thesis using assms wfG-cons1I wfG-elims wfX-wfY wfC-elims
    by (metis wfG-elim2)
```

qed

lemma *check-match*:
fixes $s'::s$ **and** $s::s$ **and** *css*::*branch-list* **and** *cs*::*branch-s*
shows $\Theta\ ;\ \Phi\ ;\ \mathcal{B}\ ;\ \Gamma\ ;\ \Delta \vdash\ s \Leftarrow \tau \Longrightarrow$ *True* **and**
$\Theta\ ;\ \Phi\ ;\ B\ ;\ G\ ;\ \Delta\ ;$ *tid* ; *dc* ; *const* ; *vcons* $\vdash$ *cs* $\Leftarrow\ \tau \Longrightarrow$
vcons = *V-cons tid dc v* $\Longrightarrow$ $B = \{||\} \Longrightarrow G =$ *GNil* $\Longrightarrow$ *cs* = $(dc\ x' \Rightarrow s') \Longrightarrow$
$\Theta\ ;\ \{||\}\ ;$ *GNil* $\vdash v \Leftarrow$ *const* $\Longrightarrow$
$\Theta\ ;\ \Phi\ ;\ \{||\}\ ;$ *GNil* ; $\Delta\ \vdash s'[x'::=v]_{sv} \Leftarrow \tau$ **and**
$\Theta\ ;\ \Phi\ ;\ B\ ;\ G\ ;\ \Delta\ ;$ *tid* ; *dclist* ; *vcons* $\vdash$ *css* $\Leftarrow \tau \Longrightarrow$ *distinct* (*map fst dclist*) $\Longrightarrow$
vcons = *V-cons tid dc v* $\Longrightarrow$ $B = \{||\} \Longrightarrow (dc,$ *const*$) \in$ *set dclist* $\Longrightarrow G =$ *GNil* $\Longrightarrow$
Some (*AS-branch dc x' s'*) = *lookup-branch dc css* $\Longrightarrow\ \Theta\ ;\ \{||\}\ ;$ *GNil* $\vdash v \Leftarrow$ *const* $\Longrightarrow$
$\Theta\ ;\ \Phi\ ;\ \{||\}\ ;$ *GNil* ; $\Delta\ \vdash s'[x'::=v]_{sv} \Leftarrow \tau$
proof(*nominal-induct* τ **and** τ **and** τ *avoiding*: x' *v rule*: *check-s-check-branch-s-check-branch-list.strong-induct*)
case (*check-branch-list-consI* $\Theta\ \Phi\ \mathcal{B}\ \Gamma\ \Delta$ *tid consa consta va cs* τ *dclist cssa*)

then obtain *xa* **and** *sa* **where** *cseq*:*cs* = *AS-branch consa xa sa* **using** *check-branch-s-elims2*[*OF check-branch-list-consI*(*1*)] **by** *metis*

show *?case* **proof**(*cases dc* = *consa*)
case *True*
hence *cs* = *AS-branch consa x' s'* **using** *check-branch-list-consI cseq*
by (*metis lookup-branch.simps*(*2*) *option.inject*)
moreover have *const* = *consta* **using** *check-branch-list-consI distinct.simps*
by (*metis True dclist-distinct-unique list.set-intros*(*1*))
moreover have *va* = *V-cons tid consa v* **using** *check-branch-list-consI True* **by** *auto*
ultimately show *?thesis* **using** *check-branch-list-consI* **by** *auto*
next
case *False*
hence *Some* (*AS-branch dc x' s'*) = *lookup-branch dc cssa* **using** *lookup-branch.simps*(*2*) *check-branch-list-consI*(*10*) *cseq* **by** *auto*
moreover have (*dc*, *const*) $\in$ *set dclist* **using** *check-branch-list-consI False* **by** *simp*
ultimately show *?thesis* **using** *check-branch-list-consI* **by** *auto*
qed

next
case (*check-branch-list-finalI* $\Theta\ \Phi\ \mathcal{B}\ \Gamma\ \Delta$ *tid cons const va cs* τ)
hence *cs* = *AS-branch cons x' s'* **using** *lookup.simps check-branch-list-finalI lookup-branch.simps option.inject*
by (*metis map-of.simps*(*1*) *map-of-Cons-code*(*2*) *option.distinct*(*1*) *s-branch-s-branch-list.exhaust*(*2*) *weak-map-of-SomeI*)
then show *?case* **using** *check-branch-list-finalI* **by** *auto*
next
case (*check-branch-s-branchI* $\Theta\ \mathcal{B}\ \Gamma\ \Delta\ \tau$ *const x* Φ *tid cons va s*)

Supporting facts here to make the main body of the proof concise

have *xf*:*atom x* $\sharp\ \tau$ **proof** −
have *supp* $\tau \subseteq$ *supp* $\mathcal{B}$ **using** *wf-supp*(*4*) *check-branch-s-branchI atom-dom.simps setG.simps* **by** *blast*
thus *?thesis* **using** *fresh-def x-not-in-b-set* **by** *blast*

qed

hence $\tau[x::=v]_{\tau v} = \tau$ **using** *forget-subst-v subst-v-τ-def* **by** *auto*
have $\Delta[x::=v]_{\Delta v} = \Delta$ **using** *forget-subst-dv wfD-x-fresh fresh-GNil check-branch-s-branchI* **by** *metis*

have *supp v* = {} **using** *check-branch-s-branchI check-v-wf wfV-supp-nil* **by** *metis*
hence *supp va* = {} **using** ‹ *va* = *V-cons tid cons v*› *v.supp pure-supp* **by** *auto*

let *?G* = (*x*, *b-of const*, [*va*]^{ce} == [*V-cons tid cons* [*x*]^{v}]^{ce} *AND c-of const x*) #$_{\Gamma}$ Γ
obtain *z*::*x* **where** *z*: *const* = {‖ *z* : *b-of const* | *c-of const z* ‖} ∧ *atom z* ♯ (*x'*, *v*,*x*,*const*,*va*)
using *obtain-fresh-z-c-of* **by** *metis*

thm *check-branch-s-branchI*(*23*)

have *vt*: ‹Θ ; $\mathcal{B}$; *GNil* ⊢ *v* ⇐ {‖ *z* : *b-of const* | [*va*]^{ce} == [*V-cons tid cons* [*z*]^{v}]^{ce} *AND c-of const z* ‖}›
proof(*rule infer-v-conj*)
obtain *t'* **where** *t*: Θ ; $\mathcal{B}$; *GNil* ⊢ *v* ⇒ *t'* ∧ Θ ; $\mathcal{B}$; *GNil* ⊢ *t'* ≲ *const*
using *check-v-elims check-branch-s-branchI* **by** *metis*
show Θ ; $\mathcal{B}$; *GNil* ⊢ *v* ⇐ {‖ *z* : *b-of const* | [*va*]^{ce} == [*V-cons tid cons* [*z*]^{v}]^{ce} ‖}
proof(*rule check-v-top*)

show Θ ; $\mathcal{B}$; *GNil* ⊢$_{wf}$ {‖ *z* : *b-of const* | [*va*]^{ce} == [*V-cons tid cons* [*z*]^{v}]^{ce} ‖}

proof(*rule wfG-wfT*)
show ‹ Θ ; $\mathcal{B}$ ⊢$_{wf}$ (*x*, *b-of const*, ([*va*]^{ce} == [*V-cons tid cons* [*z*]^{v}]^{ce})[*z*::=[*x*]^{v}]$_{cv}$) #$_{\Gamma}$ *GNil* ›
proof −
have *1*: *va*[*z*::=[*x*]^{v}]$_{vv}$ = *va* **using** *forget-subst-v subst-v-v-def z fresh-prodN* **by** *metis*
moreover have *2*: Θ ; $\mathcal{B}$ ⊢$_{wf}$ (*x*, *b-of const*, [*va*]^{ce} == [*V-cons tid cons* [*x*]^{v}]^{ce} *AND c-of const x*) #$_{\Gamma}$ *GNil*
using *check-branch-s-branchI*(*17*)[*THEN check-s-wf*] ‹Γ = *GNil*› **by** *auto*
moreover hence Θ ; $\mathcal{B}$ ⊢$_{wf}$ (*x*, *b-of const*, [*va*]^{ce} == [*V-cons tid cons* [*x*]^{v}]^{ce}) #$_{\Gamma}$ *GNil*
using *wfG-conj* **by** *metis*
ultimately show *?thesis*
unfolding *subst-cv.simps subst-cev.simps subst-vv.simps* **by** *auto*
qed
show ‹*atom x* ♯ ([*va*]^{ce} == [*V-cons tid cons* [*z*]^{v}]^{ce})› **unfolding** *c.fresh ce.fresh v.fresh*
apply(*intro conjI*)
using *check-branch-s-branchI fresh-at-base z freshers* **apply** *simp*
using *check-branch-s-branchI fresh-at-base z freshers* **apply** *simp*
using *pure-supp* **apply** *force*
using *z fresh-x-neq fresh-prod5* **by** *metis*
qed
show ‹[*va*]^{ce} = [*V-cons tid cons* [*z*]^{v}]^{ce}[*z*::=*v*]$_{cev}$›
using ‹ *va* = *V-cons tid cons v*› *subst-cev.simps subst-vv.simps* **by** *auto*
show ‹ Θ ; $\mathcal{B}$; *GNil* ⊢ *v* ⇐ *const* › **using** *check-branch-s-branchI* **by** *auto*
show *supp* [*va*]^{ce} ⊆ *supp* $\mathcal{B}$ **using** ‹*supp va* = {}› *ce.supp* **by** *simp*
qed
show Θ ; $\mathcal{B}$; *GNil* ⊢ *v* ⇐ {‖ *z* : *b-of const* | *c-of const z* ‖}
using *check-branch-s-branchI z* **by** *auto*
qed

Main body of proof for this case

have Θ ; Φ ; $\mathcal{B}$; $(?G)[x::=v]_{\Gamma v}$; $\Delta[x::=v]_{\Delta v}$ $\vdash$ $s[x::=v]_{sv} \Leftarrow \tau[x::=v]_{\tau v}$
proof(*rule subst-check-check-s*)
show ‹Θ ; $\mathcal{B}$; *GNil* $\vdash$ $v \Leftarrow \{\!|\ z : b\text{-}of\ const\ |\ [\ va\ ]^{ce}\ ==\ [\ V\text{-}cons\ tid\ cons\ [\ z\ ]^{v}\ ]^{ce}\ \ AND\ c\text{-}of\ const\ z\ |\!\}$› **using** *vt* **by** *auto*
show ‹*atom z* ♯ (x, v)› **using** *z fresh-prodN* **by** *auto*
show ‹ Θ ; Φ ; $\mathcal{B}$; *?G* ; Δ $\vdash$ $s \Leftarrow \tau$›
using *check-branch-s-branchI* **by** *auto*

show ‹ *?G* = *GNil* @ $(x,$ *b-of const*, $([\ va\ ]^{ce}\ ==\ [\ V\text{-}cons\ tid\ cons\ [\ z\ ]^{v}\ ]^{ce}\ \ AND\ c\text{-}of\ const\ z)[z::=[\ x\ ]^{v}]_{cv})$ $\#_{\Gamma}$ *GNil*›
proof −
have $va[z::=[\ x\ ]^{v}]_{vv}$ = *va* **using** *forget-subst-v subst-v-v-def z fresh-prodN* **by** *metis*
moreover have $(c\text{-}of\ const\ z)[z::=[\ x\ ]^{v}]_{cv}$ = *c-of const x*
using *c-of-switch*[*of z const x*] *z fresh-prodN* **by** *metis*
ultimately show *?thesis*
unfolding *subst-cv.simps subst-cev.simps subst-vv.simps append-g.simps*
using *c-of-switch*[*of z const x*] *z fresh-prodN z fresh-prodN check-branch-s-branchI* **by** *argo*
qed
qed
moreover have $s[x::=v]_{sv}$ = $s'[x'::=v]_{sv}$
using *check-branch-s-branchI subst-v-flip-eq-two subst-v-s-def s-branch-s-branch-list.eq-iff* **by** *metis*
ultimately show *?case* **using** *check-branch-s-branchI* ‹$\tau[x::=v]_{\tau v}$ = τ› ‹$\Delta[x::=v]_{\Delta v}$ = Δ› **by** *auto*

qed(*auto+*)

Lemmas for while reduction. Making these separate lemmas allows flexibility in wiring them into the main proof and robustness if we change it

lemma *check-unit*:
fixes τ::τ **and** Φ::Φ **and** Δ::Δ **and** G::Γ
assumes Θ ; $\{||\}$; *GNil* $\vdash$ $\{\!|\ z : B\text{-}unit\ |\ TRUE\ |\!\} \lesssim \tau'$ **and** Θ ; $\{||\}$; *GNil* $\vdash_{wf}$ Δ **and** Θ $\vdash_{wf}$ Φ **and** Θ ; $\{||\}$ $\vdash_{wf}$ G
shows ‹ Θ ; Φ ; $\{||\}$; G ; Δ $\vdash$ $[[\ L\text{-}unit\ ]^{v}]^{s} \Leftarrow \tau'$›
proof −
have $*$:Θ ; $\{||\}$; *GNil* $\vdash$ $[L\text{-}unit]^{v} \Rightarrow \{\!|\ z : B\text{-}unit\ |\ [\ [\ z\ ]^{v}\ ]^{ce}\ ==\ [\ [\ L\text{-}unit\ ]^{v}\ ]^{ce}\ |\!\}$
using *infer-l.intros*(*4*) *infer-v-litI fresh-GNil assms wfX-wfY* **by** (*meson subtype-g-wf*)
moreover have Θ ; $\{||\}$; *GNil* $\vdash$ $\{\!|\ z : B\text{-}unit\ |\ [\ [\ z\ ]^{v}\ ]^{ce}\ ==\ [\ [\ L\text{-}unit\ ]^{v}\ ]^{ce}\ |\!\} \lesssim \tau'$
using *subtype-top subtype-trans* $*$ *infer-v-wf*
by (*meson assms*(*1*))
ultimately show *?thesis*
using *subtype-top subtype-trans fresh-GNil assms check-valI assms check-s-g-weakening assms setG.simps*

by (*metis bot.extremum infer-v-g-weakening subtype-weakening wfD-wf*)
qed

lemma *preservation-var*:
shows Θ ; Φ ; $\{||\}$; *GNil* ; Δ $\vdash$ *VAR* $u : \tau' = v$ *IN* $s \Leftarrow \tau \Longrightarrow \Theta \vdash \delta \sim \Delta \Longrightarrow$ *atom* u ♯ $\delta \Longrightarrow$ *atom* u ♯ $\Delta \Longrightarrow$
Θ ; Φ ; $\{||\}$; *GNil* ; $(u,\tau')\#_{\Delta}\Delta$ $\vdash$ $s \Leftarrow \tau \wedge \Theta \vdash (u,v)\#\delta \sim (u,\tau')\#_{\Delta}\Delta$
and
check-branch-s Θ Φ $\{||\}$ *GNil* Δ *tid dc const v cs* $\tau \Longrightarrow$ *True* **and**

check-branch-list Θ Φ $\{||\}$ Γ Δ *tid dclist v css* $\tau \Longrightarrow$ *True*
proof(*nominal-induct* $\{||\}$::*bv fset GNil* Δ *VAR u* : $\tau' = v$ *IN s* τ **and** τ **and** τ *rule*: *check-s-check-branch-s-check-branch-*
case (*check-varI u'* Θ Φ Δ τ *s'*)
hence Θ ; Φ ; $\{||\}$; *GNil* ; (*u*, τ') $\#_\Delta$ Δ $\vdash$ *s* $\Leftarrow$ τ **using** *check-s-abs-u check-v-wf* **by** *metis*

moreover have $\Theta \vdash ((u,v)\#\delta) \sim ((u,\tau')\#_\Delta\Delta)$ **proof**
show ‹Θ $\vdash$ $\delta \sim \Delta$ › **using** *check-varI* **by** *auto*
show ‹Θ ; $\{||\}$; *GNil* $\vdash$ *v* $\Leftarrow$ τ'› **using** *check-varI* **by** *auto*
show ‹*u* $\notin$ *fst* ' *set* δ› **using** *check-varI fresh-d-fst-d* **by** *auto*
qed

ultimately show *?case* **by** *simp*
qed(*auto+*)

lemma *check-while*:
shows Θ ; Φ ; $\{||\}$; *GNil* ; Δ $\vdash$ *WHILE s1 DO* { *s2* } $\Leftarrow$ $\tau \Longrightarrow$ *atom x* $\sharp$ (*s1*, *s2*) $\Longrightarrow$ *atom z'* $\sharp$ *x* $\Longrightarrow$
Θ ; Φ ; $\{||\}$; *GNil* ; Δ $\vdash$ *LET x* : ($\{\!|$ *z'* : *B-bool* | *TRUE* $|\!\}$) = *s1 IN* (*IF* (*V-var x*) *THEN* (*s2* ;; (*WHILE s1 DO* {*s2*}))
ELSE ([*V-lit L-unit*]^s)) $\Leftarrow$ τ **and**
check-branch-s Θ Φ $\{||\}$ *GNil* Δ *tid dc const v cs* $\tau \Longrightarrow$ *True* **and**
check-branch-list Θ Φ $\{||\}$ Γ Δ *tid dclist v css* $\tau \Longrightarrow$ *True*
proof(*nominal-induct* $\{||\}$::*bv fset GNil* Δ *AS-while s1 s2* τ **and** τ **and** τ *avoiding*: *s1 s2 x z' rule*: *check-s-check-branch-s-check-branch-list.strong-induct*)
case (*check-whileI* Θ Φ Δ *s1 z s2* τ')
have *teq*:$\{\!|$ *z'* : *B-bool* | *TRUE* $|\!\}$ = $\{\!|$ *z* : *B-bool* | *TRUE* $|\!\}$ **using** τ.*eq-iff* **by** *auto*

show *?case* **proof**
have *atom x* $\sharp$ τ' **using** *wfT-nil-supp fresh-def subtype-wfT check-whileI*(*5*) **by** *fast*
moreover have *atom x* $\sharp$ $\{\!|$ *z'* : *B-bool* | *TRUE* $|\!\}$ **using** τ.*fresh c.fresh b.fresh* **by** *metis*
ultimately show ‹*atom x* $\sharp$ (Θ, Φ, $\{||\}$, *GNil*, Δ, $\{\!|$ *z'* : *B-bool* | *TRUE* $|\!\}$, *s1*, τ')›
apply(*unfold fresh-prodN*)
using *check-whileI wb-x-fresh check-s-wf wfD-x-fresh fresh-empty-fset fresh-GNil fresh-Pair* ‹*atom x* $\sharp$ τ'› **by** *metis*

show ‹ Θ ; Φ ; $\{||\}$; *GNil* ; Δ $\vdash$ *s1* $\Leftarrow$ $\{\!|$ *z'* : *B-bool* | *TRUE* $|\!\}$› **using** *check-whileI teq* **by** *metis*

let *?G* = (*x*, *b-of* $\{\!|$ *z'* : *B-bool* | *TRUE* $|\!\}$, *c-of* $\{\!|$ *z'* : *B-bool* | *TRUE* $|\!\}$ *x*) $\#_\Gamma$ *GNil*

have *cof*:(*c-of* $\{\!|$ *z'* : *B-bool* | *TRUE* $|\!\}$ *x*) = *C-true* **using** *c-of.simps check-whileI subst-cv.simps* **by** *metis*
have *wfg*: Θ ; $\{||\}$ $\vdash_{wf}$ *?G* **proof**
show *c-of* $\{\!|$ *z'* : *B-bool* | *TRUE* $|\!\}$ *x* $\in$ {*TRUE*, *FALSE*} **using** *subst-cv.simps cof* **by** *auto*
show Θ ; $\{||\}$ $\vdash_{wf}$ *GNil* **using** *wfG-nilI check-whileI wfX-wfY check-s-wf* **by** *metis*
show *atom x* $\sharp$ *GNil* **using** *fresh-GNil* **by** *auto*
show Θ ; $\{||\}$ $\vdash_{wf}$ *b-of* $\{\!|$ *z'* : *B-bool* | *TRUE* $|\!\}$ **using** *wfB-boolI wfX-wfY check-s-wf b-of.simps*
by (*metis* ‹Θ ; $\{||\}$ $\vdash_{wf}$ *GNil*›)
qed

obtain *zz*::*x* **where** *zf*:‹*atom zz* $\sharp$ ((Θ, Φ, $\{||\}$::*bv fset*, *?G* , Δ, [*x*]^v,
AS-seq s2 (*AS-while s1 s2*), *AS-val* [*L-unit*]^v, τ'),*x*,*?G*)›
using *obtain-fresh* **by** *blast*

show ‹ Θ ; Φ ; $\{||\}$; *?G* ; Δ $\vdash$
AS-if [*x*]^v (*AS-seq s2* (*AS-while s1 s2*)) (*AS-val* [*L-unit*]^v) $\Leftarrow$ τ'›
proof
show *atom zz* $\sharp$ (Θ, Φ, $\{||\}$::*bv fset*, *?G* , Δ, [*x*]^v, *AS-seq s2* (*AS-while s1 s2*), *AS-val* [*L-unit*]^v, τ') **using** *zf* **by** *auto*
show ‹Θ ; $\{||\}$; *?G* $\vdash$ [*x*]^v $\Leftarrow$ $\{\!|$ *zz* : *B-bool* | *TRUE* $|\!\}$› **proof**
have *atom zz* $\sharp$ *x* $\wedge$ *atom zz* $\sharp$ *?G* **using** *zf fresh-prodN* **by** *metis*
thus ‹ Θ ; $\{||\}$; *?G* $\vdash$ [*x*]^v $\Rightarrow$$\{\!|$ *zz* : *B-bool* | [[*zz*]^v]^{ce} == [[*x*]^v]^{ce} $|\!\}$›
using *infer-v-varI lookup.simps wfg b-of.simps* **by** *metis*
thus ‹Θ ; $\{||\}$; *?G* $\vdash$ $\{\!|$ *zz* : *B-bool* | [[*zz*]^v]^{ce} == [[*x*]^v]^{ce} $|\!\}$ $\lesssim$ $\{\!|$ *zz* : *B-bool* | *TRUE* $|\!\}$›
using *subtype-top infer-v-wf* **by** *metis*
qed
show ‹ Θ ; Φ ; $\{||\}$; *?G* ; Δ $\vdash$ *AS-seq s2* (*AS-while s1 s2*) $\Leftarrow$ $\{\!|$ *zz* : *b-of* τ' | [[*x*]^v]^{ce} == [[*L-true*]^v]^{ce} *IMP* *c-of* τ' *zz* $|\!\}$›
proof
have $\{\!|$ *zz* : *B-unit* | *TRUE* $|\!\}$ = $\{\!|$ *z* : *B-unit* | *TRUE* $|\!\}$ **using** τ*.eq-iff* **by** *auto*
thus ‹ Θ ; Φ ; $\{||\}$; *?G* ; Δ $\vdash$ *s2* $\Leftarrow$ $\{\!|$ *zz* : *B-unit* | *TRUE* $|\!\}$› **using** *check-s-g-weakening*(*1*) [*OF check-whileI*(*3*) *- wfg*] *setG.simps*
by (*simp add*: ‹$\{\!|$ *zz* : *B-unit* | *TRUE* $|\!\}$ = $\{\!|$ *z* : *B-unit* | *TRUE* $|\!\}$›)

show ‹ Θ ; Φ ; $\{||\}$; *?G* ; Δ $\vdash$ *AS-while s1 s2* $\Leftarrow$ $\{\!|$ *zz* : *b-of* τ' | [[*x*]^v]^{ce} == [[*L-true*]^v]^{ce} *IMP* *c-of* τ' *zz* $|\!\}$›
proof(*rule check-s-supertype*(*1*))
have ‹ Θ ; Φ ; $\{||\}$; *GNil* ; Δ $\vdash$ *AS-while s1 s2* $\Leftarrow$ τ'› **using** *check-whileI* **by** *auto*
thus $*$:‹ Θ ; Φ ; $\{||\}$; *?G* ; Δ $\vdash$ *AS-while s1 s2* $\Leftarrow$ τ' › **using** *check-s-g-weakening*(*1*)[*OF - - wfg*] *setG.simps* **by** *auto*

show ‹Θ ; $\{||\}$; *?G* $\vdash$ τ' $\lesssim$ $\{\!|$ *zz* : *b-of* τ' | [[*x*]^v]^{ce} == [[*L-true*]^v]^{ce} *IMP* *c-of* τ' *zz* $|\!\}$›
proof(*rule subtype-eq-if-*τ)
show ‹ Θ ; $\{||\}$; *?G* $\vdash_{wf}$ τ' › **using** $*$ *check-s-wf* **by** *auto*
thm *wfT-wfT-if-rev*
show ‹ Θ ; $\{||\}$; *?G* $\vdash_{wf}$ $\{\!|$ *zz* : *b-of* τ' | [[*x*]^v]^{ce} == [[*L-true*]^v]^{ce} *IMP* *c-of* τ' *zz* $|\!\}$ ›
apply(*rule wfT-eq-imp*, *simp add*: *base-for-lit.simps*)
using *subtype-wf check-whileI wfg zf fresh-prodN* **by** *metis*+
show ‹*atom zz* $\sharp$ τ'› **using** *zf fresh-prodN* **by** *metis*
qed
qed

qed
show ‹ Θ ; Φ ; $\{||\}$; *?G* ; Δ $\vdash$ *AS-val* [*L-unit*]^v $\Leftarrow$ $\{\!|$ *zz* : *b-of* τ' | [[*x*]^v]^{ce} == [[*L-false*]^v]^{ce} *IMP* *c-of* τ' *zz* $|\!\}$›
proof(*rule check-s-supertype*(*1*))

show $*$:‹ Θ ; Φ ; $\{||\}$; *?G* ; Δ $\vdash$ *AS-val* [*L-unit*]^v $\Leftarrow$ τ'›
using *check-unit*[*OF check-whileI*(*5*) *- - wfg*] **using** *check-whileI wfg wfX-wfY check-s-wf* **by** *metis*
show ‹Θ ; $\{||\}$; *?G* $\vdash$ τ' $\lesssim$ $\{\!|$ *zz* : *b-of* τ' | [[*x*]^v]^{ce} == [[*L-false*]^v]^{ce} *IMP* *c-of* τ' *zz* $|\!\}$›
proof(*rule subtype-eq-if-*τ)

```
          show ‹ Θ ; {||} ; ?G ⊢wf τ' › using ∗ check-s-wf by metis
          show ‹ Θ ; {||} ; ?G ⊢wf {‖ zz : b-of τ' | [ [ x ]v ]ce == [ [ L-false ]v ]ce IMP c-of τ' zz ‖} ›
            apply(rule wfT-eq-imp, simp add: base-for-lit.simps)
         using subtype-wf check-whileI wfg zf fresh-prodN by metis+
          show ‹atom zz ♯ τ'› using zf fresh-prodN by metis
         qed
      qed
     qed
  qed
qed(auto+)

thm split.intros


lemma check-s-narrow:
  fixes s::s and x::x
  assumes atom x ♯ (Θ, Φ, B, Γ, Δ, c, τ, s) and Θ ; Φ ; B ; (x, B-bool, c) #Γ Γ ; Δ ⊢ s ⇐ τ and
    Θ ; B ; Γ ⊨ c
  shows Θ ; Φ ; B ; Γ ; Δ ⊢ s ⇐ τ
proof −
  let ?B = ({||}::bv fset)
  let ?v = V-lit L-true


  obtain z::x where z: atom z ♯ (x, [ L-true ]v,c) using obtain-fresh by metis

  have atom z ♯ c using z fresh-prodN by auto
  hence c: c[x::=[ z ]v]v[z::=[ x ]v]cv = c using subst-v-c-def by simp

  have Θ ; Φ ; B ; ((x,B-bool, c) #Γ Γ)[x::=?v]Γv ; Δ[x::=?v]Δv ⊢ s[x::=?v]sv ⇐ τ[x::=?v]τv
proof(rule subst-infer-check-s(1) )
    show vt: ‹ Θ ; B ; Γ ⊢ [ L-true ]v ⇒ {‖ z : B-bool | (CE-val (V-var z)) == (CE-val (V-lit L-true )) ‖} ›
      using infer-v-litI check-s-wf wfG-elims(2) infer-trueI assms by metis
    show ‹Θ ; B ; Γ ⊢ {‖ z : B-bool | (CE-val (V-var z)) == (CE-val (V-lit L-true )) ‖} ≲ {‖ z : B-bool
| c[x::=[ z ]v]v ‖}› proof
      show ‹atom x ♯ (Θ, B, Γ, z, [ [ z ]v ]ce == [ [ L-true ]v ]ce , z, c[x::=[ z ]v]v)›
        apply(unfold fresh-prodN, intro conjI)
        prefer 5
        using c.fresh ce.fresh v.fresh z fresh-prodN apply auto[1]
            prefer 6
        using fresh-subst-v-if[of atom x c x] assms fresh-prodN apply simp
        using z assms fresh-prodN apply metis
        using z assms fresh-prodN apply metis
        using z assms fresh-prodN apply metis
 using z fresh-prodN assms fresh-at-base by metis+
      show ‹ Θ ; B ; Γ ⊢wf {‖ z : B-bool | [ [ z ]v ]ce == [ [ L-true ]v ]ce ‖} › using vt infer-v-wf by
simp
      show ‹ Θ ; B ; Γ ⊢wf {‖ z : B-bool | c[x::=[ z ]v]v ‖} › proof(rule wfG-wfT)
        show ‹ Θ ; B ⊢wf (x, B-bool, c[x::=[ z ]v]v[z::=[ x ]v]cv) #Γ Γ › using c check-s-wf assms by
metis
```

have *atom x* ♯ [*z*]^v **using** *v.fresh z fresh-at-base* **by** *auto*
thus ‹*atom x* ♯ *c*[*x*::=[*z*]^v]$_v$› **using** *fresh-subst-v-if*[*of atom x c*] **by** *auto*
qed
have *wfg*: Θ ; $\mathcal{B}$ ⊢$_{wf}$(*x*, *B-bool*, ([[*z*]^v]^{ce} == [[*L-true*]^v]^{ce})[*z*::=[*x*]^v]$_v$) #$_\Gamma$ Γ
using *wfT-wfG vt infer-v-wf fresh-prodN assms* **by** *simp*
show ‹Θ ; $\mathcal{B}$; (*x*, *B-bool*, ([[*z*]^v]^{ce} == [[*L-true*]^v]^{ce})[*z*::=[*x*]^v]$_v$) #$_\Gamma$ Γ ⊨ *c*[*x*::=[*z*]^v]$_v$[*z*::=[*x*]^v]$_v$ ›
using *c valid-weakening*[*OF assms*(3) - *wfg*] *setG.simps*
using *subst-v-c-def* **by** *auto*
qed
show ‹*atom z* ♯ (*x*, [*L-true*]^v)› **using** *z fresh-prodN* **by** *metis*
show ‹ Θ ; Φ ; $\mathcal{B}$; (*x*, *B-bool*, *c*) #$_\Gamma$ Γ ; Δ ⊢ *s* ⇐ τ› **using** *assms* **by** *auto*

thus ‹(*x*, *B-bool*, *c*) #$_\Gamma$ Γ = *GNil* @ (*x*, *B-bool*, *c*[*x*::=[*z*]^v]$_v$[*z*::=[*x*]^v]$_{cv}$) #$_\Gamma$ Γ› **using** *append-g.simps c* **by** *auto*
qed

moreover have ((*x*,*B-bool*, *c*) #$_\Gamma$ Γ)[*x*::=*?v*]$_{\Gamma v}$ = Γ **using** *subst-gv.simps* **by** *auto*
ultimately show *?thesis* **using** *assms forget-subst-dv forget-subst-sv forget-subst-tv fresh-prodN* **by** *metis*
qed

lemma *check-assert-s*:
fixes *s*::*s* **and** *x*::*x*
assumes Θ ; Φ ; {||} ; *GNil* ; Δ ⊢ *AS-assert c s* ⇐ τ
shows Θ ; Φ ; {||} ; *GNil* ; Δ ⊢ *s* ⇐ τ ∧ Θ ; {||} ; *GNil* ⊨ *c*
proof −
let *?B* = ({||}::*bv fset*)
let *?v* = *V-lit L-true*

obtain *x* **where** *x*: Θ ; Φ ; *?B* ; (*x*,*B-bool*, *c*) #$_\Gamma$ *GNil* ; Δ ⊢ *s* ⇐ τ ∧ *atom x* ♯ (Θ, Φ, *?B*, *GNil*, Δ, *c*, τ, *s*) ∧ Θ ; *?B* ; *GNil* ⊨ *c*
using *check-s-elims*(10)[*OF* ‹Θ ; Φ ; *?B* ; *GNil* ; Δ ⊢ *AS-assert c s* ⇐ τ›] *valid.simps* **by** *metis*

show *?thesis* **using** *assms check-s-narrow x* **by** *metis*
qed

lemma *preservation*:
fixes *s*::*s* **and** *s'*::*s*
assumes Φ ⊢ ⟨ δ , *s* ⟩ ⟶ ⟨ δ' , *s'* ⟩ **and** Θ ; Φ ; Δ ⊢ ⟨ δ , *s* ⟩ ⇐ τ
shows ∃Δ'. Θ ; Φ ; Δ' ⊢ ⟨ δ' , *s'* ⟩ ⇐ τ ∧ *setD* Δ ⊆ *setD* Δ'
using *assms*
proof(*induct arbitrary*: τ *rule*: *reduce-stmt.induct*)
case (*reduce-let-plusI* δ *x n1 n2 s'*)
then show *?case* **using** *preservation-plus*
by (*metis order-refl*)
next
case (*reduce-let-leqI b n1 n2* δ *x s*)
then show *?case* **using** *preservation-leq* **by** (*metis order-refl*)
next
case (*reduce-let-appI f z b c* τ' *s'* Φ δ *x v s*)

hence *tt*: Θ ; Φ ; $\{||\}$; *GNil* ; Δ $\vdash$ *AS-let x* (*AE-app f v*) *s* $\Leftarrow$ τ $\wedge$ $\Theta \vdash \delta \sim \Delta$ $\wedge$ ($\forall$ *fd*$\in$*set* Φ. *check-fundef* Θ Φ *fd*) **using** *config-type-elims*[*OF reduce-let-appI*(*2*)] **by** *metis*
hence $*$:Θ ; Φ ; $\{||\}$; *GNil* ; Δ $\vdash$ *AS-let x* (*AE-app f v*) *s* $\Leftarrow$ τ **by** *auto*

hence Θ ; Φ ; $\{||\}$; *GNil* ; Δ $\vdash$ *AS-let2 x* $(\tau'[z::=v]_{\tau v})$ $(s'[z::=v]_{sv})$ *s* $\Leftarrow$ τ **using** *preservation-app reduce-let-appI tt* **by** *auto*

hence Θ ; Φ ; $\Delta \vdash \langle$ δ , *AS-let2 x* $(\tau'[z::=v]_{\tau v})$ $s'[z::=v]_{sv}$ *s* $\rangle \Leftarrow \tau$ **using** *config-typeI tt* **by** *auto*
then show *?case* **using** *tt order-refl reduce-let-appI* **by** *metis*

next
case (*reduce-let-appPI f bv z b c* τ' *s'* Φ δ *x b' v s*)

hence *tt*: Θ ; Φ ; $\{||\}$; *GNil* ; Δ $\vdash$ *AS-let x* (*AE-appP f b' v*) *s* $\Leftarrow$ τ $\wedge$ $\Theta \vdash \delta \sim \Delta$ $\wedge$ ($\forall$ *fd*$\in$*set* Φ. *check-fundef* Θ Φ *fd*) **using** *config-type-elims*[*OF reduce-let-appPI*(*2*)] **by** *metis*
hence $*$:Θ ; Φ ; $\{||\}$; *GNil* ; Δ $\vdash$ *AS-let x* (*AE-appP f b' v*) *s* $\Leftarrow$ τ **by** *auto*

have Θ ; Φ ; $\{||\}$; *GNil* ; Δ $\vdash$ *AS-let2 x* $(\tau'[bv::=b']_{\tau b}[z::=v]_{\tau v})$ $(s'[bv::=b']_{sb}[z::=v]_{sv})$ *s* $\Leftarrow$ τ
proof(*rule preservation-poly-app*)
show ‹*Some* (*AF-fundef f* (*AF-fun-typ-some bv* (*AF-fun-typ z b c* τ' *s'*))) = *lookup-fun* Φ *f*› **using** *reduce-let-appPI* **by** *metis*
show ‹$\forall$ *fd*$\in$*set* Φ. *check-fundef* Θ Φ *fd*› **using** *tt lookup-fun-member* **by** *metis*
show ‹ Θ ; Φ ;$\{||\}$; *GNil* ; Δ $\vdash$ *AS-let x* (*AE-appP f b' v*) *s* $\Leftarrow$ τ› **using** $*$ **by** *auto*
show ‹ Θ ; $\{||\}$ $\vdash_{wf}$ *b'* › **using** *check-s-elims infer-e-wf wfE-elims* $*$ **by** *metis*
qed(*auto*+)

hence Θ ; Φ ; $\Delta \vdash \langle$ δ , *AS-let2 x* $(\tau'[bv::=b']_{\tau b}[z::=v]_{\tau v})$ $s'[bv::=b']_{sb}[z::=v]_{sv}$ *s* $\rangle \Leftarrow \tau$ **using** *config-typeI tt* **by** *auto*
then show *?case* **using** *tt order-refl reduce-let-appI* **by** *metis*

next
case (*reduce-if-trueI* δ *s1 s2*)
then show *?case* **using** *preservation-if* **by** *metis*
next
case (*reduce-if-falseI uw* δ *s1 s2*)
then show *?case* **using** *preservation-if* **by** *metis*
next
case (*reduce-let-valI* δ *x v s*)
then show *?case* **using** *preservation-let-val* **by** *presburger*
next
case (*reduce-let-mvar u v* δ Φ *x s*)
hence $*$:Θ ; Φ ; $\{||\}$; *GNil* ; Δ $\vdash$ *AS-let x* (*AE-mvar u*) *s* $\Leftarrow$ τ $\wedge$ $\Theta \vdash \delta \sim \Delta$ $\wedge$ ($\forall$ *fd*$\in$*set* Φ. *check-fundef* Θ Φ *fd*)
using *config-type-elims* **by** *blast*

hence $**$: Θ ; Φ ; $\{||\}$; *GNil* ; Δ $\vdash$ *AS-let x* (*AE-mvar u*) *s* $\Leftarrow$ τ **by** *auto*
obtain *xa*::*x* **and** *za*::*x* **and** *ca*::*c* **and** *ba*::*b* **and** *sa*::*s* **where**
sa1: *atom xa* $\sharp$ (Θ, Φ, $\{||\}$::*bv fset*, *GNil*, Δ, *AE-mvar u*, τ) $\wedge$ *atom za* $\sharp$ (*xa*, Θ, Φ, $\{||\}$::*bv fset*, *GNil*, Δ, *AE-mvar u*, τ, *sa*) $\wedge$
Θ ; Φ ; $\{||\}$; *GNil* ; Δ $\vdash$ *AE-mvar u* $\Rightarrow$ $\{\!|$ *za* : *ba* | *ca* $|\!\}$ $\wedge$
Θ ; Φ ; $\{||\}$; (*xa*, *ba*, *ca*[*za*::=*V-var xa*]$_{cv}$) $\#_\Gamma$ *GNil* ; Δ $\vdash$ *sa* $\Leftarrow$ τ $\wedge$
($\forall$ *c*. *atom c* $\sharp$ (*s*, *sa*) $\longrightarrow$ *atom c* $\sharp$ (*x*, *xa*, *s*, *sa*) $\longrightarrow$ $(x \leftrightarrow c) \cdot s = (xa \leftrightarrow c) \cdot sa$)

using *check-s-elims(2)*[*OF* ∗∗] *subst-defs* **by** *metis*

have Θ ; {||} ; *GNil* ⊢ *v* ⇐ {‖ *za* : *ba* | *ca* ‖} **proof** −
have (*u* , {‖ *za* : *ba* | *ca* ‖}) ∈ *setD* Δ **using** *infer-e-elims(11) sa1* **by** *fast*
thus *?thesis* **using** *delta-sim-v reduce-let-mvar config-type-elims check-s-wf* **by** *metis*
qed

then obtain τ′ **where** *vst*: Θ ; {||} ; *GNil* ⊢ *v* ⇒ τ′ ∧
Θ ; {||} ; *GNil* ⊢ τ′ ≲ {‖ *za* : *ba* | *ca* ‖} **using** *check-v-elims* **by** *blast*

obtain *za2* **and** *ba2* **and** *ca2* **where** *zbc*: τ′ = ({‖ *za2* : *ba2* | *ca2* ‖}) ∧ *atom za2* ♯ (*xa*, (*xa*, Θ, Φ, {||}::*bv fset*, *GNil*, Δ, *AE-val v*, τ, *sa*))
using *obtain-fresh-z* **by** *blast*
have *beq*: *ba*=*ba2* **using** *subtype-eq-base vst zbc* **by** *blast*

moreover have *xaf*: *atom xa* ♯ (*za*, *za2*)
apply(*unfold fresh-prodN*, *intro conjI*)
using *sa1 zbc fresh-prodN fresh-x-neq* **by** *metis+*

have *sat2*: Θ ; Φ ; {||} ; *GNil*@(*xa*, *ba*, *ca2*[*za2*::=*V-var xa*]$_{cv}$) #$_\Gamma$ *GNil* ; Δ ⊢ *sa* ⇐ τ **proof**(*rule ctx-subtype-s*)
show Θ ; Φ ; {||} ; *GNil* @ (*xa*, *ba*, *ca*[*za*::=*V-var xa*]$_{cv}$) #$_\Gamma$ *GNil* ; Δ ⊢ *sa* ⇐ τ **using** *sa1* **by** *auto*
show Θ ; {||} ; *GNil* ⊢ {‖ *za2* : *ba* | *ca2* ‖} ≲ {‖ *za* : *ba* | *ca* ‖} **using** *beq zbc vst* **by** *fast*
show *atom xa* ♯ (*za*, *za2*, *ca*, *ca2*) **proof** −
have ∗:Θ ; {||} ; *GNil* ⊢$_{wf}$ ({‖ *za2* : *ba2* | *ca2* ‖}) **using** *zbc vst subtype-wf* **by** *auto*
hence *supp ca2* ⊆ { *atom za2* } **using** *wfT-supp-c*[*OF* ∗] *supp-GNil* **by** *simp*
moreover have *atom za2* ♯ *xa* **using** *zbc fresh-Pair fresh-x-neq* **by** *metis*
ultimately have *atom xa* ♯ *ca2* **using** *zbc supp-at-base fresh-def*
by (*metis empty-iff singleton-iff subset-singletonD*)
moreover have *atom xa* ♯ *ca* **proof** −
have ∗:Θ ; {||} ; *GNil* ⊢$_{wf}$ ({‖ *za* : *ba* | *ca* ‖}) **using** *zbc vst subtype-wf* **by** *auto*
hence *supp ca* ⊆ { *atom za* } **using** *wfT-supp* τ.*supp* **by** *force*
moreover have *xa* ≠ *za* **using** *fresh-def fresh-x-neq xaf fresh-Pair* **by** *metis*
ultimately show *?thesis* **using** *fresh-def* **by** *auto*
qed
ultimately show *?thesis* **using** *xaf sa1 fresh-prod4 fresh-Pair* **by** *metis*
qed
qed
hence *dwf*: Θ ; {||} ; *GNil* ⊢$_{wf}$ Δ **using** *sa1 infer-e-wf* **by** *meson*

have Θ ; Φ ; {||} ; *GNil* ; Δ ⊢ *AS-let xa* (*AE-val v*) *sa* ⇐ τ **proof**
have *atom xa* ♯ (*AE-val v*) **using** *infer-v-wf(1) wfV-supp fresh-def e.fresh x-not-in-b-set vst* **by** *fastforce*
thus *atom xa* ♯ (Θ, Φ, {||}::*bv fset*, *GNil*, Δ, *AE-val v*, τ) **using** *sa1 freshers* **by** *simp*
have *atom za2* ♯ (*AE-val v*) **using** *infer-v-wf(1) wfV-supp fresh-def e.fresh x-not-in-b-set vst* **by** *fastforce*
thus *atom za2* ♯ (*xa*, Θ, Φ, {||}::*bv fset*, *GNil*, Δ, *AE-val v*, τ, *sa*) **using** *zbc freshers fresh-prodN* **by** *auto*
have Θ ⊢$_{wf}$ Φ **using** *sa1 infer-e-wf* **by** *auto*
thus Θ ; Φ ; {||} ; *GNil* ; Δ ⊢ *AE-val v* ⇒ {‖ *za2* : *ba* | *ca2* ‖}
using *zbc vst beq dwf infer-e-valI* **by** *blast*

show Θ ; Φ ; $\{||\}$; $(xa,\ ba,\ ca2[za2::=V\text{-}var\ xa]_v)$ $\#_\Gamma$ *GNil* ; Δ $\vdash$ $sa \Leftarrow \tau$ **using** *sat2 append-g.simps subst-defs* **by** *metis*
qed
moreover have *AS-let xa* (*AE-val v*) *sa* = *AS-let x* (*AE-val v*) *s* **proof** −
have $[[atom\ x]]lst.\ s = [[atom\ xa]]lst.\ sa$
using *sa1 Abs1-eq-iff-all*(*3*)[**where** *z*= (*s*, *sa*)] **by** *metis*
thus *?thesis* **using** *s-branch-s-branch-list.eq-iff*(*2*) **by** *metis*
qed
ultimately have Θ ; Φ ; $\{||\}$; *GNil* ; Δ $\vdash$ *AS-let x* (*AE-val v*) *s* $\Leftarrow \tau$ **by** *auto*

then show *?case* **using** *reduce-let-mvar* ∗ *config-typeI*
by (*meson order-refl*)
next
case (*reduce-let2I* Φ δ *s1* δ' *s1'* *x t s2*)
hence ∗∗: Θ ; Φ ; $\{||\}$; *GNil* ; Δ $\vdash$ *AS-let2 x t s1 s2* $\Leftarrow \tau$ $\wedge$ $\Theta \vdash \delta \sim \Delta \wedge (\forall fd{\in}set\ \Phi.$ *check-fundef* Θ Φ *fd*) **using** *config-type-elims*[*OF reduce-let2I*(*3*)] **by** *blast*
hence ∗:Θ ; Φ ; $\{||\}$; *GNil* ; Δ $\vdash$ *AS-let2 x t s1 s2* $\Leftarrow \tau$ **by** *auto*

obtain *xa*::*x* **and** *z*::*x* **and** *c* **and** *b* **and** *s2a*::*s* **where** *st*: *atom xa* ♯ (Θ, Φ, $\{||\}$::*bv fset*, *GNil*, Δ, *t*, *s1*, τ) $\wedge$
Θ ; Φ ; $\{||\}$; *GNil* ; Δ $\vdash$ *s1* $\Leftarrow$ *t* $\wedge$
Θ ; Φ ; $\{||\}$; (*xa*, *b-of t*, *c-of t xa*) $\#_\Gamma$ *GNil* ; Δ $\vdash$ *s2a* $\Leftarrow \tau$ $\wedge$ ($[[atom\ x]]lst.\ s2 = [[atom\ xa]]lst.$ *s2a*)
using *check-s-elims*(*4*)[*OF* ∗] *Abs1-eq-iff-all*(*3*) **by** *metis*

hence Θ ; Φ ; $\Delta \vdash \langle\ \delta\ ,\ s1\ \rangle \Leftarrow t$ **using** *config-typeI* ∗∗ **by** *auto*
then obtain Δ' **where** *s1r*: Θ ; Φ ; $\Delta' \vdash \langle\ \delta'\ ,\ s1'\ \rangle \Leftarrow t \wedge$ *setD* $\Delta \subseteq$ *setD* Δ' **using** *reduce-let2I* **by** *presburger*

have Θ ; Φ ; $\{||\}$; *GNil* ; Δ' $\vdash$ *AS-let2 xa t s1' s2a* $\Leftarrow \tau$
proof(*rule check-let2I*)
show ∗:Θ ; Φ ; $\{||\}$; *GNil* ; Δ' $\vdash$ *s1'* $\Leftarrow$ *t* **using** *config-type-elims st s1r* **by** *metis*
show *atom xa* ♯ (Θ, Φ, $\{||\}$::*bv fset*, *GNil*, Δ',*t*, *s1'*, τ) **proof** −
have *atom xa* ♯ *s1'* **using** *check-s-x-fresh* ∗ **by** *auto*
moreover have *atom xa* ♯ Δ' **using** *check-s-x-fresh* ∗ **by** *auto*
ultimately show *?thesis* **using** *st fresh-prodN* **by** *metis*
qed

show Θ ; Φ ; $\{||\}$; (*xa*, *b-of t*, *c-of t xa*) $\#_\Gamma$ *GNil* ; Δ' $\vdash$ *s2a* $\Leftarrow \tau$ **proof** −
have Θ ; $\{||\}$; *GNil* $\vdash_{wf}$ Δ' **using** ∗ *check-s-wf* **by** *auto*
moreover have Θ ; $\{||\} \vdash_{wf}$ ((*xa*, *b-of t*, *c-of t xa*) $\#_\Gamma$ *GNil*) **using** *st check-s-wf* **by** *auto*
ultimately have Θ ; $\{||\}$; ((*xa*, *b-of t* , *c-of t xa*) $\#_\Gamma$ *GNil*) $\vdash_{wf}$ Δ' **using** *wf-weakening* **by** *auto*
thus *?thesis* **using** *check-s-d-weakening check-s-wf st s1r* **by** *metis*
qed
qed
moreover have *AS-let2 xa t s1' s2a* = *AS-let2 x t s1' s2* **using** *st s-branch-s-branch-list.eq-iff* **by** *metis*
ultimately have Θ ; Φ ; $\{||\}$; *GNil* ; Δ' $\vdash$ *AS-let2 x t s1' s2* $\Leftarrow \tau$ **using** *st* **by** *argo*
moreover have $\Theta \vdash \delta' \sim \Delta'$ **using** *config-type-elims s1r* **by** *fast*
ultimately show *?case* **using** *config-typeI* ∗∗
by (*meson s1r*)
next

case (*reduce-let2-valI vb δ x t v s*)
then show *?case* **using** *preservation-let-val* **by** *meson*
next
case (*reduce-varI u δ Φ τ′ v s*)
thm *check-s-flip-u*
hence ∗∗ : Θ ; Φ ; {||} ; *GNil* ; Δ ⊢ *AS-var u τ′ v s* ⇐ τ ∧ Θ ⊢ δ ∼ Δ ∧ (∀ *fd*∈*set* Φ. *check-fundef* Θ Φ *fd*)
using *config-type-elims* **by** *meson*
have *uf*: *atom u* ♯ Δ **using** *reduce-varI delta-sim-fresh* **by** *force*
hence ∗: Θ ; Φ ; {||} ; *GNil* ; Δ ⊢ *AS-var u τ′ v s* ⇐ τ **and** Θ ⊢ δ ∼ Δ **using** ∗∗ **by** *auto*

thus *?case* **using** *preservation-var reduce-varI config-typeI* ∗∗ *set-subset-Cons setD-ConsD subsetI* **by** (*metis delta-sim-fresh*)

next
case (*reduce-assignI Φ δ u v*)
hence ∗: Θ ; Φ ; {||} ; *GNil* ; Δ ⊢ *AS-assign u v* ⇐ τ ∧ Θ ⊢ δ ∼ Δ ∧ (∀ *fd*∈*set* Φ. *check-fundef* Θ Φ *fd*)
using *config-type-elims* **by** *meson*
then obtain *z* **and** τ′ **where** *zt*: Θ ; {||} ; *GNil* ⊢ ({‖ *z* : *B-unit* | *TRUE* ‖}) ≲ τ ∧ (*u*,τ′) ∈ *setD* Δ ∧ Θ ; {||} ; *GNil* ⊢ *v* ⇐ τ′ ∧ Θ ; {||} ; *GNil* ⊢$_{wf}$ Δ
using *check-s-elims*(*8*) **by** *metis*
hence Θ ⊢ *update-d δ u v* ∼ Δ **using** *update-d-sim* ∗ **by** *metis*
moreover have Θ ; Φ ; {||} ; *GNil* ; Δ ⊢ *AS-val* (*V-lit L-unit*) ⇐ τ **using** *zt* ∗ *check-s-v-unit check-s-wf*
by *auto*
ultimately show *?case* **using** *config-typeI* ∗ **by** (*meson order-refl*)
next
case (*reduce-seq1I Φ δ s*)
hence Θ ; Φ ; {||} ; *GNil* ; Δ ⊢ *s* ⇐ τ ∧ Θ ⊢ δ ∼ Δ ∧ (∀ *fd*∈*set* Φ. *check-fundef* Θ Φ *fd*)
using *check-s-elims config-type-elims* **by** *force*
then show *?case* **using** *config-typeI* **by** *blast*
next
case (*reduce-seq2I s1 v Φ δ δ′ s1′ s2*)
hence *tt*: Θ ; Φ ; {||} ; *GNil* ; Δ ⊢ *AS-seq s1 s2* ⇐ τ ∧ Θ ⊢ δ ∼ Δ ∧ (∀ *fd*∈*set* Φ. *check-fundef* Θ Φ *fd*)
using *config-type-elims* **by** *blast*
then obtain *z* **where** *zz*: Θ ; Φ ; {||} ; *GNil*; Δ ⊢ *s1* ⇐ ({‖ *z* : *B-unit* | *TRUE* ‖}) ∧ Θ ; Φ ; {||} ; *GNil* ; Δ ⊢ *s2* ⇐ τ
using *check-s-elims* **by** *blast*
hence Θ ; Φ ; Δ ⊢ ⟨ δ , *s1* ⟩ ⇐ ({‖ *z* : *B-unit* | *TRUE* ‖})
using *tt config-typeI tt* **by** *simp*
then obtain Δ′ **where** ∗: Θ ; Φ ; Δ′ ⊢ ⟨ δ′ , *s1′* ⟩ ⇐ ({‖ *z* : *B-unit* | *TRUE* ‖}) ∧ *setD* Δ ⊆ *setD* Δ′
using *reduce-seq2I* **by** *meson*
moreover hence *s′t*: Θ ; Φ ; {||} ; *GNil* ; Δ′ ⊢ *s1′* ⇐ ({‖ *z* : *B-unit* | *TRUE* ‖}) ∧ Θ ⊢ δ′ ∼ Δ′
using *config-type-elims* **by** *force*
moreover hence Θ ; {||} ; *GNil* ⊢$_{wf}$ Δ′ **using** *check-s-wf* **by** *meson*
moreover hence Θ ; Φ ; {||} ; *GNil* ; Δ′ ⊢ *s2* ⇐ τ
using *calculation*(*1*) *zz check-s-d-weakening* ∗ **by** *metis*
moreover hence Θ ; Φ ; {||} ; *GNil* ; Δ′ ⊢ (*AS-seq s1′ s2*) ⇐ τ
using *check-seqI zz s′t* **by** *meson*
ultimately have Θ ; Φ ; Δ′ ⊢ ⟨ δ′ , *AS-seq s1′ s2* ⟩ ⇐ τ ∧ *setD* Δ ⊆ *setD* Δ′

using *zz config-typeI tt* **by** *meson*
then show *?case* **by** *meson*
next
case (*reduce-whileI x s1 s2 z' Φ δ*)

hence ∗: Θ ; Φ ; {||} ; *GNil* ; Δ ⊢ *AS-while s1 s2* ⇐ τ ∧ Θ ⊢ δ ∼ Δ ∧ (∀ *fd*∈*set* Φ. *check-fundef* Θ Φ *fd*)
using *config-type-elims* **by** *meson*

hence ∗∗:Θ ; Φ ; {||} ; *GNil* ; Δ ⊢ *AS-while s1 s2* ⇐ τ **by** *auto*
hence Θ ; Φ ; {||} ; *GNil* ; Δ ⊢ *AS-let2 x* (⦃ *z'* : *B-bool* | *TRUE* ⦄) *s1* (*AS-if* (*V-var x*) (*AS-seq s2* (*AS-while s1 s2*)) (*AS-val* (*V-lit L-unit*))) ⇐ τ
using *check-while reduce-whileI* **by** *auto*
thus *?case* **using** *config-typeI* ∗ **by** (*meson subset-refl*)

next
case (*reduce-caseI dc x' s' css Φ δ tyid v*)

hence ∗∗: Θ ; Φ ; {||} ; *GNil* ; Δ ⊢ *AS-match* (*V-cons tyid dc v*) *css* ⇐ τ ∧ Θ ⊢ δ ∼ Δ ∧ (∀ *fd*∈*set* Φ. *check-fundef* Θ Φ *fd*)
using *config-type-elims*[*OF reduce-caseI*(*2*)] **by** *metis*
hence ∗∗∗: Θ ; Φ ; {||} ; *GNil* ; Δ ⊢ *AS-match* (*V-cons tyid dc v*) *css* ⇐ τ **by** *auto*

let *?vcons* = *V-cons tyid dc v*

obtain *dclist tid* **and** *z*::*x* **where** *cv*: Θ; {||} ; *GNil* ⊢ (*V-cons tyid dc v*) ⇐ (⦃ *z* : *B-id tid* | *TRUE* ⦄) ∧
Θ ; Φ ; {||} ; *GNil* ; Δ ; *tid* ; *dclist* ; (*V-cons tyid dc v*) ⊢ *css* ⇐ τ ∧ *AF-typedef tid dclist* ∈ *set* Θ ∧
Θ ; {||} ; *GNil* ⊢ *V-cons tyid dc v* ⇐ ⦃ *z* : *B-id tid* | *TRUE* ⦄
using *check-s-elims*(*9*)[*OF* ∗∗∗] **by** *metis*

hence *vi*: Θ ; {||} ; *GNil* ⊢ *V-cons tyid dc v* ⇐ ⦃ *z* : *B-id tid* | *TRUE* ⦄ **by** *auto*
obtain *tcons* **where** *vi2*: Θ ; {||} ; *GNil* ⊢ *V-cons tyid dc v* ⇒ *tcons* ∧ Θ ; {||} ; *GNil* ⊢ *tcons* ≲ ⦃ *z* : *B-id tid* | *TRUE* ⦄
using *check-v-elims*(*1*)[*OF vi*] **by** *metis*
hence *vi1*: Θ ; {||} ; *GNil* ⊢ *V-cons tyid dc v* ⇒ *tcons* **by** *auto*

show *?case* **proof**(*rule infer-v-elims*(*4*)[*OF vi1*],*goal-cases*)
case (*1 dclist2 x2 b2 c2 z2' c2' z2*)
have *tyid* = *tid* **using** τ.*eq-iff* **using** *subtype-eq-base vi2 1* **by** *fastforce*
hence *deq*:*dclist* = *dclist2* **using** *check-v-wf wfX-wfY cv 1 wfTh-dclist-unique* **by** *metis*
have Θ ; Φ ; {||} ; *GNil* ; Δ ⊢ *s'*[*x'*::=*v*]$_{sv}$ ⇐ τ **proof**(*rule check-match*(*3*))
show ‹ Θ ; Φ ; {||} ; *GNil* ; Δ ; *tyid* ; *dclist* ; *?vcons* ⊢ *css* ⇐ τ› **using** ‹*tyid* = *tid*› *cv* **by** *auto*
show *distinct* (*map fst dclist*) **using** *wfTh-dclist-distinct check-v-wf wfX-wfY cv* **by** *metis*
show ‹*?vcons* = *V-cons tyid dc v*› **by** *auto*
show ‹{||} = {||}› **by** *auto*
show ‹(*dc*, ⦃ *x2* : *b2* | *c2* ⦄) ∈ *set dclist*› **using** *1 deq* **by** *auto*
show ‹*GNil* = *GNil*› **by** *auto*
show ‹*Some* (*AS-branch dc x' s'*) = *lookup-branch dc css*› **using** *reduce-caseI* **by** *auto*
show ‹Θ ; {||} ; *GNil* ⊢ *v* ⇐ ⦃ *x2* : *b2* | *c2* ⦄› **using** *1 check-v.intros* **by** *auto*
qed

thus *?case* **using** *config-typeI* ∗∗ **by** *blast*
qed

next
case (*reduce-let-fstI* Φ δ *x v1 v2 s*)
thus *?case* **using** *preservation-fst-snd order-refl* **by** *metis*
next
case (*reduce-let-sndI* Φ δ *x v1 v2 s*)
thus *?case* **using** *preservation-fst-snd order-refl* **by** *metis*
next
case (*reduce-let-concatI* Φ δ *x v1 v2 s*)
hence *elim*: Θ ; Φ ; {||} ; *GNil* ; Δ ⊢ *AS-let x* (*AE-concat* (*V-lit* (*L-bitvec v1*)) (*V-lit* (*L-bitvec v2*))) *s* ⇐ τ ∧
Θ ⊢ δ ∼ Δ ∧ (∀ *fd*∈*set* Φ. *check-fundef* Θ Φ *fd*)
using *config-type-elims* **by** *metis*

obtain *z*::*x* **where** *z*: *atom z* ♯ (*AE-concat* (*V-lit* (*L-bitvec v1*)) (*V-lit* (*L-bitvec v2*)), *GNil*, *CE-val* (*V-lit* (*L-bitvec* (*v1* @ *v2*))))
using *obtain-fresh* **by** *metis*

have Θ ; {||} ⊢$_{wf}$ *GNil* **using** *check-s-wf elim* **by** *auto*

have Θ ; Φ ; {||} ; *GNil* ; Δ ⊢ *AS-let x* (*AE-val* (*V-lit* (*L-bitvec* (*v1* @ *v2*)))) *s* ⇐ τ **proof**(*rule subtype-let*)
show ‹ Θ ; Φ ; {||} ; *GNil* ; Δ ⊢ *AS-let x* (*AE-concat* (*V-lit* (*L-bitvec v1*)) (*V-lit* (*L-bitvec v2*))) *s* ⇐ τ› **using** *elim* **by** *auto*
show ‹Θ ; Φ ; {||} ; *GNil* ; Δ ⊢ (*AE-concat* (*V-lit* (*L-bitvec v1*)) (*V-lit* (*L-bitvec v2*))) ⇒ {‖ *z* : *B-bitvec* | *CE-val* (*V-var z*) == (*CE-concat* ([*V-lit* (*L-bitvec v1*)]^{ce}) ([*V-lit* (*L-bitvec v2*)]^{ce}))‖} ›
(**is** Θ ; Φ ; {||} ; *GNil* ; Δ ⊢ *?e1* ⇒ *?t1*)
proof
show ‹ Θ ; {||} ; *GNil* ⊢$_{wf}$ Δ › **using** *check-s-wf elim* **by** *auto*
show ‹ Θ ⊢$_{wf}$ Φ › **using** *check-s-wf elim* **by** *auto*
show ‹ Θ ; {||} ; *GNil* ⊢ *V-lit* (*L-bitvec v1*) ⇒ {‖ *z* : *B-bitvec* | *CE-val* (*V-var z*) == *CE-val* (*V-lit* (*L-bitvec v1*)) ‖}›
using *infer-v-litI infer-l.intros* ‹Θ ; {||} ⊢$_{wf}$ *GNil*› *fresh-GNil* **by** *auto*
show ‹ Θ ; {||} ; *GNil* ⊢ *V-lit* (*L-bitvec v2*) ⇒ {‖ *z* : *B-bitvec* | *CE-val* (*V-var z*) == *CE-val* (*V-lit* (*L-bitvec v2*)) ‖}›
using *infer-v-litI infer-l.intros* ‹Θ ; {||} ⊢$_{wf}$ *GNil*› *fresh-GNil* **by** *auto*
show ‹*atom z* ♯ *AE-concat* (*V-lit* (*L-bitvec v1*)) (*V-lit* (*L-bitvec v2*))› **using** *z fresh-Pair* **by** *metis*
show ‹*atom z* ♯ *GNil*› **using** *z fresh-Pair* **by** *auto*
qed
show ‹Θ ; Φ ; {||} ; *GNil* ; Δ ⊢ *AE-val* (*V-lit* (*L-bitvec* (*v1* @ *v2*))) ⇒ {‖ *z* : *B-bitvec* | *CE-val* (*V-var z*) == *CE-val* (*V-lit* (*L-bitvec* (*v1* @ *v2*))) ‖} ›
(**is** Θ ; Φ ; {||} ; *GNil* ; Δ ⊢ *?e2* ⇒ *?t2*)
using *infer-e-valI infer-v-litI infer-l.intros* ‹Θ ; {||} ⊢$_{wf}$ *GNil*› *fresh-GNil check-s-wf elim* **by** *metis*
show ‹Θ ; {||} ; *GNil* ⊢ *?t2* ≲ *?t1*› **using** *subtype-concat check-s-wf elim* **by** *auto*
qed

thus *?case* **using** *config-typeI elim* **by** (*meson order-refl*)
next
case (*reduce-let-lenI* Φ δ *x v s*)

hence *elim*: Θ ; Φ ; $\{||\}$; *GNil* ; Δ $\vdash$ *AS-let x* (*AE-len* (*V-lit* (*L-bitvec v*))) *s* $\Leftarrow$ τ $\wedge$ $\Theta \vdash \delta \sim \Delta \wedge$ ($\forall$ *fd*$\in$*set* Φ. *check-fundef* Θ Φ *fd*)
using *check-s-elims config-type-elims* **by** *metis*

then obtain *t* **where** *t*: Θ ; Φ ; $\{||\}$; *GNil* ; Δ $\vdash$ *AE-len* (*V-lit* (*L-bitvec v*)) $\Rightarrow$ *t* **using** *check-s-elims* **by** *meson*

moreover then obtain *z*::*x* **where** *t* = $\{\!|$ *z* : *B-int* | *CE-val* (*V-var z*) == *CE-len* ([*V-lit* (*L-bitvec v*)]^{ce}) $|\!\}$ **using** *infer-e-elims* **by** *meson*

moreover obtain *z'*::*x* **where** *atom z'* $\sharp$ *v* **using** *obtain-fresh* **by** *metis*
moreover have Θ ; Φ ; $\{||\}$; *GNil* ; $\Delta \vdash$ *AE-val* (*V-lit* (*L-num* (*int* (*length v*)))) $\Rightarrow$ $\{\!|$ *z'* : *B-int* | *CE-val* (*V-var z'*) == *CE-val* (*V-lit* (*L-num* (*int* (*length v*)))) $|\!\}$
using *infer-e-valI infer-v-litI infer-l.intros*(*3*) *t check-s-wf elim*
by (*metis infer-l-form2 type-for-lit.simps*(*3*))

moreover have Θ ; $\{||\}$; *GNil* $\vdash$ $\{\!|$ *z'* : *B-int* | *CE-val* (*V-var z'*) == *CE-val* (*V-lit* (*L-num* (*int* (*length v*)))) $|\!\}$ $\lesssim$
$\{\!|$ *z* : *B-int* | *CE-val* (*V-var z*) == *CE-len* [(*V-lit* (*L-bitvec v*))]^{ce} $|\!\}$ **using** *subtype-len check-s-wf elim* **by** *auto*

ultimately have Θ ; Φ ; $\{||\}$; *GNil* ; $\Delta \vdash$ *AS-let x* (*AE-val* (*V-lit* (*L-num* (*int* (*length v*))))) *s* $\Leftarrow$ τ **using** *subtype-let* **by** (*meson elim*)
thus *?case* **using** *config-typeI elim* **by** (*meson order-refl*)
next
case (*reduce-let-splitI n v v1 v2* Φ δ *x s*)
hence *elim*: Θ ; Φ ; $\{||\}$; *GNil* ; Δ $\vdash$ *AS-let x* (*AE-split* (*V-lit* (*L-bitvec v*)) (*V-lit* (*L-num n*))) *s* $\Leftarrow$ τ $\wedge$
$\Theta \vdash \delta \sim \Delta \wedge$ ($\forall$ *fd*$\in$*set* Φ. *check-fundef* Θ Φ *fd*)
using *config-type-elims* **by** *metis*

obtain *z*::*x* **where** *z*: *atom z* $\sharp$ (*AE-split* (*V-lit* (*L-bitvec v*)) (*V-lit* (*L-num n*)), *GNil*, *CE-val* (*V-lit* (*L-bitvec* (*v1* @ *v2*))),
([*L-bitvec v1*]^{v}, [*L-bitvec v2*]^{v}))
using *obtain-fresh* **by** *metis*

have $*$:Θ ; $\{||\}$ $\vdash_{wf}$ *GNil* **using** *check-s-wf elim* **by** *auto*

have Θ ; Φ ; $\{||\}$; *GNil* ; $\Delta \vdash$ *AS-let x* (*AE-val* (*V-pair* (*V-lit* (*L-bitvec v1*)) (*V-lit* (*L-bitvec v2*)))) *s* $\Leftarrow$ τ **proof**(*rule subtype-let*)

show ‹ Θ ; Φ ; $\{||\}$; *GNil* ; Δ $\vdash$ *AS-let x* (*AE-split* (*V-lit* (*L-bitvec v*)) (*V-lit* (*L-num n*))) *s* $\Leftarrow$ τ › **using** *elim* **by** *auto*
show ‹Θ ; Φ ; $\{||\}$; *GNil* ; Δ $\vdash$ (*AE-split* (*V-lit* (*L-bitvec v*)) (*V-lit* (*L-num n*))) $\Rightarrow$ $\{\!|$ *z* : *B-pair B-bitvec B-bitvec*
| ((*CE-val* (*V-lit* (*L-bitvec v*))) == (*CE-concat* (*CE-fst* (*CE-val* (*V-var z*))) (*CE-snd* (*CE-val* (*V-var z*)))))
AND (((*CE-len* (*CE-fst* (*CE-val* (*V-var z*))))) == (*CE-val* (*V-lit* (*L-num n*)))) $|\!\}$ ›
(**is** Θ ; Φ ; $\{||\}$; *GNil* ; Δ $\vdash$ *?e1* $\Rightarrow$ *?t1*)
proof
show ‹ Θ ; $\{||\}$; *GNil* $\vdash_{wf}$ Δ › **using** *check-s-wf elim* **by** *auto*
show ‹ Θ $\vdash_{wf}$ Φ › **using** *check-s-wf elim* **by** *auto*

show ⟨ Θ ; {||} ; *GNil* ⊢ *V-lit* (*L-bitvec v*) ⇒ ⦃ *z* : *B-bitvec* | *CE-val* (*V-var z*) == *CE-val* (*V-lit* (*L-bitvec v*)) ⦄⟩
using *infer-v-litI infer-l.intros* ⟨Θ ; {||} $\vdash_{wf}$ *GNil*⟩ *fresh-GNil* **by** *auto*
show Θ ; {||} ; *GNil* ⊢ [*L-num*
n $]^v$ ⇐ ⦃ *z* : *B-int* | [*leq* [[*L-num*
0 $]^v$ $]^{ce}$ [[*z* $]^v$ $]^{ce}$ $]^{ce}$ == [[*L-true* $]^v$ $]^{ce}$ *AND*
[*leq* [[*z* $]^v$ $]^{ce}$ [|| [[*L-bitvec*
v $]^v$ $]^{ce}$ ||$]^{ce}$ $]^{ce}$ == [[*L-true* $]^v$ $]^{ce}$ ⦄ **using** *split-n reduce-let-splitI check-v-num-leq*
∗ *wfX-wfY* **by** *metis*
show ⟨*atom z* ♯ *AE-split* [*L-bitvec v* $]^v$ [*L-num n* $]^v$⟩ **using** *z fresh-Pair* **by** *auto*
show ⟨*atom z* ♯ *GNil*⟩ **using** *z fresh-Pair* **by** *auto*
show ⟨*atom z* ♯ *AE-split* [*L-bitvec v* $]^v$ [*L-num n* $]^v$⟩ **using** *z fresh-Pair* **by** *auto*
show ⟨*atom z* ♯ *GNil*⟩ **using** *z fresh-Pair* **by** *auto*
show ⟨*atom z* ♯ *AE-split* [*L-bitvec v* $]^v$ [*L-num n* $]^v$⟩ **using** *z fresh-Pair* **by** *auto*
show ⟨*atom z* ♯ *GNil*⟩ **using** *z fresh-Pair* **by** *auto*
qed

show ⟨Θ ; Φ ; {||} ; *GNil* ; Δ ⊢ *AE-val* (*V-pair* (*V-lit* (*L-bitvec v1*)) (*V-lit* (*L-bitvec v2*))) ⇒ ⦃ *z* : *B-pair B-bitvec B-bitvec* | *CE-val* (*V-var z*) == *CE-val* ((*V-pair* (*V-lit* (*L-bitvec v1*)) (*V-lit* (*L-bitvec v2*)))) ⦄ ⟩
(**is** Θ ; Φ ; {||} ; *GNil* ; Δ ⊢ *?e2* ⇒ *?t2*)
apply(*rule infer-e-valI*)
using *check-s-wf elim* **apply** *metis*
using *check-s-wf elim* **apply** *metis*
apply(*rule infer-v-pairI*)
using *z fresh-prodN* **apply** *metis*
using *fresh-GNil* **apply** *metis*
using *infer-v-litI infer-l.intros* ⟨Θ ; {||} $\vdash_{wf}$ *GNil*⟩ **apply** *blast+*
done
show ⟨Θ ; {||} ; *GNil* ⊢ *?t2* ≲ *?t1*⟩ **using** *subtype-split check-s-wf elim reduce-let-splitI* **by** *auto*
qed

thus *?case* **using** *config-typeI elim* **by** (*meson order-refl*)
next
case (*reduce-assert1I* Φ δ *c v*)

hence *elim*: Θ ; Φ ; {||} ; *GNil* ; Δ ⊢ *AS-assert c* $[v]^s$ ⇐ τ ∧
Θ ⊢ δ ∼ Δ ∧ (∀ *fd*∈*set* Φ. *check-fundef* Θ Φ *fd*)
using *config-type-elims reduce-assert1I* **by** *metis*
hence ∗:Θ ; Φ ; {||} ; *GNil* ; Δ ⊢ *AS-assert c* $[v]^s$ ⇐ τ **by** *auto*

have Θ ; Φ ; {||} ; *GNil* ; Δ ⊢ $[v]^s$ ⇐ τ **using** *check-assert-s* ∗ **by** *metis*
thus *?case* **using** *elim config-typeI* **by** *blast*
next
case (*reduce-assert2I* Φ δ *s* δ′ *s*′ *c*)

hence *elim*: Θ ; Φ ; {||} ; *GNil* ; Δ ⊢ *AS-assert c s* ⇐ τ ∧
Θ ⊢ δ ∼ Δ ∧ (∀ *fd*∈*set* Φ. *check-fundef* Θ Φ *fd*)
using *config-type-elims* **by** *metis*
hence ∗:Θ ; Φ ; {||} ; *GNil* ; Δ ⊢ *AS-assert c s* ⇐ τ **by** *auto*

have *cv*: Θ ; Φ ; {||} ; *GNil* ; Δ ⊢ *s* ⇐ τ ∧ Θ ; {||} ; *GNil* ⊨ *c* **using** *check-assert-s* ∗ **by** *metis*

hence Θ ; Φ ; $\Delta \vdash \langle\ \delta\ ,\ s\ \rangle \Leftarrow \tau$ **using** *elim config-typeI* **by** *simp*
then obtain Δ' **where** *D*: Θ ; Φ ; $\Delta' \vdash \langle\ \delta'\ ,\ s'\ \rangle \Leftarrow \tau\ \wedge$ *setD* $\Delta \subseteq$ *setD* Δ' **using** *reduce-assert2I* **by** *metis*
hence $**$:Θ ; Φ ; {||} ; *GNil* ; $\Delta' \vdash s' \Leftarrow \tau \wedge \Theta \vdash \delta' \sim \Delta'$ **using** *config-type-elims* **by** *metis*

obtain *x*::*x* **where** *x*:*atom x* $\sharp$ (Θ, Φ, ({||}::*bv fset*), *GNil*, Δ', *c*, τ, s') **using** *obtain-fresh* **by** *metis*

have $*$:Θ ; Φ ; {||} ; *GNil* ; Δ' $\vdash$ *AS-assert c s'* $\Leftarrow \tau$ **proof**
show *atom x* $\sharp$ (Θ, Φ, {||}, *GNil*, Δ', *c*, τ, s') **using** *x* **by** *auto*
have Θ ; {||} ; *GNil* $\vdash_{wf}$ *c* **using** $*$ *check-s-wf* **by** *auto*
hence *wfg*:Θ ; {||} $\vdash_{wf}$ (*x*, *B-bool*, *c*) $\#_\Gamma$ *GNil* **using** *wfC-wfG wfB-boolI check-s-wf* $*$ *fresh-GNil* **by** *auto*
moreover have *cs*: Θ ; Φ ; {||} ; *GNil* ; $\Delta' \vdash s' \Leftarrow \tau$ **using** $**$ **by** *auto*
ultimately show Θ ; Φ ; {||} ; (*x*, *B-bool*, *c*) $\#_\Gamma$ *GNil* ; $\Delta' \vdash s' \Leftarrow \tau$ **using** *check-s-g-weakening(1)*[*OF cs - wfg*] *setG.simps* **by** *simp*
show Θ ; {||} ; *GNil* $\models$ *c* **using** *cv* **by** *auto*
show Θ ; {||} ; *GNil* $\vdash_{wf}$ Δ' **using** *check-s-wf* $**$ **by** *auto*
qed

thus *?case* **using** *elim config-typeI D* $**$ **by** *metis*
qed

thm *valid-wfC*

lemma *preservation-many*:
fixes *s*::*s* **and** *s'*::*s*
assumes $\Phi \vdash \langle\ \delta\ ,\ s\ \rangle \longrightarrow^* \langle\ \delta'\ ,\ s'\ \rangle$
shows Θ ; Φ ; $\Delta \vdash \langle\ \delta\ ,\ s\ \rangle \Leftarrow \tau \Longrightarrow \exists \Delta'.\ \Theta$; Φ ; $\Delta' \vdash \langle\ \delta'\ ,\ s'\ \rangle \Leftarrow \tau \wedge$ *setD* $\Delta \subseteq$ *setD* Δ'
using *assms* **proof**(*induct arbitrary*: Δ *rule*: *reduce-stmt-many.induct*)
case (*reduce-stmt-many-oneI* Φ δ *s* δ' *s'*)
then show *?case* **using** *preservation* **by** *simp*
next
case (*reduce-stmt-many-manyI* Φ δ *s* δ' *s'* δ'' *s''*)
then show *?case* **using** *preservation subset-trans* **by** *metis*
qed

16.3 Progress

Well typed program is either a value or we can make a step

lemma *check-let-op-infer*:
assumes Θ ; Φ ; {||} ; Γ ; $\Delta \vdash$ *LET x* = (*AE-op opp v1 v2*) *IN s* $\Leftarrow \tau$ **and** *supp* (*LET x* = (*AE-op opp v1 v2*) *IN s*) $\subseteq$ *atom*`*fst*`*setD* Δ
shows $\exists$ *z b c*. Θ ; Φ ; {||} ; Γ ; $\Delta \vdash$ (*AE-op opp v1 v2*) $\Rightarrow$ $\{\!|z{:}b|c|\!\}$
proof −
have *xx*: Θ ; Φ ; {||} ; Γ ; $\Delta \vdash$ *LET x* = (*AE-op opp v1 v2*) *IN s* $\Leftarrow \tau$ **using** *assms* **by** *simp*
then show *?thesis* **using** *check-s-elims(2)*[*OF xx*] **by** *meson*
qed

lemma *infer-pair*:
assumes Θ ; *B*; $\Gamma \vdash$ *v* $\Rightarrow$ $\{\!|$ *z* : *B-pair b1 b2* | *c* $|\!\}$ **and** *supp v* = {}

obtains *v1* **and** *v2* **where** $v = V\text{-}pair\ v1\ v2$
using *assms* **proof**(*nominal-induct v rule*: *v.strong-induct*)
case (*V-lit x*)
then show *?case* **by** *auto*
next
case (*V-var x*)
then show *?case* **using** *v.supp supp-at-base* **by** *auto*
next
case (*V-pair x1a x2a*)
then show *?case* **by** *auto*
next
case (*V-cons x1a x2a x3*)
then show *?case* **by** *auto*
next
case (*V-consp x1a x2a x3 x4*)
then show *?case* **by** *auto*
qed

lemma *progress-fst*:
assumes $\Theta\ ;\ \Phi\ ;\ \{||\}\ ;\ \Gamma\ ;\ \Delta\ \vdash LET\ x = (AE\text{-}fst\ v)\ IN\ s \Leftarrow \tau$ **and** $\Theta \vdash \delta \sim \Delta$ **and** $supp\ (LET\ x = (AE\text{-}fst\ v)\ IN\ s) \subseteq atom`fst`setD\ \Delta$
shows $\exists \delta'\ s'.\ \Phi\ \vdash \langle\ \delta\ ,\ LET\ x = (AE\text{-}fst\ v)\ IN\ s\ \rangle \longrightarrow \langle\ \delta'\ ,\ s'\ \rangle$
proof −
have $*$:$supp\ v = \{\}$ **using** *assms s-branch-s-branch-list.supp* **by** *auto*
obtain *z* **and** *b* **and** *c* **where** $\Theta\ ;\ \Phi\ ;\ \{||\}\ ;\ \Gamma\ ;\ \Delta \vdash\ (AE\text{-}fst\ v\ ) \Rightarrow \{\!|\ z : b\ \mid c\ |\!\}$
using *check-s-elims*(*2*) **using** *assms* **by** *meson*
moreover obtain z' **and** b' **and** c' **where** $\Theta\ ;\ \{||\}\ ;\ \Gamma \vdash\ v\ \Rightarrow \{\!|\ z' : B\text{-}pair\ b\ b'\ \mid c'\ |\!\}$
using *infer-e-elims*(*8*) **using** *calculation* **by** *auto*
moreover then obtain *v1* **and** *v2* **where** $V\text{-}pair\ v1\ v2 = v$
using $*$ *infer-pair* **by** *metis*
ultimately show *?thesis* **using** *reduce-let-fstI assms* **by** *metis*
qed

lemma *progress-let*:
assumes $\Theta\ ;\ \Phi\ ;\ \{||\}\ ;\ \Gamma\ ;\ \Delta \vdash LET\ x = e\ IN\ s \Leftarrow \tau$ **and** $\Theta \vdash \delta \sim \Delta$ **and** $supp\ (LET\ x = e\ IN\ s) \subseteq atom\ `\ fst\ `\ setD\ \Delta$ **and** *sble* $\Theta\ \Gamma$
shows $\exists \delta'\ s'.\ \Phi\ \vdash \langle\ \delta\ ,\ LET\ x = e\ IN\ s\rangle \longrightarrow \langle\ \delta'\ ,\ s'\rangle$
using *assms*
proof(*nominal-induct e rule*: *e.strong-induct*)
case (*AE-val v*)
then show *?case* **using** *reduce-stmt-elims reduce-let-valI*
proof −
show *?thesis*
by (*metis* (*no-types*) *reduce-let-valI*)
qed
next
case (*AE-app f v*)
obtain τ'' **where** $\Theta\ ;\ \Phi\ ;\ \{||\}\ ;\ \Gamma\ ;\ \Delta \vdash\ (AE\text{-}app\ f\ v) \Rightarrow \tau''$
using *check-s-elims*(*2*)[*OF AE-app*(*1*)] **by** *metis*

hence $\exists\, y\ b\ c\ \tau'\ s'.$ *Some* (*AF-fundef f* (*AF-fun-typ-none* (*AF-fun-typ y b c* τ' s'))) = *lookup-fun* Φ *f* **using** *infer-e-app2E* **by** *metis*
then obtain *y b c* τ' s' **where** $*$:*Some* (*AF-fundef f* (*AF-fun-typ-none* (*AF-fun-typ y b c* τ' s'))) = *lookup-fun* Φ *f* **by** *auto*
hence $\Phi \vdash \langle\ \delta\ ,\ AS\text{-}let\ x\ (AE\text{-}app\ f\ v)\ s\ \rangle \longrightarrow \langle\ \delta\ ,\ AS\text{-}let2\ x\ \tau'[y::=v]_{\tau v}\ s'[y::=v]_{sv}\ s\ \rangle$ **using** *reduce-let-appI* **by** *auto*
thus *?case* **by** *meson*
next
case (*AE-appP f b' v*)
obtain τ'' **where** $\Theta\ ;\ \Phi\ ;\ \{||\}\ ;\ \Gamma\ ;\ \Delta \vdash\ (AE\text{-}appP\ f\ b'\ v) \Rightarrow \tau''$
using *check-s-elims AE-appP* **by** *metis*

hence $\exists\, bv\ y\ b\ c\ \tau'\ s'.$ *Some* (*AF-fundef f* (*AF-fun-typ-some bv* (*AF-fun-typ y b c* τ' s'))) = *lookup-fun* Φ *f* **using** *infer-e-app2E* **by** *blast*
then obtain *bv y b c* τ' s' **where** $*$:*Some* (*AF-fundef f* (*AF-fun-typ-some bv* (*AF-fun-typ y b c* τ' s'))) = *lookup-fun* Φ *f* **by** *auto*
hence $\Phi \vdash \langle\ \delta\ ,\ AS\text{-}let\ x\ (AE\text{-}appP\ f\ b'\ v)\ s\ \rangle \longrightarrow \langle\ \delta\ ,\ AS\text{-}let2\ x\ \tau'[bv::=b']_{\tau b}[y::=v]_{\tau v}\ s'[bv::=b']_{sb}[y::=v]_{sv}\ s\ \rangle$ **using** *reduce-let-appPI* **by** *simp*
thus *?case* **by** *metis*
next
case (*AE-op opp v1 v2*)
then obtain *z* **and** *b* **and** *c* **where** $\Theta\ ;\ \Phi\ ;\ \{||\}\ ;\ \Gamma\ ;\ \Delta \vdash\ (AE\text{-}op\ opp\ v1\ v2) \Rightarrow \{\!|z{:}b|c|\!\}$ **using** *check-let-op-infer* **by** *meson*
have *vf*: *supp v1* = {} ∧ *supp v2* = {} **using** *AE-op s-branch-s-branch-list.supp* **by** *auto*
consider *opp* = *Plus* | *opp* = *LEq* **using** *opp.exhaust* **by** *meson*
thus *?case* **proof**(*cases*)
case *1*
hence $\Theta\ ;\ \Phi\ ;\ \{||\}\ ;\ \Gamma\ ;\ \Delta\ \vdash (AS\text{-}let\ x\ (AE\text{-}op\ Plus\ v1\ v2)\ s) \Leftarrow \tau$ **using** *AE-op.prems* **by** *blast*
then obtain *z* **and** *b* **and** *c* **where** *infer-e* Θ Φ {||} Γ Δ (*AE-op Plus v1 v2*) ({|| *z*:*b*|*c*||}) **using** *check-s-elims*(*2*)
using *1* ‹*infer-e* Θ Φ {||} Γ Δ (*AE-op opp v1 v2*) ({|| *z* : *b* | *c* ||})› **by** *auto*
hence $\exists$ *z1 c1 z2 c2*. *infer-v* Θ {||} Γ *v1* ({|| *z1* : *B-int* | *c1* ||}) ∧ *infer-v* Θ {||} Γ *v2* ({|| *z2* : *B-int* | *c2* ||}) **using** *infer-e-elims* **by** *blast*
then obtain *n1* **and** *n2* **where** *v1* = *V-lit* (*L-num n1*) ∧ *v2* = *V-lit* (*L-num n2*) **using** *infer-int vf* **by** *metis*
have $\Phi \vdash \langle \delta\ ,\ AS\text{-}let\ x\ (AE\text{-}op\ Plus\ ((V\text{-}lit\ (L\text{-}num\ n1)))\ ((V\text{-}lit\ (L\text{-}num\ n2))))\ \ s\ \rangle \longrightarrow \langle\ \delta\ ,\ AS\text{-}let\ x\ \ (AE\text{-}val\ (V\text{-}lit\ (L\text{-}num\ (\ ((\ n1)+(n2))))))\ s\ \rangle$
by (*simp add*: *reduce-let-plusI*)
thus *?thesis*
by (*metis 1* ‹$\bigwedge$*thesis*. ($\bigwedge$*n1 n2*. *v1* = *V-lit* (*L-num n1*) ∧ *v2* = *V-lit* (*L-num n2*) ⟹ *thesis*) ⟹ *thesis*› *reduce-let-plusI*)
next
case *2*
hence $\Theta\ ;\ \Phi\ ;\ \{||\}\ ;\ \Gamma\ ;\ \Delta\ \vdash (AS\text{-}let\ x\ \ (AE\text{-}op\ LEq\ v1\ v2)\ s) \Leftarrow \tau$ **using** *AE-op.prems* **by** *blast*
then obtain *z* **and** *b* **and** *c* **where** *infer-e* Θ Φ {||} Γ Δ (*AE-op LEq v1 v2*) ({|| *z*:*b*|*c*||}) **using** *check-s-elims*(*2*)
using *2* ‹*infer-e* Θ Φ {||} Γ Δ (*AE-op opp v1 v2*) ({|| *z* : *b* | *c* ||})› *vf* **by** *metis*
hence $\exists$ *z1 c1 z2 c2*. *infer-v* Θ {||} Γ *v1* ({|| *z1* : *B-int* | *c1* ||}) ∧ *infer-v* Θ {||} Γ *v2* ({|| *z2* : *B-int* | *c2* ||}) **using** *infer-e-elims vf* **by** *blast*
then obtain *n1* **and** *n2* **where** *v1* = *V-lit* (*L-num n1*) ∧ *v2* = *V-lit* (*L-num n2*) **using** *infer-int vf* **by** *metis*
obtain *b* **where** *b* = (*if n1* ≤ *n2 then L-true else L-false*) **by** *simp*

hence $\Phi \vdash \langle\ \delta\ ,\ AS\text{-}let\ x\ \ (AE\text{-}op\ LEq\ ((V\text{-}lit\ (L\text{-}num\ n1)))\ ((V\text{-}lit\ (L\text{-}num\ n2))))\ \ s\ \rangle \longrightarrow \langle\ \delta\ ,\ AS\text{-}let\ x\ \ (AE\text{-}val\ (V\text{-}lit\ (b)))\ s\ \rangle$
using *reduce-let-leqI* **by** *blast*
thus *?thesis*
by (*metis 2* ‹$\bigwedge thesis.\ (\bigwedge n1\ n2.\ v1 = V\text{-}lit\ (L\text{-}num\ n1) \wedge v2 = V\text{-}lit\ (L\text{-}num\ n2) \Longrightarrow thesis) \Longrightarrow thesis$› *reduce-let-leqI*)
qed
next
case (*AE-fst v*)
thus *?case* **using** *progress-fst* **by** *auto*
next
case (*AE-snd v*)
have $*$:*supp v* = {} **using** *AE-snd s-branch-s-branch-list.supp* **by** *auto*
then obtain *z* **and** *b* **and** *c* **where** $\Theta\ ;\ \Phi\ ;\ \{||\}\ ;\ \Gamma\ ;\ \Delta\ \vdash\ (AE\text{-}snd\ v\ ) \Rightarrow \{\!|\ z : b\ \ |\ c\ |\!\}$
using *check-s-elims(2)* **using** *AE-snd.prems* **by** *meson*
moreover obtain z' **and** b' **and** c' **where** $\Theta\ ;\ \{||\}\ ;\ \Gamma \vdash\ v\ \Rightarrow \{\!|\ z' : B\text{-}pair\ b'\ b\ \ |\ c'\ |\!\}$
using *infer-e-elims(8)* **using** *calculation* **by** *auto*
moreover then obtain *v1* **and** *v2* **where** *V-pair v1 v2* = *v*
using $*$ *infer-pair* **by** *metis*

ultimately show *?case* **using** *reduce-let-sndI AE-snd* **by** *metis*
next
case (*AE-mvar u*)
then obtain *z* **and** *b* **and** *c* **where** $\Theta\ ;\ \Phi\ ;\ \{||\}\ ;\ \Gamma\ ;\ \Delta \vdash\ (AE\text{-}mvar\ u) \Rightarrow \{\!|\ z : b\ \ |\ c\ |\!\}$
using *check-s-elims(2)* **by** *meson*
hence (*u*, $\{\!|\ z : b\ \ |\ c\ |\!\}$) $\in$ *setD* Δ **using** *infer-e-elims(10)* **by** *meson*
then obtain *v* **where** (*u*,*v*) $\in$ *set* δ **using** *assms delta-sim-delta-lookup* **by** *meson*
then show *?case* **using** *reduce-let-mvar* **by** *blast*
next
case (*AE-len v*)
have $*$:*supp v* = {} **using** *AE-len s-branch-s-branch-list.supp* **by** *auto*
then obtain *z* **and** *b* **and** *c* **where** $\Theta\ ;\ \Phi\ ;\ \{||\}\ ;\ \Gamma\ ;\ \Delta \vdash\ (AE\text{-}len\ v) \Rightarrow \{\!|\ z : b\ \ |\ c\ |\!\}$
using *check-s-elims(2) AE-len* **by** *meson*
then obtain z' **and** c' **where** $\Theta\ ;\ \{||\}\ ;\ \Gamma \vdash v \Rightarrow \{\!|\ z' : B\text{-}bitvec\ |\ c'\ |\!\}$ **using** *infer-e-elims* **by** *auto*
then obtain *bv* **where** *v* = *V-lit* (*L-bitvec bv*) **using** *infer-bitvec* $*$ **by** *metis*
thus *?case* **using** *reduce-let-lenI AE-len* **by** *metis*
next
case (*AE-concat v1 v2*)
have $*$:*supp v1* = {} $\wedge$ *supp v2* = {} **using** *AE-concat s-branch-s-branch-list.supp* **by** *auto*
then obtain *z* **and** *b* **and** *c* **where** $\Theta\ ;\ \Phi\ ;\ \{||\}\ ;\ \Gamma\ ;\ \Delta \vdash\ (AE\text{-}concat\ v1\ v2) \Rightarrow \{\!|\ z : b\ \ |\ c\ |\!\}$
using *check-s-elims(2) AE-concat* **by** *meson*
then obtain *z1* **and** *c1* **and** *z2* **and** *c2* **where** $\Theta\ ;\ \{||\}\ ;\ \Gamma \vdash v1 \Rightarrow \{\!|\ z1 : B\text{-}bitvec\ |\ c1\ |\!\} \wedge \Theta\ ;\ \{||\}\ ;\ \Gamma \vdash v2 \Rightarrow \{\!|\ z2 : B\text{-}bitvec\ |\ c2\ |\!\}$ **using** *infer-e-elims* **by** *auto*
then obtain *bv1* **and** *bv2* **where** *v1* = *V-lit* (*L-bitvec bv1*) $\wedge$ *v2* = *V-lit* (*L-bitvec bv2*) **using** *infer-bitvec* $*$ **by** *metis*
thus *?case* **using** *reduce-let-concatI AE-concat* **by** *metis*
next
case (*AE-split v1 v2*)
have *vs*:*supp v1* = {} $\wedge$ *supp v2* = {} **using** *AE-split s-branch-s-branch-list.supp* **by** *auto*
then obtain *z* **and** *b* **and** *c* **where** $*$:$\Theta\ ;\ \Phi\ ;\ \{||\}\ ;\ \Gamma\ ;\ \Delta \vdash\ (AE\text{-}split\ v1\ v2) \Rightarrow \{\!|\ z : b\ \ |\ c\ |\!\}$
using *check-s-elims(2) AE-split* **by** *meson*

then obtain *z1* **and** *c1* **and** *z2* **and** *z3* **where** $**:\Theta$; $\{||\}$; $\Gamma \vdash v1 \Rightarrow \{\!|\ z1 : B\text{-}bitvec \mid c1\ |\!\} \wedge \Theta$; $\{||\}$; $\Gamma \vdash v2 \Leftarrow \{\!|\ z2 : B\text{-}int \mid$ [[*leq* [[*L-num* *0*]^v]^{ce} [[*z2*]^v]^{ce}]^{ce} == [[*L-true*]^v]^{ce} *AND* [*leq* [[*z2*]^v]^{ce} [|| [*v1*]^{ce} ||]^{ce}]^{ce} == [[*L-true*]^v]^{ce} $|\!\}$ $\wedge$ *atom z2* $\sharp$ Γ
using *infer-e-elims*(*22*)[*OF* $*$] **by** *metis*
then obtain *bv* **and** *n* **where** $*$: *v1* = *V-lit* (*L-bitvec bv*) $\wedge$ *v2* = *V-lit* (*L-num n*) **using** *infer-bitvec check-int vs* **by** *metis*
moreover have *atom z2* $\sharp$ Γ **using** $**$ **by** *auto*
ultimately have *0* $\leq n \wedge n \leq$ *int* (*length bv*) **using** *check-v-range*[*OF* - $*$] $**$ *AE-split* **by** *metis*
then obtain *bv1* **and** *bv2* **where** *split n bv* (*bv1* , *bv2*) **using** *obtain-split* **by** *metis*

thus *?case* **using** *reduce-let-splitI*[*of n bv bv1 bv2* Φ δ *x s*] *AE-split* $*$ **by** *metis*
qed

lemma *check-css-lookup-branch-exist*:
fixes *s*::*s* **and** *cs*::*branch-s* **and** *css*::*branch-list* **and** *v*::*v*
shows
Θ ; Φ ; *B* ; *G* ; Δ $\vdash$ *s* $\Leftarrow \tau \Longrightarrow$ *True* **and**
check-branch-s Θ Φ $\{||\}$ *GNil* Δ *tid dc const v cs* $\tau \Longrightarrow$ *True* **and**
Θ ; Φ ; $\mathcal{B}$; Γ ; Δ ; *tid* ; *dclist* ; *v* $\vdash$ *css* $\Leftarrow \tau \Longrightarrow$ (*dc*, *t*) $\in$ *set dclist* $\Longrightarrow$
$\exists x' s'$. *Some* (*AS-branch dc x' s'*) = *lookup-branch dc css*
proof(*nominal-induct* τ **and** τ **and** τ *rule*: *check-s-check-branch-s-check-branch-list.strong-induct*)
case (*check-branch-list-consI* Θ Φ $\mathcal{B}$ Γ Δ *tid cons const v cs* τ *dclist css*)
then show *?case* **using** *lookup-branch.simps check-branch-list-finalI* **by** *force*
next
case (*check-branch-list-finalI* Θ Φ $\mathcal{B}$ Γ Δ *tid cons const v cs* τ)
then show *?case* **using** *lookup-branch.simps check-branch-list-finalI* **by** *force*
qed(*auto*+)

lemma *progress-aux*:
fixes *s*::*s* **and** *cs*::*branch-s* **and** *css*::*branch-list*
shows Θ ; Φ ; $\mathcal{B}$; Γ ; Δ $\vdash$ *s* $\Leftarrow \tau \Longrightarrow \mathcal{B} = \{||\} \Longrightarrow$ *sble* Θ $\Gamma \Longrightarrow$ *supp s* $\subseteq$ *atom* ‘ *fst* ‘ *setD* Δ $\Longrightarrow \Theta \vdash \delta \sim \Delta \Longrightarrow$
$(\exists v.\ s = [v]^s) \vee (\exists \delta' s'.\ \Phi \vdash \langle\ \delta\ ,\ s\ \rangle \longrightarrow \langle\ \delta'\ ,\ s'\ \rangle)$ **and**
Θ ; Φ ; $\{||\}$; Γ ; Δ ; *tid* ; *dc* ; *const* ; *v2* $\vdash$ *cs* $\Leftarrow \tau \Longrightarrow$ *supp cs* = $\{\} \Longrightarrow$ *True*
Θ ; Φ ; $\{||\}$; Γ ; Δ ; *tid* ; *dclist* ; *v2* $\vdash$ *css* $\Leftarrow \tau \Longrightarrow$ *supp css* = $\{\} \Longrightarrow$ *True*
proof(*induct rule*: *check-s-check-branch-s-check-branch-list.inducts*)
case (*check-valI* Δ Θ Γ *v* τ' τ)
then show *?case* **by** *auto*
next
case (*check-letI x* Θ Φ $\mathcal{B}$ Γ Δ *e* τ *z s b c*)
hence Θ ; Φ ; $\{||\}$; Γ ; Δ $\vdash$ *AS-let x e s* $\Leftarrow \tau$ **using** *Typing.check-letI* **by** *meson*
then show *?case* **using** *progress-let check-letI* **by** *metis*
next
case (*check-branch-s-branchI* Θ $\mathcal{B}$ Γ Δ τ *const x* Φ *tid cons v s*)
then show *?case* **by** *auto*
next
case (*check-branch-list-consI* Θ Φ $\mathcal{B}$ Γ Δ *tid dclist v cs* τ *css*)
then show *?case* **by** *auto*

next
case (*check-branch-list-finalI* Θ Φ $\mathcal{B}$ Γ Δ *tid dclist v cs* τ)
then show *?case* **by** *auto*
next
case (*check-ifI z* Θ Φ $\mathcal{B}$ Γ Δ *v s1 s2* τ)
have *supp v* = {} **using** *check-ifI s-branch-s-branch-list.supp* **by** *auto*
hence *v* = *V-lit L-true* $\vee$ *v* = *V-lit L-false* **using** *check-bool-options check-ifI* **by** *auto*
then show *?case* **using** *reduce-if-falseI reduce-if-trueI check-ifI* **by** *meson*
next
case (*check-let2I x* Θ Φ $\mathcal{B}$ *G* Δ *t s1* τ *s2*)
then consider ($\exists$ *v*. *s1* = *AS-val v*) | ($\exists$ δ' *a*. Φ $\vdash$ $\langle$ δ , *s1* $\rangle$ $\longrightarrow$ $\langle$ δ' , *a* $\rangle$) **by** *auto*
then show *?case* **proof**(*cases*)
case *1*
then show *?thesis* **using** *reduce-let2-valI* **by** *fast*
next
case *2*
then show *?thesis* **using** *reduce-let2I check-let2I* **by** *meson*
qed
next
case (*check-varI u* Θ Φ $\mathcal{B}$ Γ Δ τ' *v* τ *s*)

obtain *uu*::*u* **where** *uf*: *atom uu* $\sharp$ (*u*,δ,*s*) **using** *obtain-fresh* **by** *blast*
obtain *sa* **where** (*uu* $\leftrightarrow$ *u*) $\cdot$ *s* = *sa* **by** *presburger*
moreover have *atom uu* $\sharp$ *s* **using** *uf fresh-prod3* **by** *auto*
ultimately have *AS-var uu* τ' *v sa* = *AS-var u* τ' *v s* **using** *s-branch-s-branch-list.eq-iff*(*7*) *Abs1-eq-iff*(*3*)[*of uu sa u s*] **by** *auto*

moreover have *atom uu* $\sharp$ δ **using** *uf fresh-prod3* **by** *auto*
ultimately have Φ $\vdash$ $\langle$ δ , *AS-var u* τ' *v s* $\rangle$ $\longrightarrow$ $\langle$ (*uu*, *v*) # δ , *sa* $\rangle$
using *reduce-varI uf* **by** *metis*
then show *?case* **by** *auto*
next
case (*check-assignI* Δ *u* τ *P G v z* τ')
then show *?case* **using** *reduce-assignI* **by** *blast*
next
case (*check-whileI* Θ Φ $\mathcal{B}$ Γ Δ *s1 z s2* τ')
obtain *x*::*x* **where** *atom x* $\sharp$ (*s1*,*s2*) **using** *obtain-fresh* **by** *metis*
moreover obtain *z*::*x* **where** *atom z* $\sharp$ *x* **using** *obtain-fresh* **by** *metis*
ultimately show *?case* **using** *reduce-whileI* **by** *fast*
next
case (*check-seqI P* Φ $\mathcal{B}$ *G* Δ *s1 z s2* τ)
thus *?case* **proof**(*cases* $\exists$ *v*. *s1* = *AS-val v*)
case *True*
then obtain *v* **where** *v*: *s1* = *AS-val v* **by** *blast*
hence *supp v* = {} **using** *check-seqI* **by** *auto*
have $\exists$ *z1 c1*. *P* ; $\mathcal{B}$; *G* $\vdash$ *v* $\Rightarrow$ ({| *z1* : *B-unit* | *c1* |}) **proof** −
obtain *t* **where** *t*:*P* ; $\mathcal{B}$; *G* $\vdash$ *v* $\Rightarrow$ *t* $\wedge$ *P* ; $\mathcal{B}$; *G* $\vdash$ *t* $\lesssim$ ({| *z* : *B-unit* | *TRUE* |})
using *v check-seqI*(*1*) *check-s-elims*(*1*) **by** *blast*
obtain *z1* **and** *b1* **and** *c1* **where** *teq*: *t* = ({| *z1* : *b1* | *c1* |}) **using** *obtain-fresh-z* **by** *meson*
hence *b1* = *B-unit* **using** *subtype-eq-base t* **by** *meson*
thus *?thesis* **using** *t teq* **by** *fast*
qed

then obtain *z1* **and** *c1* **where** P ; $\mathcal{B}$; $G \vdash v \Rightarrow (\{\!|\ z1 : B\text{-}unit \mid c1\ |\!\})$ **by** *auto*
hence v = *V-lit L-unit* **using** *infer-v-unit-form* ‹*supp* $v = \{\}$› **by** *simp*
hence *s1* = *AS-val* (*V-lit L-unit*) **using** v **by** *auto*
then show *?thesis* **using** *check-seqI reduce-seq1I* **by** *meson*
next
case *False*
then show *?thesis* **using** *check-seqI reduce-seq2I*
by (*metis Un-subset-iff s-branch-s-branch-list.supp*(*9*))
qed

next
case (*check-caseI* Θ Φ $\mathcal{B}$ Γ Δ *tid dclist v cs* τ z)
hence *supp* $v = \{\}$ **by** *auto*

then obtain v' **and** *dc* **and** t::τ **where** v: v = *V-cons tid dc* $v' \wedge$ $(dc, t) \in$ *set dclist*
using *check-v-tid-form check-caseI* **by** *metis*
obtain z **and** b **and** c **where** *teq*: $t = (\{\!|\ z : b \mid c\ |\!\})$ **using** *obtain-fresh-z* **by** *meson*

moreover then obtain x' s' **where** *Some* (*AS-branch dc* x' s') = *lookup-branch dc cs* **using** *v teq check-caseI check-css-lookup-branch-exist* **by** *metis*
ultimately show *?case* **using** *reduce-caseI v check-caseI dc-of*.*cases* **by** *metis*
next
case (*check-assertI* x Θ Φ $\mathcal{B}$ Γ Δ c τ s)
hence *sps*: *supp* $s \subseteq$ *atom* ‘ *fst* ‘ *setD* Δ **by** *auto*
have *atom* $x \sharp c$ **using** *check-assertI* **by** *auto*
have *atom* $x \sharp \Gamma$ **using** *check-assertI check-s-wf wfG-elims* **by** *metis*
have *sble* Θ $((x, B\text{-}bool, c) \#_\Gamma \Gamma)$ **proof** −
obtain i' **where** i': $i' \models \Gamma \ \wedge\ \Theta$; $\Gamma \vdash i'$ **using** *check-assertI sble-def* **by** *metis*
obtain i::*valuation* **where** i:$i = i'$ ($x \mapsto$ *SBool True*) **by** *auto*

have $i \models (x, B\text{-}bool, c) \#_\Gamma \Gamma$ **proof** −
have $i' \models c$ **using** *valid.simps* i' *check-assertI* **by** *metis*
hence $i \models c$ **using** *is-satis-weakening-x* i ‹*atom* $x \sharp c$› **by** *auto*
moreover have $i \models \Gamma$ **using** *is-satis-g-weakening-x* i' i *check-assertI* ‹*atom* $x \sharp \Gamma$› **by** *metis*
ultimately show *?thesis* **using** *is-satis-g.simps* i **by** *auto*
qed
moreover have Θ ; $((x, B\text{-}bool, c) \#_\Gamma \Gamma) \vdash i$ **proof**(*rule wfI-cons*)
show ‹ $i' \models \Gamma$ › **using** i' **by** *auto*
show ‹ Θ ; $\Gamma \vdash i'$› **using** i' **by** *auto*
show ‹$i = i'(x \mapsto$ *SBool True*)› **using** i **by** *auto*
show ‹ $\Theta \vdash$ *SBool True*: *B-bool*› **using** *wfRCV-BBoolI* **by** *auto*
show ‹*atom* $x \sharp \Gamma$› **using** *check-assertI check-s-wf wfG-elims* **by** *auto*
qed
ultimately show *?thesis* **using** *sble-def* **by** *auto*
qed
then consider $(\exists v.\ s = [v]^s) \mid (\exists \delta'\ a.\ \Phi \vdash \langle\ \delta\ ,\ s\ \rangle \longrightarrow \langle\ \delta'\ ,\ a\ \rangle)$ **using** *check-assertI sps* **by** *metis*
hence $(\exists \delta'\ a.\ \Phi \vdash \langle\ \delta\ ,$ *ASSERT c IN s* $\rangle \longrightarrow \langle\ \delta'\ ,\ a\ \rangle)$ **proof**(*cases*)
case *1*
then show *?thesis* **using** *reduce-assert1I* **by** *metis*
next
case *2*

then show *?thesis* **using** *reduce-assert2I* **by** *metis*
qed
thus *?case* **by** *auto*
qed

lemma *progress*:
fixes *s*::*s*
assumes Θ ; Φ ; $\Delta \vdash \langle\ \delta\ ,\ s\ \rangle \Leftarrow \tau$
shows $(\exists\, v.\ s = [v]^s) \vee (\exists\, \delta'\ s'.\ \Phi\ \vdash \langle\ \delta\ ,\ s\ \rangle \longrightarrow \langle\ \delta'\ ,\ s'\ \rangle)$
proof −
have Θ ; Φ ; $\{||\}$; *GNil* ; $\Delta\ \vdash s \Leftarrow \tau$ **and** $\Theta \vdash \delta \sim \Delta$
using *config-type-elims*[*OF assms*(*1*)] **by** *auto*+
moreover hence *supp s* $\subseteq$ *atom ' fst ' setD* Δ **using** *check-s-wf wfS-supp* **by** *fastforce*
moreover have *sble* Θ *GNil* **using** *sble-def wfI-def is-satis-g.simps* **by** *simp*
ultimately show *?thesis* **using** *progress-aux* **by** *blast*
qed

16.4 Safety

lemma *safety*:
assumes $\Phi\ \vdash \langle\ \delta\ ,\ s\ \rangle \longrightarrow^* \langle\ \delta'\ ,\ s'\ \rangle$ **and** Θ ; Φ ; $\Delta \vdash\ \langle\ \delta\ ,\ s\ \rangle \Leftarrow \tau$
shows $(\exists\, v.\ s' = [v]^s) \vee (\exists\, \delta''\ s''.\ \Phi\ \vdash \langle\ \delta'\ ,\ s'\ \rangle \longrightarrow \langle\ \delta''\ ,\ s''\ \rangle)$
using *preservation-many progress assms* **by** *meson*

unused-thms *Eisbach-Tools Nominal2 AList Nominal−Utils RCLogic−*

end
theory *Utils*
imports *Main Native-Word.Uint32*
begin

16.5 Conversion Utilities

fun *string-of-digit* :: *nat* ⇒ *string*
where
string-of-digit n =
(*if n* = *0 then ''0''*
else if n = *1 then ''1''*
else if n = *2 then ''2''*
else if n = *3 then ''3''*
else if n = *4 then ''4''*
else if n = *5 then ''5''*
else if n = *6 then ''6''*
else if n = *7 then ''7''*
else if n = *8 then ''8''*
else ''9'')

```
fun string-of-nat :: nat ⇒ string
where
  string-of-nat n = (if n < 10 then string-of-digit n else
    string-of-nat (n div 10) @ (string-of-digit (n mod 10)))

fun string-of-literal :: String.literal ⇒ string where
  string-of-literal s = String.explode s

fun string-to-literal :: string ⇒ String.literal where
  string-to-literal s = String.implode s

fun string-lit-concat :: String.literal list ⇒ String.literal where
 string-lit-concat s = List.foldr (+) s (STR '''')

definition string-of-int :: int ⇒ string
where
  string-of-int i = (if i < 0 then ''-'' @ string-of-nat (nat (- i)) else
    string-of-nat (nat i))

definition string-lit-of-int :: int ⇒ String.literal
where
  string-lit-of-int = String.implode ∘ string-of-int


definition string-lit-of-uint32 :: uint32 ⇒ String.literal
where
  string-lit-of-uint32 = String.implode ∘ string-of-int ∘ int-of-uint32

definition string-lit-of-integer :: integer ⇒ String.literal
where
  string-lit-of-integer = string-lit-of-int ∘ int-of-integer


definition string-lit-of-nat :: nat ⇒ String.literal where
  string-lit-of-nat = String.implode ∘ string-of-nat

fun string-map :: string ⇒ ('a ⇒ string) ⇒ 'a list ⇒ string where
   string-map delim f [] = ''''
|  string-map delim f [x] = f x
|  string-map delim f (x#xs) = f x @ delim @ (string-map delim f xs)

fun string-lit-map :: String.literal ⇒ ('a ⇒ String.literal) ⇒ 'a list ⇒ String.literal where
   string-lit-map delim f [] = STR ''''
|  string-lit-map delim f [x] = f x
|  string-lit-map delim f (x#xs) = f x + delim + (string-lit-map delim f xs)

fun unzip3 :: ('a * 'b * 'c) list ⇒ ('a list * 'b list * 'c list) where
  unzip3 [] = ([],[],[])
| unzip3 ((x,y,z)#xyz) = (let (xs,ys,zs) = unzip3 xyz in (x#xs, y#ys, z#zs))

fun unzip :: ('a * 'b ) list ⇒ ('a list * 'b list ) where
```

unzip [] = ([],[])
| *unzip* ((*x*,*y*)#*xy*) = (*let* (*xs*,*ys*) = *unzip xy in* (*x*#*xs*, *y*#*ys*))

end

theory *SailAST*
imports *Main Native-Word.Uint32*
begin

16.6 Sail AST

type-synonym *string* = *String.literal*

hide-const *id*

datatype *value* = *Val*

datatype *loop* = *While* | *Until*

type-synonym *'a annot* = *'a*

type-synonym *x* = *string* — identifier
type-synonym *ix* = *string* — infix identifier
datatype *id* = — Identifier
  id *x*
| *operator* *x* — remove infix status

datatype *kid* = — kinded IDs: Type, Int, and Order variables
  var *x*

datatype *base-effect* = — effect
  BE-rreg — read register
| *BE-wreg* — write register
| *BE-rmem* — read memory
| *BE-rmemt* — read memory and tag
| *BE-wmem* — write memory
| *BE-eamem* — signal effective address for writing memory
| *BE-exmem* — determine if a store-exclusive (ARM) is going to succeed
| *BE-wmv* — write memory, sending only value
| *BE-wmvt* — write memory, sending only value and tag
| *BE-barr* — memory barrier
| *BE-depend* — dynamic footprint
| *BE-undef* — undefined-instruction exception
| *BE-unspec* — unspecified values
| *BE-nondet* — nondeterminism, from *nondet*
| *BE-escape* — potential exception
| *BE-config* — configuration option

datatype *kind* = — base kind

```
  K-type    — kind of types
| K-int    — kind of natural number size expressions
| K-order    — kind of vector order specifications
| K-bool    — kind of constraints

datatype nexp =  — numeric expression, of kind Int
  Nexp-id  id   — abbreviation identifier
| Nexp-var  kid   — variable
| Nexp-constant  integer   — constant
| Nexp-app  id nexp list   — app
| Nexp-times  nexp nexp   — product
| Nexp-sum  nexp nexp   — sum
| Nexp-minus  nexp nexp   — subtraction
| Nexp-exp  nexp   — exponential
| Nexp-neg  nexp   — unary negation

datatype effect =
  Effect-set  base-effect list   — effect set

datatype kinded-id =  — optionally kind-annotated identifier
  KOpt-kind  kind kid   — kind-annotated variable

datatype order =  — vector order specifications, of kind Order
  Ord-var  kid   — variable
| Ord-inc    — increasing
| Ord-dec    — decreasing

datatype typ =  — type expressions, of kind Type
  Typ-internal-unknown
| Typ-id  id   — defined type
| Typ-var  kid   — type variable
| Typ-fn  typ list typ effect   — Function (first-order only)
| Typ-bidir  typ typ effect   — Mapping
| Typ-tup  typ list   — Tuple
| Typ-app  id typ-arg list   — type constructor application
| Typ-exist  kinded-id list n-constraint typ
and typ-arg =  — type constructor arguments of all kinds
  A-nexp  nexp
| A-typ  typ
| A-order  order
| A-bool  n-constraint
and n-constraint =  — constraint over kind Int
  NC-equal  nexp nexp
| NC-bounded-ge  nexp nexp
| NC-bounded-gt  nexp nexp
| NC-bounded-le  nexp nexp
| NC-bounded-lt  nexp nexp
| NC-not-equal  nexp nexp
| NC-set  kid integer list
| NC-or  n-constraint n-constraint
| NC-and  n-constraint n-constraint
| NC-app  id typ-arg list
```

| *NC-var kid*
| *NC-true*
| *NC-false*

datatype *typ-pat* = — type pattern
TP-wild
| *TP-var kid*
| *TP-app id typ-pat list*

datatype *lit* = — literal constant
L-unit
| *L-zero*
| *L-one*
| *L-true*
| *L-false*
| *L-num integer* — natural number constant
| *L-hex string* — bit vector constant, C-style
| *L-bin string* — bit vector constant, C-style
| *L-string string* — string constant
| *L-undef* — undefined-value constant
| *L-real string*

datatype *quant-item* = — kinded identifier or Int constraint
QI-id kinded-id — optionally kinded identifier
| *QI-constraint n-constraint* — constraint
| *QI-constant kinded-id list*

datatype *'a pat* = — pattern
P-lit (*annot-pat* : *'a annot*) *lit* — literal constant pattern
| *P-wild* (*annot-pat* : *'a annot*) — wildcard
| *P-or* (*annot-pat* : *'a annot*) *'a pat 'a pat* — pattern disjunction
| *P-not* (*annot-pat* : *'a annot*) *'a pat* — pattern negation
| *P-as* (*annot-pat* : *'a annot*) *'a pat id* — named pattern
| *P-typ* (*annot-pat* : *'a annot*) *typ 'a pat* — typed pattern
| *P-id* (*annot-pat* : *'a annot*) *id* — identifier
| *P-var* (*annot-pat* : *'a annot*) *'a pat typ-pat* — bind pattern to type variable
| *P-app* (*annot-pat* : *'a annot*) *id 'a pat list* — union constructor pattern
| *P-vector* (*annot-pat* : *'a annot*) *'a pat list* — vector pattern
| *P-vector-concat* (*annot-pat* : *'a annot*) *'a pat list* — concatenated vector pattern
| *P-tup* (*annot-pat* : *'a annot*) *'a pat list* — tuple pattern
| *P-list* (*annot-pat* : *'a annot*) *'a pat list* — list pattern
| *P-cons* (*annot-pat* : *'a annot*) *'a pat 'a pat* — Cons patterns
| *P-string-append* (*annot-pat* : *'a annot*) *'a pat list* — string append pattern, x ‘y

datatype *typquant* = — type quantifiers and constraints
TypQ-tq quant-item list
| *TypQ-no-forall* — empty

datatype *'a mpat* = — Mapping pattern. Mostly the same as normal patterns but only constructible parts
MP-lit 'a annot lit
| *MP-id 'a annot id*

| *MP-app 'a annot id 'a mpat list*
| *MP-vector 'a annot 'a mpat list*
| *MP-vector-concat 'a annot 'a mpat list*
| *MP-tup 'a annot 'a mpat list*
| *MP-list 'a annot 'a mpat list*
| *MP-cons 'a annot 'a mpat 'a mpat*
| *MP-string-append 'a annot 'a mpat list*
| *MP-typ 'a annot 'a mpat typ*
| *MP-as 'a annot 'a mpat id*

datatype *'a internal-loop-measure* = — internal syntax for an optional termination measure for a loop
Measure-none
| *Measure-some 'a exp*
and *'a exp* = — expression
E-block (*annot-e* : *'a annot*) *'a exp list* — sequential block
| *E-id* (*annot-e* : *'a annot*) *id* — identifier
| *E-lit* (*annot-e* : *'a annot*) *lit* — literal constant
| *E-cast* (*annot-e* : *'a annot*) *typ 'a exp* — cast
| *E-app* (*annot-e* : *'a annot*) *id 'a exp list* — function application
| *E-app-infix* (*annot-e* : *'a annot*) *'a exp id 'a exp* — infix function application
| *E-tuple* (*annot-e* : *'a annot*) *'a exp list* — tuple
| *E-if* (*annot-e* : *'a annot*) *'a exp 'a exp 'a exp* — conditional
| *E-loop* (*annot-e* : *'a annot*) *loop 'a internal-loop-measure 'a exp 'a exp*
| *E-for* (*annot-e* : *'a annot*) *id 'a exp 'a exp 'a exp order 'a exp* — for loop
| *E-vector* (*annot-e* : *'a annot*) *'a exp list* — vector (indexed from 0)
| *E-vector-access* (*annot-e* : *'a annot*) *'a exp 'a exp* — vector access
| *E-vector-subrange* (*annot-e* : *'a annot*) *'a exp 'a exp 'a exp* — subvector extraction
| *E-vector-update* (*annot-e* : *'a annot*) *'a exp 'a exp 'a exp* — vector functional update
| *E-vector-update-subrange* (*annot-e* : *'a annot*) *'a exp 'a exp 'a exp 'a exp* — vector subrange update, with vector
| *E-vector-append* (*annot-e* : *'a annot*) *'a exp 'a exp* — vector concatenation
| *E-list* (*annot-e* : *'a annot*) *'a exp list* — list
| *E-cons* (*annot-e* : *'a annot*) *'a exp 'a exp* — cons
| *E-record* (*annot-e* : *'a annot*) *'a fexp list* — struct
| *E-record-update* (*annot-e* : *'a annot*) *'a exp 'a fexp list* — functional update of struct
| *E-field* (*annot-e* : *'a annot*) *'a exp id* — field projection from struct
| *E-case* (*annot-e* : *'a annot*) *'a exp 'a pexp list* — pattern matching
| *E-let* (*annot-e* : *'a annot*) *'a letbind 'a exp* — let expression
| *E-assign* (*annot-e* : *'a annot*) *'a lexp 'a exp* — imperative assignment
| *E-sizeof* (*annot-e* : *'a annot*) *nexp* — the value of *nexp* at run time
| *E-return* (*annot-e* : *'a annot*) *'a exp* — return *'aexp* from current function
| *E-exit* (*annot-e* : *'a annot*) *'a exp* — halt all current execution
| *E-ref* (*annot-e* : *'a annot*) *id*
| *E-throw* (*annot-e* : *'a annot*) *'a exp*
| *E-try* (*annot-e* : *'a annot*) *'a exp 'a pexp list*
| *E-assert* (*annot-e* : *'a annot*) *'a exp 'a exp* — halt with error message *'aexp* when not *'aexp*. exp' is optional.
| *E-var* (*annot-e* : *'a annot*) *'a lexp 'a exp 'a exp* — This is an internal node for compilation that demonstrates the scope of a local mutable variable
| *E-internal-plet* (*annot-e* : *'a annot*) *'a pat 'a exp 'a exp* — This is an internal node, used to distinguised some introduced lets during processing from original ones
| *E-internal-return* (*annot-e* : *'a annot*) *'a exp* — For internal use to embed into monad definition

| *E-internal-value* (*annot-e* : *'a annot*) *value* — For internal use in interpreter to wrap pre-evaluated values when returning an action
| *E-constraint* (*annot-e* : *'a annot*) *n-constraint*
and *'a lexp* = — lvalue expression
LEXP-id (*annot-lexp* : *'a annot*) *id* — identifier
| *LEXP-deref* (*annot-lexp* : *'a annot*) *'a exp*
| *LEXP-memory* (*annot-lexp* : *'a annot*) *id 'a exp list* — memory or register write via function call
| *LEXP-cast* (*annot-lexp* : *'a annot*) *typ id*
| *LEXP-tup* (*annot-lexp* : *'a annot*) *'a lexp list* — multiple (non-memory) assignment
| *LEXP-vector-concat* (*annot-lexp* : *'a annot*) *'a lexp list* — vector concatenation L-exp
| *LEXP-vector* (*annot-lexp* : *'a annot*) *'a lexp 'a exp* — vector element
| *LEXP-vector-range* (*annot-lexp* : *'a annot*) *'a lexp 'a exp 'a exp* — subvector
| *LEXP-field* (*annot-lexp* : *'a annot*) *'a lexp id* — struct field
and *'a fexp* = — field expression
FE-Fexp 'a annot id 'a exp
and *'a pexp* = — pattern match
Pat-exp 'a annot 'a pat 'a exp
| *Pat-when 'a annot 'a pat 'a exp 'a exp*
and *'a letbind* = — let binding
LB-val (*annot-letbind* : *'a annot*) *'a pat 'a exp* — let, implicit type (*'apat* must be total)

datatype *index-range* = — index specification, for bitfields in register types
BF-single nexp — single index
| *BF-range nexp nexp* — index range
| *BF-concat index-range index-range* — concatenation of index ranges

datatype *type-union* = — type union constructors
Tu-ty-id typ id

datatype *typschm* = — type scheme
TypSchm-ts typquant typ

datatype *'a mpexp* =
MPat-pat 'a annot 'a mpat
| *MPat-when 'a annot 'a mpat 'a exp*

datatype *'a pexp-funcl* = *PEXP-funcl 'a pexp*
datatype *'a reg-id* =
RI-id 'a annot id

datatype *type-def-aux* = — type definition body
TD-abbrev id typquant typ-arg — type abbreviation
| *TD-record id typquant* (*typ∗id*) *list bool* — struct type definition
| *TD-variant id typquant type-union list bool* — tagged union type definition
| *TD-enum id id list bool* — enumeration type definition
| *TD-bitfield id typ* (*id∗index-range*) *list* — register mutable bitfield type definition

type-synonym *val-spec-aux* = *typschm ∗ id ∗* (*string => string option*) *∗ bool*
datatype *'a mapcl* = — mapping clause (bidirectional pattern-match)
MCL-bidir 'a annot 'a mpexp 'a mpexp
| *MCL-forwards 'a annot 'a mpexp 'a exp*
| *MCL-backwards 'a annot 'a mpexp 'a exp*

datatype *tannot-opt* = — optional type annotation for functions
  Typ-annot-opt-none
| *Typ-annot-opt-some typquant typ*

datatype *'a rec-opt* = — optional recursive annotation for functions
  Rec-nonrec — non-recursive
| *Rec-rec* — recursive without termination measure
| *Rec-measure 'a pat 'a exp* — recursive with termination measure

datatype *effect-opt* = — optional effect annotation for functions
  Effect-opt-none — no effect annotation
| *Effect-opt-effect effect*

datatype *'a funcl* = — function clause
  FCL-Funcl 'a annot id 'a pexp-funcl

datatype *'a alias-spec* = — register alias expression forms
  AL-subreg 'a annot 'a reg-id id
| *AL-bit 'a annot 'a reg-id 'a exp*
| *AL-slice 'a annot 'a reg-id 'a exp 'a exp*
| *AL-concat 'a annot 'a reg-id 'a reg-id*

datatype *type-def* =
  TD-aux type-def-aux

datatype *val-spec* =
  VS-aux val-spec-aux

datatype *'a mapdef* = — mapping definition (bidirectional pattern-match function)
  MD-mapping 'a annot id tannot-opt 'a mapcl list

datatype *'a fundef* = — function definition
  FD-function 'a annot 'a rec-opt tannot-opt effect-opt 'a funcl list

datatype *default-spec* = — default kinding or typing assumption
  DT-order order

datatype *'a dec-spec* = — register declarations
  DEC-reg 'a annot effect effect typ id
| *DEC-config 'a annot id typ 'a exp*
| *DEC-alias 'a annot id 'a alias-spec*
| *DEC-typ-alias 'a annot typ id 'a alias-spec*

datatype *'a scattered-def* = — scattered function and union type definitions
  SD-function 'a annot 'a rec-opt tannot-opt effect-opt id — scattered function definition header
| *SD-funcl 'a annot 'a funcl* — scattered function definition clause
| *SD-variant 'a annot id typquant* — scattered union definition header
| *SD-unioncl 'a annot id type-union* — scattered union definition member
| *SD-mapping 'a annot id tannot-opt*
| *SD-mapcl 'a annot id 'a mapcl*
| *SD-end 'a annot id* — scattered definition end

datatype *prec* =
  Infix
| *InfixL*
| *InfixR*

datatype *'a loop-measure* =
  Loop loop 'a exp

datatype *'a def* = — top-level definition
  DEF-type type-def — type definition
| *DEF-fundef 'a fundef* — function definition
| *DEF-mapdef 'a mapdef* — mapping definition
| *DEF-val 'a letbind* — value definition
| *DEF-spec val-spec* — top-level type constraint
| *DEF-fixity prec integer id* — fixity declaration
| *DEF-overload id id list* — operator overload specification
| *DEF-default default-spec* — default kind and type assumptions
| *DEF-scattered 'a scattered-def* — scattered function and type definition
| *DEF-measure id 'a pat 'a exp* — separate termination measure declaration
| *DEF-loop-measures id 'a loop-measure list* — separate termination measure declaration
| *DEF-reg-dec 'a dec-spec* — register declaration
| *DEF-internal-mutrec 'a fundef list* — internal representation of mutually recursive functions
| *DEF-pragma string string* — compiler directive

datatype *'a opt-default* = — optional default value for indexed vector expressions
  Def-val-empty 'a annot
| *Def-val-dec 'a annot 'a exp*

datatype *'a defs* = — definition sequence
  Defs 'a def list

end
theory *SailASTUtils*
  imports *SailAST*
begin

16.7 AST Utils

definition *unit-typ* **where**
  unit-typ ≡ ((*Typ-id* ((*id* (*STR ''unit''*)))))

definition *num-typ* **where**
  num-typ n ≡ ((*Typ-app* ((*id* (*STR ''atom''*)))
    [((*A-nexp* ((*Nexp-constant* *n*))))]))

definition *int-typ* **where**
  int-typ ≡ ((*Typ-id* ((*id* (*STR ''int''*)))))

fun *bool-typ* **where**
  bool-typ b = ((*Typ-app* ((*id* (*STR ''atom-bool''*)))

```
    [( (A-bool  ( (if b then NC-true  else NC-false )  ))  ) ] )  )

definition bool-all-typ where
  bool-all-typ = ( (Typ-id  ( (id  (STR ''bool'')) )) )

abbreviation nc-eq ≡ λn1 n2.  (NC-equal  n1 n2)

abbreviation nint ≡ λi.  (Nexp-constant  i)

abbreviation nc-pos ≡ λne.  (NC-bounded-ge  ne (nint 0))

abbreviation nc-true ≡  NC-true




abbreviation nexp-var where
  nexp-var s ≡  (Nexp-var  ( (var  s)))

abbreviation atom-typ-var where
  atom-typ-var s ≡  (Typ-app  ( (id  (STR ''atom'')) [  (A-nexp   ( (Nexp-var  ( (var  s))))) ] )

abbreviation nexp-const where
  nexp-const s ≡  (Nexp-constant  s)

abbreviation num-or where
 num-or n1 n2 ≡  (Typ-exist  [] ( (NC-or  (nc-eq  (nexp-var  (STR ''n'')) (nexp-const  1))
(nc-eq (nexp-var (STR ''n'')) (nexp-const 2)))) (atom-typ-var (STR ''n'') ))


fun collect-pat ::  'a pat ⇒  id list where
  collect-pat  (P-or t p1 p2)  =  collect-pat p1 @ collect-pat p2
| collect-pat  (P-not t p)  =  collect-pat p
| collect-pat  (P-tup t pats)  =  concat (map (collect-pat) pats)
| collect-pat  (P-as t pat i ) =  collect-pat pat @ [i]
| collect-pat  (P-typ t typ pat) = collect-pat pat
| collect-pat  (P-app t ff pats) =concat (map (collect-pat) pats)
| collect-pat  (P-vector t  pats) =  concat (map (collect-pat) pats)
| collect-pat  (P-vector-concat  t pats) = concat (map (collect-pat) pats)
| collect-pat  (P-list t pats) =  concat (map (collect-pat) pats)
| collect-pat  (P-cons t p1 p2) =  collect-pat p1 @ collect-pat p2
| collect-pat  (P-string-append t pats) = concat (map (collect-pat) pats)
| collect-pat  (P-id - i) = [i]
| collect-pat  p = []



fun collect-e ::  'a exp ⇒  id list and
    collect-pexp ::  'a pexp ⇒  id list and
```

```
    collect-lexp :: 'a lexp ⇒ id list and
    collect-letbind :: 'a letbind ⇒ id list

  where
 collect-pexp  (Pat-exp t p e) = collect-pat p @ collect-e e

| collect-e  (E-block t es) = (concat (map (collect-e ) es))
| collect-e  (E-if t e1 e2 e3) = collect-e  e1 @ collect-e  e2 @ collect-e  e3
| collect-e  (E-case t e1 pexps) =  concat (map (collect-pexp ) pexps)
| collect-e  (E-app t f es) =  concat (map (collect-e ) es)
| collect-e  (E-let t lb e) =  collect-letbind lb @ collect-e e
| collect-e  (E-assign t lexp e1) =  collect-lexp lexp @ collect-e e1
| collect-e  (E-id - i) = [i]
| collect-e (E-lit - - ) = []
| collect-e (E-tuple - es) =  concat (map (collect-e ) es)

| collect-lexp (LEXP-id - i) = [i]
| collect-lexp (LEXP-cast - - i) = [i]

| collect-letbind (LB-val - pat exp) = collect-pat pat @ collect-e exp

fun collect-pexp-funcl-ids :: 'a pexp-funcl ⇒ id list where
   collect-pexp-funcl-ids (PEXP-funcl pexp) = collect-pexp pexp

fun collect-ids :: 'a funcl ⇒ id list where
  collect-ids (FCL-Funcl - - pf) = remdups (collect-pexp-funcl-ids pf)

end
theory SailEnv
  imports SailASTUtils
begin
```

16.8 Sail Environment

```
datatype mut = Mutable | Immutable

record env =
        top-val-specs :: (id  * (typquant * typ)) list
        typ-vars :: (kid * kind) list
        locals :: (id * (mut * typ)) list
        records :: (id * ( typquant * (( id * typ) list) )) list
        variants :: (id * (typquant * (type-union list))) list
        constraints :: n-constraint list
        default-order :: order option
        ret-typ :: typ option
        registers :: (id * typ) list
        typ-synonyms :: (id * (typquant * typ-arg)) list
        enums :: (id * (id list)) list
        prover :: ((n-constraint list) ⇒ n-constraint ⇒ bool) option

definition emptyEnv :: env where
  emptyEnv = (| top-val-specs = [],  typ-vars = [] , locals = [], records=[],variants=[],constraints=[],default-order=None,
```

```
ret-typ=None,
    registers=[], typ-synonyms=[] , enums=[], prover = None⦈


record tannot' =
  tannot-env :: env
  tannot-typ :: typ
  tannot-effect :: effect
  tannot-expected :: typ option
  tannot-instantiations :: ((kid*typ-arg) list) option

type-synonym tannot = tannot' option


definition bind' where
  bind' f t = Option.bind t (Some ∘ f)
```

16.9 Unpacking tannot

```
definition get-type :: tannot ⇒ typ option  where
  get-type  = bind' tannot-typ

definition get-env :: tannot ⇒ env option where
  get-env  = bind' tannot-env

primrec get :: tannot ⇒ (env*typ) option where
   get None = None
 | get (Some tan) = Some (tannot-env tan, tannot-typ tan)

fun get-e :: tannot exp ⇒ (env*typ) option where
  get-e exp  = get (annot-e exp)

fun get-tan-e :: tannot exp ⇒ tannot where
  get-tan-e exp = annot-e exp

fun get-tan-pat :: tannot pat ⇒ tannot where
  get-tan-pat exp = annot-pat exp

fun get-tan-lexp :: tannot lexp ⇒ tannot where
  get-tan-lexp lexp = annot-lexp lexp

fun get-tan-letbind :: tannot letbind ⇒ tannot where
  get-tan-letbind lexp = annot-letbind lexp

fun type-of-exp :: tannot exp ⇒ typ option where
  type-of-exp exp = get-type (get-tan-e exp)

fun type-of-pat :: tannot pat ⇒ typ option where
  type-of-pat pat  = get-type (get-tan-pat pat)

fun type-of-pexp :: tannot pexp ⇒ typ option where
  type-of-pexp (Pat-exp - pat exp) = type-of-pat pat
```

```
| type-of-pexp (Pat-when - pat - exp) = type-of-pat pat

fun get-env-exp :: tannot exp ⇒ env option where
  get-env-exp exp = get-env (get-tan-e exp)

fun get-env-letbind :: tannot letbind ⇒ env option where
  get-env-letbind lexp  = get-env (get-tan-letbind lexp)

definition ret-type :: tannot ⇒ typ option where
  ret-type t = Option.bind (Option.bind t (Some ∘ tannot-env) ) ret-typ

primrec set-type :: tannot ⇒ typ ⇒ tannot where
  set-type (Some t) typ = Some (t (| tannot-typ := typ |))
| set-type None typ = Some (| tannot-env = emptyEnv ,   tannot-typ = typ, tannot-effect = ( (Effect-set
[])), tannot-expected = None, tannot-instantiations = None  |)


fun type-of-lexp :: tannot lexp ⇒ typ option where
  type-of-lexp lexp = get-type (get-tan-lexp lexp)
```

16.10 Generic Lookup

```
fun lookup :: ('a*'b) list ⇒ 'a ⇒ 'b option where
  lookup [] - = None
| lookup ((x,t)#xs) y = (if x = y then Some t else lookup xs y)
```

16.11 Enumerations

```
fun lookup-enum-env :: env ⇒ id ⇒ typ option where
lookup-enum-env env x = (case find (λ(-,es). List.member es x ) (enums env) of
                           None ⇒ None | Some (tyid,-) ⇒ Some ( (Typ-id tyid)))


primrec lookup-enum :: tannot ⇒ id ⇒ typ option where
  lookup-enum (Some tan) x = lookup-enum-env (tannot-env tan) x
| lookup-enum None - = None
```

16.12 Constraints

```
fun add-constraint :: n-constraint ⇒ env ⇒ env where
  add-constraint nc env = env (| constraints := nc # (constraints env) |)
```

16.13 Records

```
fun lookup-record-field-env :: env ⇒ id ⇒ id ⇒ typ option where
  lookup-record-field-env env recc-id field-id = (case lookup (records env) recc-id of
                                                    Some (-, fieldds) ⇒ lookup fieldds field-id | None ⇒ None)

fun lookup-record-field :: tannot ⇒ id ⇒ id ⇒ typ option where
```

```
lookup-record-field (Some tan) x y = lookup-record-field-env (tannot-env tan) x y
| lookup-record-field None x y = None


fun deconstruct-record-type :: typ ⇒ id option where
  deconstruct-record-type ( (Typ-id recid) ) = Some recid
| deconstruct-record-type ( (Typ-app recid - ) ) = Some recid
| deconstruct-record-type - = None
```

16.14 Variants

```
fun lookup-tud :: type-union list ⇒ id ⇒ typ option where
  lookup-tud [] - = None
| lookup-tud (( (Tu-ty-id typ ( (id x) ))) # tus) ( (id y)  ) = (if x = y then Some typ else lookup-tud
tus ( (id y) ))
| lookup-tud ( - # tus ) - = None

fun lookup-variant-env :: env ⇒ id ⇒ id ⇒ typ option where
lookup-variant-env env tid vid = (case lookup (variants env) tid of
                                  Some (- , tus) ⇒ lookup-tud tus vid | None ⇒ None)

fun lookup-variant :: tannot ⇒ id ⇒ id ⇒ typ option where
 lookup-variant (Some tan) x y = lookup-variant-env (tannot-env tan) x y
| lookup-variant None - - = None
```

16.15 Val Specs

```
fun get-val-spec-env :: env ⇒ id ⇒ (typquant * typ) option where
  get-val-spec-env env x = lookup (top-val-specs env) x

fun get-val-spec :: tannot ⇒ id ⇒ (typquant * typ) option where
 get-val-spec (Some tan) x = get-val-spec-env (tannot-env tan) x
| get-val-spec None - = None

fun lookup-fun :: tannot ⇒ id ⇒ (typ list * typ) option where
lookup-fun (Some tan) fid = (case get-val-spec-env (tannot-env tan) fid of
                               Some (- ,  (Typ-fn in-typs rett-typ - )  ) ⇒ Some (in-typs, rett-typ)
                             | - ⇒ None)
| lookup-fun None - = None
```

16.16 Registers

```
fun lookup-register-env :: env ⇒ id ⇒ typ option where
lookup-register-env env x =  (case lookup (registers env) x of
                               Some t ⇒ Some t | None ⇒ None)

fun lookup-index :: (id * 'a) list ⇒ id ⇒ nat option where
  lookup-index [] - = None
| lookup-index (x#xs) y = (if fst x = y then Some (List.length xs)
                          else lookup-index xs y)
```

```
fun lookup-register-index-env :: env ⇒ id ⇒ nat option where
  lookup-register-index-env env x = (case lookup-index (registers env) x of
                                Some t ⇒ Some t | None ⇒ None)

fun lookup-register :: tannot ⇒ id ⇒ typ option where
lookup-register (Some tan) x = lookup-register-env (tannot-env tan) x
| lookup-register None - = None
```

16.17 Local Identifiers

```
fun lookup-local-id-env :: env ⇒ id ⇒ typ option where
  lookup-local-id-env env x = (case (lookup (locals env) x) of
                                Some (-,t) ⇒ Some t | None ⇒ lookup-enum-env env x)

primrec lookup-local-id :: tannot ⇒ id  ⇒ typ option where
  lookup-local-id (Some tan) x = lookup-local-id-env (tannot-env tan) x
| lookup-local-id None x = None

fun lookup-mutable-env :: env ⇒ id ⇒ typ option where
  lookup-mutable-env env x = (case (lookup (locals env) x) of
                                Some (Mutable,t) ⇒ Some t | None ⇒ lookup-register-env env x)

fun lookup-mutable :: tannot ⇒ id ⇒ typ option where
  lookup-mutable (Some tan) x = lookup-mutable-env (tannot-env tan) x
| lookup-mutable None x = None

fun add-local-env :: env ⇒ id ⇒ typ ⇒ env where
  add-local-env env x typ = env (| locals := (x,(Immutable,typ)) # (locals env) |)

fun add-local :: tannot ⇒ id ⇒ typ ⇒ tannot where
  add-local (Some tan) x typ = Some (tan  (| tannot-env := add-local-env (tannot-env tan) x typ |))
| add-local None - - = None



fun lookup-id :: tannot ⇒ id ⇒ typ option where
 lookup-id t x = (case lookup-local-id t x of
                   Some typ ⇒ Some typ
                 | None ⇒ lookup-register t x)


fun deconstruct-register-type :: typ ⇒ typ option where
deconstruct-register-type t = (case  t of
                                ( (Typ-app ( (id ( app-id ))) [ ( (A-typ typ )) ])) ⇒
                                  (if app-id = STR ''register'' then Some typ else None) | - ⇒ None)
```

16.18 Vectors

```
fun deconstruct-bitvector-type ::  typ ⇒ (nexp *order*typ) option where
deconstruct-bitvector-type t = (case  t of
                                ( (Typ-app ( (id ( app-id ))) [ ( (A-nexp len)),  (A-order ord) ])) ⇒
```

(*if app-id = STR ''bitvector'' then Some (len,ord, (Typ-id ((id (STR ''bit'')))))) else None*) | - ⇒ *None*)

fun *is-bitvector-type* :: *tannot* ⇒ (*nexp* ∗*order*∗*typ*) *option* **where**
is-bitvector-type None = None
| *is-bitvector-type* (*Some t*) = *deconstruct-bitvector-type* (*tannot-typ t*)

fun *deconstruct-vector-type* :: *typ* ⇒ (*nexp* ∗*order*∗*typ*) *option* **where**
deconstruct-vector-type t = (*case t of*
((*Typ-app* ((*id* (*app-id*))) [((*A-nexp len*)), (*A-order ord*), (*A-typ typ*)])) ⇒
(*if app-id = STR ''vector'' then Some* (*len*,*ord*,*typ*) *else None*) | - ⇒ *deconstruct-bitvector-type t*)

fun *is-vector-type* :: *tannot* ⇒ (*nexp* ∗*order*∗*typ*) *option* **where**
is-vector-type None = None
| *is-vector-type* (*Some t*) = *deconstruct-vector-type* (*tannot-typ t*)

fun *is-list-type* :: *tannot* ⇒ *typ option* **where**
is-list-type None = None
| *is-list-type* (*Some t*) = (*case tannot-typ t of*
((*Typ-app* ((*id* (*app-id*))) [((*A-typ typ*))])) ⇒ (*if app-id = STR ''list'' then Some typ else None*) | - ⇒ *None*)

fun *deconstruct-bool-type* :: *typ* ⇒ *n-constraint option* **where**
deconstruct-bool-type ((*Typ-id* ((*id b*)))) = (*if b = STR ''bool'' then Some nc-true else None*)
| *deconstruct-bool-type* ((*Typ-app* ((*id b*)) [(*A-bool nc*)])) = (*if b = STR ''atom-bool'' then Some nc else None*)
| *deconstruct-bool-type* ((*Typ-exist kids nc typ*)) = *deconstruct-bool-type typ*
| *deconstruct-bool-type Typ-internal-unknown* = *None*
| *deconstruct-bool-type* (*Typ-id* (*operator va*)) = *None*
| *deconstruct-bool-type* (*Typ-var v*) = *None*
| *deconstruct-bool-type* (*Typ-fn v va vb*) = *None*
| *deconstruct-bool-type* (*Typ-bidir v va vb*) = *None*
| *deconstruct-bool-type* (*Typ-tup v*) = *None*
| *deconstruct-bool-type* (*Typ-app* (*operator vb*) *va*) = *None*
| *deconstruct-bool-type* (*Typ-app* (*id va*) -) = *None*

fun *prove* :: *env* ⇒ *n-constraint* ⇒ *bool* **where**
prove env nc = (*case* (*prover env*) *of*
Some p ⇒ *p* (*constraints env*) *nc* | *None* ⇒ *True*)

16.19 Substitution in Types

fun *subst-nexp* :: (*kid∗typ-arg*) ⇒ *nexp* ⇒ *nexp* **where**

subst-nexp ks (*Nexp-id x*) = *Nexp-id x*
| *subst-nexp* (*k1*, (*A-nexp* (*ne*))) (*Nexp-var k2*) = (*if k1* = *k2 then ne else* (*Nexp-var k2*))
| *subst-nexp* (*k1*, (*A-bool* -)) (*Nexp-var k2*) = ((*Nexp-var k2*))
| *subst-nexp* (*k1*, (*A-typ* -)) (*Nexp-var k2*) = ((*Nexp-var k2*))
| *subst-nexp* (*k1*, (*A-order* -)) (*Nexp-var k2*) = ((*Nexp-var k2*))

| *subst-nexp ks* (*Nexp-constant n*) = *Nexp-constant n* — constant
| *subst-nexp ks* (*Nexp-app x nes*) = *Nexp-app x* (*List.map* (*subst-nexp ks*) *nes*) — app
| *subst-nexp ks* (*Nexp-times n1 n2*) = *Nexp-times* (*subst-nexp ks n1*) (*subst-nexp ks n1*)
| *subst-nexp ks* (*Nexp-sum n1 n2*) = *Nexp-sum* (*subst-nexp ks n1*) (*subst-nexp ks n1*)
| *subst-nexp ks* (*Nexp-minus n1 n2*) = *Nexp-minus* (*subst-nexp ks n1*) (*subst-nexp ks n1*)
| *subst-nexp ks* (*Nexp-exp n1*) = *Nexp-exp* (*subst-nexp ks n1*)
| *subst-nexp ks* (*Nexp-neg n1*) = *Nexp-neg* (*subst-nexp ks n1*)

fun *subst-order* :: (*kid∗typ-arg*) ⇒ *order* ⇒ *order*
where

subst-order (*k1*, (*A-order* (*ord*))) (*Ord-var k2*) = (*if k1* = *k2 then ord else* (*Ord-var k2*))
| *subst-order* (*k1*, -) (*Ord-var k2*) = (*Ord-var k2*)
| *subst-order* - *Ord-inc* = *Ord-inc*
| *subst-order* - *Ord-dec* = *Ord-dec*

fun *subst-typ* :: (*kid∗typ-arg*) ⇒ *typ* ⇒ *typ* **and**

subst-typ-arg :: (*kid∗typ-arg*) ⇒ *typ-arg* ⇒ *typ-arg* **and**

subst-nc :: (*kid∗typ-arg*) ⇒ *n-constraint* ⇒ *n-constraint*

where

subst-nc ks (*NC-equal ne1 ne2*) = *NC-equal* (*subst-nexp ks ne1*) (*subst-nexp ks ne1*)
| *subst-nc ks* (*NC-bounded-ge ne1 ne2*) = *NC-bounded-ge* (*subst-nexp ks ne1*) (*subst-nexp ks ne1*)
| *subst-nc ks* (*NC-bounded-gt ne1 ne2*) = *NC-bounded-gt* (*subst-nexp ks ne1*) (*subst-nexp ks ne1*)
| *subst-nc ks* (*NC-bounded-le ne1 ne2*) = *NC-bounded-le* (*subst-nexp ks ne1*) (*subst-nexp ks ne1*)
| *subst-nc ks* (*NC-bounded-lt ne1 ne2*) = *NC-bounded-lt* (*subst-nexp ks ne1*) (*subst-nexp ks ne1*)
| *subst-nc ks* (*NC-not-equal ne1 ne2*) = *NC-not-equal* (*subst-nexp ks ne1*) (*subst-nexp ks ne1*)
| *subst-nc ks* (*NC-set k is*) = *NC-set k is*
| *subst-nc ks* (*NC-or nc1 nc2*) = *NC-or* (*subst-nc ks nc1*) (*subst-nc ks nc2*)
| *subst-nc ks* (*NC-and nc1 nc2*) = *NC-and* (*subst-nc ks nc1*) (*subst-nc ks nc2*)
| *subst-nc ks* (*NC-app tid args*) = *NC-app tid* (*List.map* (*subst-typ-arg ks*) *args*)
| *subst-nc* (*k1*, (*A-bool* (*nc*))) (*NC-var k2*) = (*if k1* = *k2 then nc else* (*NC-var k2*))
| *subst-nc* (*k1*, (*A-nexp* -)) (*NC-var k2*) = ((*NC-var k2*))
| *subst-nc* (*k1*, (*A-typ* -)) (*NC-var k2*) = ((*NC-var k2*))
| *subst-nc* (*k1*, (*A-order* -)) (*NC-var k2*) = ((*NC-var k2*))

```
| subst-nc ks NC-true   = NC-true
| subst-nc ks  NC-false  = NC-false



 | subst-typ - Typ-internal-unknown   = Typ-internal-unknown
 | subst-typ - (Typ-id tyid)   =   (Typ-id tyid)
 | subst-typ (k1, (A-typ ( t))) (Typ-var k2) = (if k1=k2 then t else (Typ-var k2))
 | subst-typ (k1, (A-bool - )) (Typ-var k2) = Typ-var k2
 | subst-typ (k1, (A-order - )) (Typ-var k2) = Typ-var k2
 | subst-typ (k1, (A-nexp - )) (Typ-var k2) = (Typ-var k2)
 | subst-typ ks (Typ-fn ts tr e) = Typ-fn (List.map (subst-typ ks) ts) (subst-typ ks tr) e
 | subst-typ ks (Typ-bidir t1 t2 e) = Typ-bidir (subst-typ ks t1) (subst-typ ks t2) e
 | subst-typ ks (Typ-tup ts) =  Typ-tup (List.map (subst-typ ks) ts)
 | subst-typ ks (Typ-app tyid tas) = Typ-app tyid (List.map (subst-typ-arg ks) tas)
 | subst-typ ks (Typ-exist kids nc typ) = Typ-exist kids (subst-nc ks nc) (subst-typ ks typ)


| subst-typ-arg ks (A-nexp ne) = A-nexp (subst-nexp ks ne)
| subst-typ-arg ks (A-typ ne) = A-typ (subst-typ ks ne)
| subst-typ-arg ks (A-order ne) = A-order (subst-order ks ne)
| subst-typ-arg ks (A-bool nc) = A-bool (subst-nc ks nc)


fun subst-inst :: tannot ⇒ typ ⇒ typ option where
subst-inst None t = Some t
| subst-inst (Some t') t = (case tannot-instantiations t' of
                 None ⇒ Some t
               | Some is ⇒ Some (List.fold subst-typ  is t ))



fun subst-inst-list :: tannot ⇒ typ list ⇒ typ list option where
subst-inst-list tan typs = those (List.map (subst-inst tan) typs)
```

16.20 Constructors

```
definition pat-id where
  pat-id x = P-id (set-type None unit-typ) (id x)

definition pat-pair where
  pat-pair = P-tup (set-type None (Typ-tup [unit-typ,unit-typ])) [ pat-id (STR ''x'') , pat-id (STR ''y'')]

definition pat-unit where
  pat-unit  = P-lit (set-type None unit-typ) (SailAST.L-unit)

definition e-unit where
  e-unit =  (E-lit (set-type None unit-typ) ( SailAST.L-unit ))

definition e-let where
  e-let e t = (E-let (set-type None unit-typ)
                ( (LB-val None ( (P-id (set-type None t) ( (id (STR ''x'')) )) )) ))
```

e) *e-unit*)

definition *e-pair* **where**
e-pair = *E-tuple* (*set-type None* (*Typ-tup* [*unit-typ*,*unit-typ*])) [*e-unit*, *e-unit*]

definition *e-true* **where**
e-true = (*E-lit* (*set-type None* (*bool-typ True*)) (*SailAST.L-true*))

definition *bv-typ* **where**
bv-typ n = (*Typ-app* (*id* (*STR ''vector''*)) [*A-typ* (*Typ-id* (*id* (*STR ''bit''*))) , *A-order Ord-dec*, *A-nexp* (*Nexp-constant n*)])

definition *bv-typ2* **where**
bv-typ2 n = (*Typ-app* (*id* (*STR ''bitvector''*)) [*A-nexp* (*Nexp-constant n*) , *A-order Ord-dec*])

definition *bv-lit* **where**
bv-lit bs = (*E-lit* (*set-type None* (*bv-typ* (*integer-of-nat* (*List.length* (*String.explode bs*))))) (*SailAST.L-bin bs*))

definition *int-tannot* **where**
int-tannot = *set-type None int-typ*

definition *pat-lit-bv* **where**
pat-lit-bv bs = *P-lit* (*set-type None* (*bv-typ* (*integer-of-nat* (*List.length* (*String.explode bs*))))) (*SailAST.L-bin bs*)

definition *e-false* **where**
e-false = (*E-lit* (*set-type None* (*bool-typ False*)) (*SailAST.L-false*))

abbreviation *unk* ≡ (*None*)

end
theory *ShowAST*

imports *SailEnv SailASTUtils Show.Show Show.Show-Instances*
begin

16.21 Show AST Setup

16.21.1 Integer

definition *showsp-integer* :: *integer showsp* **where**
showsp-integer p n = *showsp-int p* (*int-of-integer n*)

lemma *show-law-integer* [*show-law-intros*]:
show-law showsp-integer r
by (*rule show-lawI*) (*simp add*: *showsp-integer-def show-law-simps*)

lemma *showsp-integer-append* [*show-law-simps*]:
showsp-integer p r (*x* @ *y*) = *showsp-integer p r x* @ *y*

by (*intro show-lawD show-law-intros*)

local-setup {∗
Show-Generator.register-foreign-showsp @{*typ integer*} @{*term showsp-integer*} @{*thm show-law-integer*}
∗}

16.21.2 Sting Literal

definition *showsp-literal* :: *String.literal showsp* **where**
showsp-literal p n = *shows-string* (*String.explode n*)

lemma *show-law-literal* [*show-law-intros*]:
show-law showsp-literal r
by (*rule show-lawI*) (*simp add*: *showsp-literal-def show-law-simps*)

lemma *showsp-literal-append* [*show-law-simps*]:
showsp-literal p r (*x* @ *y*) = *showsp-literal p r x* @ *y*
by (*intro show-lawD show-law-intros*)

local-setup {∗
Show-Generator.register-foreign-showsp @{*typ String.literal*} @{*term showsp-literal*} @{*thm show-law-literal*}
∗}

derive *show id kid kind kinded-id order loop*

derive *show nexp*

derive *show base-effect*

derive *show effect*

derive *show n-constraint*
derive *show typ typ-arg*
derive *show typ-pat*

derive *show lit*
derive *show pat*
derive *show value*
derive *show exp*
derive *show quant-item typquant*
derive *show type-union index-range*
derive *show mut*

fun *show-env* :: *env* ⇒ *char list* **where**
show-env env = ″*Locals*: ″ @ *List.concat* (*List.map* (λ(*i*,*t*). *show i* @ ″::″ @ (*show t*) @ ″ ″) (*locals env*)) @ ″↩″ @
″*Typvars*: ″ @ *List.concat* (*List.map* (λ(*i*,*t*). *show i* @ ″::″ @ (*show t*) @ ″ ″) (*typ-vars env*)) @ ″↩″ @
″*Val specs*: ″ @ *List.concat* (*List.map* (λ(*i*,*t*). *show i* @ ″::″ @ (*show t*) @ ″ ″) (*top-val-specs*

env)) @ ''⏎'' @
''*Variants*: '' @ *List.concat* (*List.map* (λ(*i*,*t*). *show i* @ ''::'' @ (*show t*) @ '' '') (*variants env*)) @ ''⏎'' @
''*Records*: '' @ *List.concat* (*List.map* (λ(*i*,*t*). *show i* @ ''::'' @ (*show t*) @ '' '') (*records env*)) @ ''⏎'' @
''*Typsyn*: '' @ *List.concat* (*List.map* (λ(*i*,(*tq*,*ta*)). *show i* @ ''::'' @ (*show tq*) @ (*show ta*) @ '' '') (*typ-synonyms env*))

fun *show-tannot* :: *tannot* ⇒ *char list* **where**
show-tannot None = ''*tannot* = (*None*)''
| *show-tannot* (*Some t*) = ''*Inst*: '' @ (*case tannot-instantiations t of*
Some ts ⇒ *List.concat* (*List.map* (λ(*i*,*t*). (*show i*) @ '' = '' @ (*show t*) @ '' '') *ts*)
| *None* ⇒ ''(*none*)'') @ ''⏎'' @
''*Typ*: '' @ (*show* (*tannot-typ t*))

value *show* (*L-num 10*)

value *show unit-typ*

end
theory *MiniSailAST*
imports *Main HOL−Library.FSet*
begin

16.22 MiniSail AST

type-synonym *num-nat* = *nat*

datatype *sort* = *Sort string string list*

datatype *x* = *Atom sort nat*
datatype *u* = *Atom sort nat*
datatype *bv* = *Atom sort nat*

type-synonym *f* = *string*
type-synonym *dc* = *string*
type-synonym *tyid* = *string*

Base types

datatype *b* =
B-int
| *B-bool*
| *B-id tyid*
| *B-pair b b* ([- , -]^b)
| *B-unit*

| *B-bitvec*
| *B-var bv*
| *B-app tyid b*

datatype *bit* = *BitOne* | *BitZero*

Literals

datatype *l* =
L-num int
| *L-true*
| *L-false*
| *L-unit*
| *L-bitvec bit list*

Values

datatype *v* =
V-lit l ([-]^{v})
| *V-var x* ([-]^{v})
| *V-pair v v* ([- , -]^{v})
| *V-cons tyid dc v*
| *V-consp tyid dc b v*

Binary Operations

datatype *opp* = *Plus* (*plus*) | *LEq* (*leq*)

Expressions

datatype *e* =
AE-val v ([-]^{e})
| *AE-app f v* ([- (-)]^{e})
| *AE-appP f b v* ([- [-] (-)]^{e})
| *AE-op opp v v* ([- - -]^{e})
| *AE-concat v v* ([- @@ -]^{e})
| *AE-fst v* ([#*1*-]^{e})
| *AE-snd v* ([#*2*-]^{e})
| *AE-mvar u* ([-]^{e})
| *AE-len v* ([| - |]^{e})
| *AE-split v v*

Expressions for Constraints

datatype *ce* =
CE-val v ([-]^{ce})
| *CE-op opp ce ce* ([- - -]^{ce})
| *CE-concat ce ce* ([- @@ -]^{ce})
| *CE-fst ce* ([#*1*-]^{ce})
| *CE-snd ce* ([#*2*-]^{ce})
| *CE-len ce* ([| - |]^{ce})

Constraints

datatype *c* =
C-true (*TRUE* [] *50*)
| *C-false* (*FALSE* [] *50*)

```
| C-conj c c  (-  AND  -  [50, 50] 50)
| C-disj c c  (- OR -  [50,50] 50)
| C-not c       ( ¬ -  [] 50 )
| C-imp c c   (-  IMP  -  [50, 50] 50)
| C-eq ce ce  (-  ==  -  [50, 50] 50)
```

Refined type

```
datatype τ =
   T-refined-type  x b c    ({| - : -  | - |} [50, 50] 1000)
```

Statements

```
datatype
 s =
  AS-val v                              ( [-]^s)
| AS-let x  e s      ( (LET - = - IN -))
| AS-let2 x τ  s s ( (LET - : - = - IN -))
| AS-if v s s                          ( (IF - THEN - ELSE -) [0, 61, 0] 61)
| AS-var u τ v s   ( (VAR - : - = - IN -))
| AS-assign u  v                        ( (- ::= -))
| AS-match v branch-list                ( (MATCH - WITH { - }))
| AS-while s s                          ( (WHILE - DO { - } ) [0, 0] 61)
| AS-seq s s                           ( ( - ;; - )  [1000, 61] 61)
| AS-assert c s                         ( (ASSERT - IN - ) )
and branch-s =
  AS-branch dc x  s  ( ( - - ⇒ - ))
and branch-list =
  AS-final  branch-s                     ( { - } )
| AS-cons  branch-s branch-list         ( ( - | -  ))
```

```
term LET x = [plus [x]^v  [x]^v]^e  IN  [[x]^v]^s
```

Function and union type definitions

```
datatype fun-typ  =
     AF-fun-typ x b c τ s
```

```
datatype fun-typ-q =
     AF-fun-typ-some bv fun-typ
   | AF-fun-typ-none fun-typ
```

```
datatype fun-def =
     AF-fundef f fun-typ-q
```

```
datatype type-def =
    AF-typedef string (string ∗ τ) list
    |  AF-typedef-poly string bv (string ∗ τ) list
```

```
datatype var-def =
    AV-def u τ v
```

Programs

datatype p =
  AP-prog type-def list fun-def list var-def list s

16.23 Contexts

type-synonym Φ = *fun-def list*
type-synonym Θ = *type-def list*
type-synonym $\mathcal{B}$ = *bv fset*

datatype Δ =
  DNil ($[]_\Delta$)
  | *DCons u∗τ* Δ (**infixr** $\#_\Delta$ *65*)

datatype Γ =
  GNil
  | *GCons x∗b∗c* Γ (**infixr** $\#_\Gamma$ *65*)

fun *append-g* :: $\Gamma \Rightarrow \Gamma \Rightarrow \Gamma$ (**infixr** @ *65*) **where**
  append-g GNil g = *g*
| *append-g* (*xbc* $\#_\Gamma$ *g1*) *g2* = (*xbc* $\#_\Gamma$ (*g1*@*g2*))

end
theory *SailToMs*
imports *Utils SailEnv SailASTUtils ShowAST MiniSailAST HOL−Library.Debug*
begin

Chapter 17

Convert from Sail to MiniSail

type-synonym *xa=x*
type-synonym *ea=e*
type-synonym *va=v*
type-synonym *sa=s*
type-synonym *la=MiniSailAST.l*
type-synonym *ba=b*
type-synonym *ca=MiniSailAST.c*

fun *trace* :: *char list* ⇒ *bool* **where**
trace s = (*let* - = *Debug.trace* (*String.implode s*) *in True*)

17.1 Variables

fun *string-to-int* :: *string* ⇒ *int* **where**
string-to-int s = (*foldr* (λ*x y*. *y*∗*256* + *of-char x*) (*String.explode s*)) *0*

fun *string-to-nat* :: *string* ⇒ *nat* **where**
string-to-nat s = *nat* (*string-to-int s*)

value *String.explode* (*STR* ''*xyz*'')

value *string-to-int* (*STR* ''*xyxxa*'')

fun *mk-x* :: *nat* ⇒ *x* **where**
mk-x n = *x.Atom* (*Sort* ''*x*'' []) *n*

fun *mk-u* :: *nat* ⇒ *u* **where**
mk-u n = *u.Atom* (*Sort* ''*u*'' []) *n*

fun *mk-fresh-x* :: *nat* ⇒ *nat* ∗ *xa* **where**

mk-fresh-x ii = (*ii+1*, *mk-x* ((*ii+1*)∗*2*))

fun *index-for* :: (*'a∗'b*) *list* ⇒ *'a* ⇒ *nat option* **where**
  index-for [] - = *None*
| *index-for* ((*x*,-)#*xs*) *y* = (*if x* = *y then Some* (*length xs* + *1*) *else index-for xs y*)

fun *mk-prog-x* :: *env* ⇒ *id* ⇒ *x option* **where**
 mk-prog-x env ((*id x*)) = *Option.bind* (*index-for* (*locals env*) ((*id x*))) (*Some* ∘ (λ*ii* . *mk-x* ((*ii+1*)∗*2+1*)))

abbreviation *mk-z* ≡ *mk-x 0*

abbreviation *mk-farg* ≡ *mk-x 1*

type-synonym *exp* = *tannot exp*

17.2 Literals and Values

inductive *lit-conv* :: *lit* ⇒ *la* ⇒ *bool* (⊢ - ⇝ -) **where**
lit-conv-unitI: ⊢ (*SailAST.L-unit*) ⇝ *MiniSailAST.L-unit*
| *lit-conv-trueI*: ⊢ (*SailAST.L-true*) ⇝ *MiniSailAST.L-true*
| *lit-conv-falseI*: ⊢ (*SailAST.L-false*) ⇝ *MiniSailAST.L-false*
| *lit-conv-bvec-bit*: ⊢ (*SailAST.L-bin bs*) ⇝ (*MiniSailAST.L-bitvec* (*List.map* (λ*b*. *if b* = *CHR ''1'' then BitOne else BitZero*) (*String.explode bs*)))
| *lit-conv-bvec-hex*: ⊢ (*SailAST.L-hex bs*) ⇝ (*MiniSailAST.L-bitvec* [])
| *lit-conv-num*: *lit-conv* (*SailAST.L-num ii*) (*MiniSailAST.L-num* (*int-of-integer ii*))

code-pred *lit-conv* .

values { *x*. *lit-conv* (*SailAST.L-bin* (*STR ''0101''*)) *x* }

inductive *b-of-typ* :: *typ* ⇒ *b* ⇒ *bool* **where**
b-of-typ unit-typ B-unit
| *b-of-typ* ((*Typ-app* ((*id STR ''atom-bool''*)) -)) *B-bool*
| *b-of-typ* ((*Typ-app* ((*id STR ''atom''*)) -)) *B-int*
| *b-of-typ* ((*Typ-app* ((*id STR ''bitvector''*)) -)) *B-bitvec*
| ⟦
  b-of-typ t1 b1;
  b-of-typ t2 b2
⟧ ⟹ *b-of-typ* ((*Typ-tup* [*t1*,*t2*])) (*B-pair b1 b2*)
| *b-of-typ* (*Typ-id* (*id tyid*)) (*B-id* (*String.explode tyid*))

definition *cf* :: *'a* ⇒ *char list*
 where *cf* ≡ (λ-. *''''*)

type-synonym *xmap* = (*SailAST.id* ∗ *xa*) *list*

inductive *v-conv* :: *env* ⇒ *xmap* ⇒ *exp* ⇒ Θ ⇒ Γ ⇒ *va* ⇒ *bool* (- ; - ⊢ - ⇝ - ; - ; -) **where**
v-conv-litI: ⟦
trace ''*v-conv-litI*'';
⊢ *l* ⇝ *la*
⟧ ⟹ *E* ; - ⊢ ((*SailAST.E-lit* - *l*)) ⇝ [] ; *GNil* ; (*MiniSailAST.V-lit la*)

| *v-conv-varI*:⟦
trace ''*v-conv-varI*'';
Some xa = *SailEnv.lookup xm idd*;
Some t = *lookup-local-id-env env idd*;
b-of-typ t ba;
trace (''*v-conv-varI end*'' @ (*cf ba*))
⟧ ⟹
env ; *xm* ⊢ ((*SailAST.E-id* - *idd*)) ⇝ [] ; (*xa*,*ba*,*C-true*) #Γ *GNil* ; (*V-var xa*)

| *v-conv-enumI*:⟦
trace ''*v-conv-enumI*'';
Some (*Typ-id* (*id enum-id*)) = *lookup-enum-env env* (*id enum*)
⟧ ⟹
env ; - ⊢ ((*SailAST.E-id* - (*id enum*))) ⇝ [] ; *GNil* ; (*V-cons* (*String.explode enum-id*) (*String.explode enum*) (*V-lit L-unit*))

| *v-conv-pairI*: ⟦
trace ''*v-conv-pairI*'';
env ; *m* ⊢ *vp1* ⇝ *T1* ; *G1*; *v1* ;
env ; *m* ⊢ *vp2* ⇝ *T2* ; *G2*; *v2*
⟧ ⟹
env ; *m* ⊢ ((*SailAST.E-tuple* - [*vp1*, *vp2*])) ⇝ *T1* @ *T2* ; *G1* @ *G2* ; (*MiniSailAST.V-pair v1 v2*)

code-pred (*modes*: *i* ⇒ *i* ⇒ *i* ⇒ *o* ⇒ *o* ⇒ *o* ⇒ *bool*) [*show-steps*, *show-mode-inference*, *show-invalid-clauses*] *v-conv* .

17.3 Constraints

type-synonym *type-vars* = (*kid* ∗ *kind* ∗ *ce*) *list*

fun *lookup-kid* :: *type-vars* ⇒ *kid* ⇒ *ce option* **where**
lookup-kid [] - = *None*
| *lookup-kid* (((*k1*) , - ,*ce*) # *tvs*) (*k2*) = (*if k1* = *k2 then Some ce else lookup-kid tvs* (*k2*))

fun *mk-type-vars* :: (*kid* ∗ *kind*) *list* ⇒ *ce* ⇒ *type-vars* **where**
mk-type-vars [] - = []
| *mk-type-vars* [(*kid*,*kind*)] *ce* = [(*kid*,*kind*,*ce*)]
| *mk-type-vars* ((*kid*,*kind*)#(*k*#*kv*)) *ce* = (*kid*,*kind*,*CE-fst ce*) # (*mk-type-vars* (*k*#*kv*) (*CE-snd ce*))

```
value mk-type-vars [ ( (var (STR ''k'')) , K-int ) ,( (var (STR ''n'')) , K-int ) ,  ( (var (STR ''m'')) ,
K-int ) ]
      (CE-fst (CE-val (V-var (mk-x 1))))


inductive
  b-conv :: typ ⇒ b ⇒ bool
  where
b-conv ( (Typ-id ( (id (STR ''unit'')) ))  ) MiniSailAST.B-unit



fun fvs-x-typ :: typ => id set where
  fvs-x-typ -  = {}


fun fvs-x-gs :: (id*mut*typ) list => id set where
  fvs-x-gs [] = {}
| fvs-x-gs ((x,-,typ)#gs) = {x} ∪ fvs-x-typ typ ∪ fvs-x-gs gs



inductive
  ce-conv :: type-vars ⇒ nexp ⇒ Θ ⇒ Γ ⇒ ce ⇒ bool (  - ⊢ - ⇝ - ; - ; -) and
  c-conv :: type-vars ⇒ n-constraint ⇒ Θ ⇒ Γ ⇒ c ⇒ bool  (  - ⊢ - ⇝ - ; - ; -) and
  t-conv-raw  :: type-vars ⇒ typ ⇒ ce ⇒ Θ ⇒ Γ ⇒  ba ⇒ ca ⇒ bool and
  g-conv-aux :: env ⇒ type-vars ⇒ (id*mut*typ) list  ⇒ Θ ⇒ MiniSailAST.Γ  ⇒ bool and
  c-bool-conv :: type-vars ⇒ n-constraint ⇒ ce ⇒ Θ ⇒ Γ ⇒ c ⇒ bool
  where

g-conv-nilI:g-conv-aux - - [] [] GNil
|g-conv-cons1I: ⟦
   g-conv-aux env type-vars gs T1 G1 ;
   Some xa = mk-prog-x env x;
   x ∉ fvs-x-gs gs;
   t-conv-raw type-vars typ (CE-val [xa]^v) T2 G2  ba ca
⟧ ⟹
   g-conv-aux env type-vars  ((x,Immutable,typ) # gs) (T1@T2) (((xa, ba, ca) #Γ G2) @ G1)

|g-conv-cons2I: ⟦
   g-conv-aux env type-vars gs T1 G1
⟧ ⟹
   g-conv-aux env type-vars  ((x,Mutable,typ) # gs) T1 G1

| ce-conv-sumI:
  ⟦
   ce-conv env ne1 T1 G1 ca1 ; ce-conv env ne2 T2 G2 ca2
⟧ ⟹ ce-conv env ( (Nexp-sum ne1 ne2)  ) (T1@T2) (G1@G2) (MiniSailAST.CE-op Plus ca1 ca2)

| ce-conj-constI:  ce-conv env ( (Nexp-constant ii)  )  [] GNil (MiniSailAST.CE-val (V-lit (L-num
```

(*int-of-integer ii*))))

|*ce-conj-kidI*:
Some ce = lookup-kid env k ⟹ *ce-conv env* ((*Nexp-var k*)) [] *GNil ce*

| ⟦ *ce-conv G cep1 T1 G1 cea1* ; *ce-conv G cep2 T2 G2 cea2*
⟧ ⟹ *c-conv G* ((*NC-equal cep1 cep2*)) (*T1@T2*) (*G1@G2*) (*MiniSailAST.C-eq cea1 cea2*)
| ⟦ *ce-conv G cep1 T1 G1 cea1* ; *ce-conv G cep2 T2 G2 cea2*
⟧ ⟹ *c-conv G* ((*NC-bounded-ge cep1 cep2*)) (*T1@T2*) (*G1@G2*) (*MiniSailAST.C-eq* (*CE-op LEq cea2 cea1*) (*CE-val* (*V-lit L-true*)))
| ⟦ *ce-conv G cep1 T1 G1 cea1* ; *ce-conv G cep2 T2 G2 cea2*
⟧ ⟹ *c-conv G* ((*NC-bounded-le cep1 cep2*)) (*T1@T2*) (*G1@G2*) (*MiniSailAST.C-eq* (*CE-op LEq cea1 cea2*) (*CE-val* (*V-lit L-true*)))

| *c-conv G* (*NC-true*) [] *GNil MiniSailAST.C-true*
| *c-conv G* (*NC-false*) [] *GNil MiniSailAST.C-false*

| ⟦ *c-conv G cp1 T1 G1 ca1* ; *c-conv G cp2 T2 G2 ca2* ⟧ ⟹ *c-conv G* ((*NC-and cp1 cp2*)) (*T1@T2*) (*G1@G2*) (*MiniSailAST.C-conj ca1 ca2*)
| ⟦ *c-conv G cp1 T1 G1 ca1* ; *c-conv G cp2 T2 G2 ca2* ⟧ ⟹ *c-conv G* ((*NC-or cp1 cp2*)) (*T1@T2*) (*G1@G2*) (*MiniSailAST.C-disj ca1 ca2*)

| *c-bool-conv-trueI*: *c-bool-conv env NC-true ce* [] *GNil* (*ce* == $[[$*MiniSailAST.L-true*$]^{v}]^{ce}$)
| *c-bool-conv env NC-false ce* [] *GNil* (*ce* == $[[$*MiniSailAST.L-false*$]^{v}]^{ce}$)
| *Some ce′ = lookup-kid env k* ⟹ *c-bool-conv env* (*NC-var k*) *ce* [] *GNil* (*ce* == *ce′*)

| *c-conv-raw-not*: ⟦
Some ce′ = lookup-kid env k
⟧ ⟹ *c-bool-conv env* (*NC-app* (*id* (*STR ''not''*)) [*A-bool* (*NC-var k*)]) *ce* [] *GNil* (*C-not* (*ce* == *ce′*))

| *c-conv-raw-andI*: ⟦
Some ce1 = lookup-kid env k1;
Some ce2 = lookup-kid env k2
⟧ ⟹ *c-bool-conv env* (*NC-and* (*NC-var k1*) (*NC-var k2*)) *ce* [] *GNil* (*C-conj* (*ce* == *ce1*) (*ce* == *ce2*))

| *c-conv-raw-eqI*: ⟦
Some ce1 = lookup-kid env k1;
Some ce2 = lookup-kid env k2
⟧ ⟹ *c-bool-conv env* (*NC-equal* (*Nexp-var k1*) (*Nexp-var k2*)) *ce* [] *GNil*
(*C-disj* (*C-conj* (*ce* == $[[$*MiniSailAST.L-true*$]^{v}]^{ce}$) (*C-eq ce1 ce2*))
(*C-conj* (*ce* == $[[$*MiniSailAST.L-false*$]^{v}]^{ce}$) (*C-not* (*C-eq ce1 ce2*))))

| *c-conv-raw-gtI*: ⟦
Some ce1 = lookup-kid env k1;
Some ce2 = lookup-kid env k2
⟧ ⟹ *c-bool-conv env* (*NC-bounded-gt* (*Nexp-var k1*) (*Nexp-var k2*)) *ce* [] *GNil* (*C-not* (*ce* == (*CE-op LEq ce1 ce2*)))

| *c-conv-raw-ltI*: ⟦
Some ce1 = lookup-kid env k1;
Some ce2 = lookup-kid env k2
⟧ ⟹ *c-bool-conv env* (*NC-bounded-lt* (*Nexp-var k1*) (*Nexp-var k2*)) *ce* [] *GNil* (*ce* == (*CE-op LEq ce2 ce1*))

| *t-conv-raw-unitI*: *t-conv-raw env unit-typ* *ce* [] *GNil* *B-unit* (*ce* == $[[MiniSailAST.L\text{-}unit]^{v}]^{ce}$)
| *t-conv-raw-intI*: *t-conv-raw env int-typ* *ce* [] *GNil* *B-int* (*C-true*)
| *t-conv-raw-boolI*: *t-conv-raw env bool-all-typ* *ce* [] *GNil* *B-bool* (*C-true*)

| *t-conv-atom-boolI*: *c-bool-conv env nc ce T G ca* ⟹
t-conv-raw env (*Typ-app* (*id* (*STR ''atom-bool''*)) [*A-bool nc*]) *ce T G B-bool ca*

| *t-conv-raw-atomI*: *ce-conv env nexp T G ce2* ⟹ *t-conv-raw env* ((*Typ-app* ((*id STR ''atom''*))
[(*A-nexp nexp*)])) *ce T G* *B-int* (*ce* == *ce2*)

| *t-conv-raw-bitvectorI*: *ce-conv env nexp T G ce2* ⟹ *t-conv-raw env* (*Typ-app* (*id STR ''bitvector''*)
[*A-nexp nexp* , -]) *ce T G* *B-bitvec* (*CE-len ce* == *ce2*)

| *t-conv-raw-rangeI*: ⟦
ce-conv env nexp1 T1 G1 ce1;
ce-conv env nexp2 T2 G2 ce2
⟧ ⟹ *t-conv-raw env* ((*Typ-app* ((*id STR ''range''*))
[(*A-nexp nexp1*) , (*A-nexp nexp2*)])) *ce* (*T1@T2*) (*G1@G2*) *B-bool*
((*C-eq* (*CE-op LEq ce1 ce*) (*CE-val* (*V-lit L-true*))) *AND* (*C-eq* (*CE-val* (*V-lit L-true*))
(*CE-op LEq ce ce2*)))

| *t-conv-raw-tuple-singleI*: ⟦
t-conv-raw env t1 ce T G *ba1 ca1*
⟧ ⟹ *t-conv-raw env* ((*Typ-tup* [*t1*])) *ce T G* *ba1 ca1*

| *t-conv-raw-tuple-pairI*: ⟦
t-conv-raw env t1 (*CE-fst ce*) *T1 G1* *ba1 ca1*;
t-conv-raw env (*Typ-tup* (*t2#ts*)) (*CE-snd ce*) *T2 G2* *ba2 ca2*
⟧ ⟹ *t-conv-raw env* (*Typ-tup* (*t1#t2#ts*)) *ce* (*T1@T2*) (*G1@G2*) (*B-pair ba1 ba2*) *C-true*

| *t-conv-raw env* (*Typ-id* (*id tyid*)) *ce* [] *GNil* (*B-id* (*String.explode tyid*)) *C-true*

| *t-conv-raw-tuple-implicit-nexpI*: ⟦
trace (*''t-conv-raw-tuple-implicit-nexpI''* @ (*show tfn*));
tfn = (*SailAST.id STR ''implicit''*);
t-conv-raw env (*Typ-app* (*SailAST.id STR ''atom''*) [*A-nexp ne*]) *ce T G* *ba1 ca1*
⟧ ⟹ *t-conv-raw env* ((*Typ-app tfn* [*A-nexp ne*])) *ce T G* *ba1 ca1*

code-pred (*modes*: *i* ⇒ *i* ⇒ *o* ⇒ *o* ⇒ *o* ⇒ *bool*) [*show-steps*, *show-mode-inference*, *show-invalid-clauses*]
ce-conv .
code-pred (*modes*: *i* ⇒ *i* ⇒ *o* ⇒ *o* ⇒ *o* ⇒ *bool*) [*show-steps*, *show-mode-inference*, *show-invalid-clauses*]

c-conv .
code-pred (*modes*: $i \Rightarrow i \Rightarrow i \Rightarrow o \Rightarrow o \Rightarrow o \Rightarrow bool$) [*show-steps*, *show-mode-inference*, *show-invalid-clauses*] *c-bool-conv* .
code-pred (*modes*: $i \Rightarrow i \Rightarrow i \Rightarrow o \Rightarrow o \Rightarrow o \Rightarrow o \Rightarrow bool$) [*show-steps*, *show-mode-inference*, *show-invalid-clauses*] *t-conv-raw* .
code-pred (*modes*: $i \Rightarrow i \Rightarrow i \Rightarrow o \Rightarrow o \Rightarrow bool$) [*show-steps*, *show-mode-inference*, *show-invalid-clauses*] *g-conv-aux* .

17.4 Types

value *mk-x 0*

values { (*ba*,*ca*,*T*,*G*). *t-conv-raw* [] (*Typ-tup* []) (*CE-val* (*V-var* (*mk-x 0*))) *T G ba ca* }
values { (*ba*,*ca*,*T*,*G*). *t-conv-raw* [] (*Typ-tup* [*unit-typ*]) (*CE-val* (*V-var* (*mk-x 0*))) *T G ba ca* }
values { (*ba*,*ca*,*T*,*G*). *t-conv-raw* [] (*Typ-tup* [*unit-typ*,*unit-typ*]) (*CE-val* (*V-var* (*mk-x 0*))) *T G ba ca* }
values { (*ba*,*ca*,*T*,*G*). *t-conv-raw* [] (*Typ-tup* [*unit-typ*,*unit-typ*,*unit-typ*]) (*CE-val* (*V-var* (*mk-x 0*))) *T G ba ca* }

values { (*ba*,*ca*,*T*,*G*). *t-conv-raw* [] (*Typ-app* (*id STR ''implicit''*) [*A-nexp* (*Nexp-constant 1*)]) (*CE-val* (*V-var* (*mk-x 0*))) *T G ba ca* }

code-pred (*modes*: $i \Rightarrow i \Rightarrow i \Rightarrow o \Rightarrow o \Rightarrow bool$) [*show-steps*, *show-mode-inference*, *show-invalid-clauses*] *t-conv-raw* .

inductive
t-conv :: *type-vars* $\Rightarrow$ *typ* $\Rightarrow$ $\tau \Rightarrow$ *bool* (- $\vdash$ - $\rightsquigarrow$ -) **where**
t-conv-raw env t (*CE-val* (*V-var mk-z*)) *G T ba ca* $\Longrightarrow$ *t-conv env t* (*MiniSailAST.T-refined-type* (*mk-z*) *ba ca*)

| ⟦
t-conv-raw ((*var k*, *K-int*, (*CE-val* (*V-var mk-z*)))#*env*) *typ* (*CE-val* (*V-var mk-z*)) *G T ba ca*;
c-conv ((*var k*, *K-int*, (*CE-val* (*V-var mk-z*)))#*env*) *nc T' G' ca'*
⟧ $\Longrightarrow$ *t-conv env* (*Typ-exist* [*KOpt-kind K-int* (*var k*)] *nc typ*) (*MiniSailAST.T-refined-type* (*mk-z*) *ba* (*C-conj ca' ca*))

code-pred (*modes*: $i \Rightarrow i \Rightarrow o \Rightarrow bool$) [*show-steps*, *show-mode-inference*, *show-invalid-clauses*] *t-conv* .

17.5 Environment to Context

inductive *g-conv* :: *env* $\Rightarrow$ *type-vars* $\Rightarrow$ *MiniSailAST.*$\Gamma \Rightarrow$ *bool* **where**
⟦
g-conv-aux env type-vars (*locals env*) *T* Γ
⟧ $\Longrightarrow$ *g-conv env type-vars* Γ

code-pred (*modes*: $i \Rightarrow i \Rightarrow o \Rightarrow bool$) [*show-steps*, *show-mode-inference*, *show-invalid-clauses*] *g-conv*.

inductive *et-conv* :: *type-vars* $\Rightarrow$ *exp* $\Rightarrow \tau \Rightarrow \Gamma \Rightarrow$ *bool* **where**
⟦ *Some t* = *get-type tan*; *Some env* = *get-env tan* ;

g-conv env type-vars Γ;
t-conv type-vars t ta;
tan = *annot-e exp*
⟧ ⟹
et-conv type-vars exp ta Γ

code-pred (*modes*: *i* ⇒ *i* ⇒ *o* ⇒ *o* ⇒ *bool*) [*show-steps*, *show-mode-inference*, *show-invalid-clauses*]
et-conv .

inductive *d-conv* :: *type-vars* ⇒ (*id*∗*mut*∗*typ*) *list* ⇒ Δ ⇒ *bool* **where**

d-conv-nilI:*d-conv* - [] *DNil*

| *d-conv-mutI*: ⟦
t-conv type-vars typ t;
u = *mk-u 0*;
d-conv type-vars locs D
⟧ ⟹ *d-conv type-vars* ((*x*,*Mutable*,*typ*)#*locs*) ((*u*,*t*)#$_{\Delta}$*D*)

| *d-conv-immutI*: ⟦
d-conv type-vars locs D
⟧ ⟹ *d-conv type-vars* ((*x*,*Immutable*,*typ*)#*locs*) *D*

inductive *env-conv* :: *env* ⇒ Γ ⇒ Δ ⇒ *bool* **where**
⟦
g-conv env [] *G*;
d-conv [] (*locals env*) *D*
⟧ ⟹ *env-conv env G D*

17.6 Let context

datatype *lctx* =
L-hole (□ *1000*)
| *L-continue*
| *L-val va*
| *L-let xa ea lctx*
| *L-if1 va lctx sa*
| *L-if2 va sa lctx*
| *L-if3 va lctx lctx*
| *L-final1 dc xa lctx*
| *L-final2 dc xa sa*
| *L-match va dc xa lctx*
| *L-compose lctx lctx* (- ∘ - *1000*)

fun *lctx-apply* :: *lctx* ⇒ *sa* ⇒ *sa* (- [-]$_L$) **where**

□ [*s*]$_L$ = *s*
| *L-continue* [*s*]$_L$ = *s*
| (*L-val va*) [-]$_L$ = (*AS-val va*)
| (*L-let xa ea L*) [*s*]$_L$ = (*AS-let xa ea* (*L* [*s*]$_L$))

```
| (L-if1 va L sa ) [ s ]L = (AS-if va (  L [ s ]L ) sa)
| (L-if2 va sa L) [ s ]L = (AS-if va sa (  L [ s ]L  ))
| (L-if3 va L1 L2) [ s ]L = (AS-if va (L1 [ s ]L) (L2 [ s ]L))
| (L-match va dc xa L) [ s ]L = AS-match va (AS-cons (AS-branch dc xa (L [ s ]L)) (AS-final (AS-branch
dc xa (AS-val (V-lit L-unit)))))
| (L1 ∘ L2 )[ s ]L = L1 [ L2 [ s ]L ]L


fun apply-hole-continue :: lctx ⇒ sa ⇒ sa ⇒ sa  ( - [ - , - ]  ) where
  □ [ s1 , s2 ] = s1
| L-continue [ s1 , s2 ] = s2
| (L-val va) [ s1 , s2 ] = AS-val va
| (L-let xa ea L) [ s1 , s2 ] = (AS-let xa ea (  L [ s1 , s2 ]  ))
| (L-if1 va L sa ) [ s1 , s2 ] = (AS-if va (  L [ s1 , s2 ] ) sa)
| (L-if2 va sa L) [ s1 , s2 ] = (AS-if va sa (  L [ s1 , s2 ]  ))
| (L-if3 va L1 L2) [ s1 , s2 ] = (AS-if va (L1 [ s1 , s2 ]) (L2 [ s1 , s2 ]))
| (L1 ∘ L2 )[ s1 , s2 ] = L1 [ L2 [ s1 , s2 ] ]L
```

17.7 Patterns

```
type-synonym local-vars = (id ∗ xa) list

type-synonym Σ = type-vars∗local-vars


type-synonym pat-list = (tannot pat)  list
type-synonym pm-row = pat-list  ∗ sa
type-synonym pm = pm-row list


fun mapi :: (nat ⇒ 'a ⇒ 'b) ⇒ 'a list ⇒ 'b list where
  mapi f xs = List.map (λ(i,x). f i x) (zip [0..<(List.length xs)] xs)

fun zipi :: 'a list ⇒ (nat ∗ 'a) list where
  zipi xs = mapi (λi x. (i,x)) xs
```

17.7.1 Expand Literals

```
fun fresh-vars-list :: nat ⇒ nat ⇒ (nat ∗ (xa list)) where
fresh-vars-list i 0 = (i,[])
| fresh-vars-list i k = (let (i2,xalist) = fresh-vars-list (i+1) (k−1) in
                              (i2, (mk-x i ) # xalist))


fun add-to-pmlit-aux :: (pm∗ (la option)) list ⇒ la option ⇒ pm-row ⇒ (pm∗ (la option)) list where
  add-to-pmlit-aux [] la pm-row = [ ([pm-row] , la) ]
| add-to-pmlit-aux (( pm, la' ) # pmlit) la pm-row = (if la' = la then
                                                        (pm-row#pm,la) # pmlit
                                                    else
                                                        (pm, la') # (add-to-pmlit-aux pmlit la pm-row))

fun add-to-pmlit :: (pm∗ (la option)) list ⇒ la option ⇒ pm-row ⇒ (pm∗ (la option)) list where
```

```
  add-to-pmlit pmlit la pm-row = (if List.member (List.map snd pmlit) la then  add-to-pmlit-aux pmlit
la pm-row else
                                  (( [pm-row],la) # pmlit))

fun is-literal-base :: typ ⇒ bool where
  is-literal-base ( (Typ-id ( (id tyid) )) ) = (tyid ∈ {STR ''bool'', STR ''int'' , STR ''unit''})
| is-literal-base ( (Typ-app ( (id tyid) ) - ) ) = (tyid ∈ {STR ''bool'', STR ''int'', STR ''atom-bool'',
STR ''atom'', STR ''vector'', STR ''bitvector'' })
| is-literal-base - = False

fun split :: ((la option) * sa) list ⇒ ( (la*sa) list) * (sa option) where
  split [] = ([], None)
| split ( x # xs ) = (let (litsa, sa ) = split xs in
                      case fst x of
                                    Some la ⇒ ( (la,snd x) # litsa , sa )
                                  | None ⇒  (litsa , Some (snd x)))

inductive get-len :: typ ⇒ v ⇒ bool where
  get-len (Typ-app (id STR ''bitvector'') [ A-nexp (Nexp-constant i), - ]) (V-lit (L-num (int-of-integer
i)))

code-pred (modes : i ⇒ o ⇒ bool) get-len .




fun get-len-lit :: lit ⇒ v where
  get-len-lit (L-hex bs) =  V-lit (L-num (int (length (String.explode bs))*4))
|  get-len-lit (L-bin bs) =  V-lit (L-num (int (length (String.explode bs))))

inductive expand-vector-concat :: nat ⇒ xa ⇒ xa ⇒ (tannot pat) list ⇒ lctx ⇒ xa ⇒ nat ⇒ bool
where


expand-vc-empty:
  ⟦
  trace ''expand-vc-empty''
⟧ ⟹ expand-vector-concat i1 -  xb [] L-hole xb i1

| expand-vc-litI: ⟦
     trace ''expand-vc-lit'';
     (i2,xb1) = mk-fresh-x i1;
     (i3,xa') = mk-fresh-x i2;
     (i4,xa1) = mk-fresh-x i3;
     (i5,xa2) = mk-fresh-x i4;
     lit-conv lit la;
     len = get-len-lit lit;
     L1 = L-let xa' (AE-split (V-var xa) len ) (
          L-let xa1 (AE-fst (V-var xa')) (L-let xa2 (AE-snd (V-var xa')) L-hole));
     L2 = L-let xb1  (AE-app ''eqand'' (V-pair (V-var xb) (V-pair (V-var xa1) (V-lit la)))) L-hole;
     trace (''expand-vc-litI i5='' @ (show i5));
```

expand-vector-concat i5 xa2 xb1 ps L3 xb2 i6;
trace (''*expand-vc-litI i6=*'' @ (*show i6*))
⟧ ⟹ *expand-vector-concat i1 xa xb* ((*P-lit - lit*) # *ps*) (*L-compose* (*L-compose L1 L2*) *L3*) *xb2 i6*

| *expand-vc-varI*: ⟦
trace (''*expand-vc-varI typ=*'' @ (*show typ*));
(*i2*,*xb1*) = *mk-fresh-x i1*;
(*i3*,*xa'*) = *mk-fresh-x i2*;
(*i4*,*xa1*) = *mk-fresh-x i3*;
(*i5*,*xa2*) = *mk-fresh-x i4*;
(*i6*,*xa-id*) = *mk-fresh-x i5*;
get-len typ len;
L1 = *L-let xa'* (*AE-split* (*V-var xa*) *len*) (
L-let xa1 (*AE-fst* (*V-var xa'*)) (*L-let xa2* (*AE-snd* (*V-var xa'*)) *L-hole*));
L2 = *L-let xa-id* (*AE-val* (*V-var xa1*)) *L-hole*;
expand-vector-concat i6 xa2 xb ps L3 xb2 i7;
trace (''*expand-vc-varI i7=*'' @ (*show i7*))
⟧ ⟹ *expand-vector-concat i1 xa xb* ((*P-typ - typ* (*P-id - idd*)) # *ps*) (*L-compose* (*L-compose L1 L2*) *L3*) *xb2 i7*

code-pred (*modes*: *i* ⇒ *i* ⇒ *i* ⇒ *i* ⇒ *o* ⇒ *o* ⇒ *o* ⇒ *bool*) [*show-steps*, *show-mode-inference*, *show-invalid-clauses*]
expand-vector-concat .

values { (*x*,*y*,*z*) . *expand-vector-concat 2* (*mk-x 0*) (*mk-x 1*) [*pat-lit-bv* (*STR* ''*0011*'')] *x y z* }

inductive *expand-lit* :: *nat* ⇒ *env* ⇒ *xmap* ⇒ *pm* ⇒ *xa* ⇒ (*pm* ∗ (*lctx*∗*xa*) *option*) *list* ⇒ (*id*∗*xa*) *list* ⇒ *nat* ⇒ *bool* **where**
expand-lit-litI: ⟦
trace ''*expand-lit-litI*'';
expand-lit i1 env xmap pm xp pmlit ms i2;
trace (''*expand-lit-litI i2 =* '' @ (*show i2*));
lit-conv lit la;
(*i3*,*xb*) = *mk-fresh-x i2*;
L = *L-let xb* (*AE-app* (''*eq*'') (*V-pair* (*V-var xp*) (*V-lit la*))) *L-hole* ;
trace (''*expand-lit-litI i3 =* '' @ (*show i3*))
⟧ ⟹
expand-lit i1 env xmap (((((*P-lit - lit*))) # *pm-pat* , *sa*) # *pm*) *xp* (([(*pm-pat*,*sa*)] , *Some* (*L*,*xb*)) # *pmlit*) *ms i3*

| *expand-lit-wildI*: ⟦
trace ''*expand-lit-wildI*''
⟧ ⟹ *expand-lit i1 env xmap* ((((*P-wild* -)) # *pm-pat* , *sa*) # *pm*) *xp* [([(*pm-pat*,*sa*)], *None*)] [] *i1*

| *expand-lit-varI*: ⟦

```
  trace ''expand-lit-varI'' ;
  Some xa = SailEnv.lookup xmap idd
⟧ ⟹
  expand-lit i1 env xmap (  (  (( (P-id - idd) ) ) # pm-pat , sa ) # pm) xp [ ( [(pm-pat, AS-let xa
(AE-val (V-var xp)) sa)],None)] [(idd,xa)] i1

| expand-lit-typI: ⟦
  trace ''expand-lit-typI'';
  expand-lit i1 env xmap ((pat # pm-pat , sa ) #pm) xp pm' ids i2
⟧ ⟹
  expand-lit i1 env xmap ((((P-typ - - pat) # pm-pat) , sa ) #pm ) xp pm' ids i2


| expand-lit-emptyI: ⟦
  trace ''expand-lit-emptyI''
⟧ ⟹ expand-lit i1 env xmap [] xp [] [] i1


| expand-lit-concatI: ⟦
  trace ''expand-lit-concatI'';
  (i2,xb) = mk-fresh-x i1;
  trace (''expand-lit-concatI i2='' @ (show i2));
  expand-vector-concat i2 xp xb pats L xb2 i3;
  trace (''expand-lit-concatI i3='' @ (show i3));
  expand-lit i3 env xmap pm xp pmlist ms i4;
  trace (''expand-lit-concatI i3='' @ (show i4))
⟧  ⟹ expand-lit i1 env xmap (  (  ( ( (P-vector-concat - pats) )) # pm-pat , sa ) # pm) xp (( [
(pm-pat,sa) ] , Some (L,xb2)) # pmlist) ms i4


code-pred (modes: i ⇒ i ⇒ i ⇒ i ⇒ i ⇒ o ⇒ o ⇒ o ⇒ bool) [show-steps, show-mode-inference,
show-invalid-clauses] expand-lit .


definition pm1 :: pm where
  pm1 = [ ( [ ( (pat-lit-bv  (STR ''0101'') )) ], AS-val (V-lit L-true)),
          ( [ ( (pat-lit-bv  (STR ''0100'') )) ], AS-val (V-lit L-false)) ]

values { (x2,x3,x4). expand-lit 0 emptyEnv [] pm1 (mk-x 0) x2 x3 x4 }
```

17.7.2 Expand Constructor

```
fun add-to-pmctor :: (pm* ((dc*xa) option)) list ⇒ (dc*xa) option ⇒ pm-row ⇒ (pm* ((dc*xa) option)) list where
  add-to-pmctor [] la pm-row = [ ([pm-row] , la) ]
| add-to-pmctor (( pm, la' ) # pmlit) la pm-row = (if la' = la then
                                                     (pm-row#pm,la) # pmlit
                                                   else
                                                     (pm, la') # (add-to-pmctor pmlit la pm-row))


inductive expand-ctor :: nat ⇒ env ⇒ pm ⇒ id ⇒ xa ⇒ (pm * ((dc * xa) option)) list ⇒ (id*xa)
list ⇒ nat ⇒ bool where
```

expand-ctorI:
⟦ *trace ''expand-ctorI''*;
expand-ctor n1 env pm tyid xa pmctor ms n2;
trace (''expand-ctorI i2='' @ *(show n2))*;
((id dc)) = ctor
⟧ ⟹
expand-ctor n1 env ((((P-app - ctor pats))#pm-pat, sa) # pm) tyid xa
(add-to-pmctor pmctor (Some (String.explode dc,mk-x 0)) (pats @ pm-pat,sa)) ms n2

| *expand-ctor-wildI*:
expand-ctor n1 env (((P-wild -)#pm-pat, sa) # pm) tyid xa [([(pm-pat,sa)],None)] [] n1

| *expand-ctor-varI*:
⟦
trace ''expand-ctor-varI'';
None = lookup-enum-env env idd
⟧ ⟹ *expand-ctor n1 env ((((P-id - idd))#pm-pat, sa) # pm) tyid xa [([(pm-pat,sa)],None)] [] n1*

| *expand-ctor-enumI*:
⟦
Some - = lookup-enum-env env (id dc);
trace ''expand-ctor-enumI'';
expand-ctor n1 env pm tyid xa pmctor ms n2;
trace (''expand-enumI i2='' @ *(show n2))*

⟧ ⟹ *expand-ctor n1 env ((((P-id - (id dc)))#pm-pat, sa) # pm) tyid xa (add-to-pmctor pmctor (Some (String.explode dc,mk-x 0))*
((P-lit unk SailAST.L-unit)# pm-pat,sa)) ms n2

| *expand-ctor-emptyI*:
expand-ctor n1 env [] tyid xa [] [] n1

code-pred (*modes*: *i* ⇒ *i* ⇒ *i* ⇒ *i* ⇒ *i* ⇒ *o* ⇒ *o* ⇒ *o* ⇒ *bool*) [*show-steps*, *show-mode-inference*, *show-invalid-clauses*] *expand-ctor* .

17.7.3 Expand Tuple

inductive *expand-tuple* :: *nat* ⇒ *pm* ⇒ *nat* ⇒ *xa* ⇒ *pm* ⇒ *xa list* ⇒ *nat* ⇒ *bool* **where**

expand-tuple-emptyI:
trace ''expand-tuple-emptyI'' ⟹ *expand-tuple i1 ([]) num-ele xa ([]) [] i1*

|*expand-tuple-tupI*: ⟦
trace (''expand-tuple-tupI pats='');
(i2,xalist) = fresh-vars-list i1 num-ele ;
trace (''expand-tuple-tupI i2='' @ *(show i2))*;
expand-tuple i2 pm num-ele xa pm' xalist' i3;
trace (''expand-tuple-tupI i3='' @ *(show i3))*
⟧ ⟹

expand-tuple i1 (Cons ((Cons (((P-tup - pats))) pm-pat) , sa) pm) num-ele xa (Cons ((pats)@pm-pat,sa) pm′) xalist′ i3

| *expand-tuple-wildI*:⟦
(i2,xalist) = fresh-vars-list i1 num-ele
⟧ ⟹
expand-tuple i1 (((P-wild lc) # pm-pat , sa) # pm) num-ele xa
(((List.map (λxa. (P-wild lc)) xalist)@pm-pat,sa)#[]) xalist i1

| *expand-tuple-varI*: ⟦
trace ″expand-tuple-varI″;
(i1,xalist) = fresh-vars-list i1 num-ele
⟧ ⟹
expand-tuple i1 (((((P-id lc idd))) # pm-pat , sa) # pm) num-ele xa (((List.map (λxa. (P-wild lc)) xalist)@pm-pat,sa)#[]) xalist i1

code-pred (*modes*: *i* ⇒ *i* ⇒ *i* ⇒ *i* ⇒ *o* ⇒ *o* ⇒ *o* ⇒ *bool*) [*show-steps*, *show-mode-inference*, *show-invalid-clauses*] *expand-tuple* .

abbreviation *s-unit* ≡ *AS-val* (*V-lit L-unit*)

inductive *as-unpack* :: *nat* ⇒ *xa* ⇒ *xa list* ⇒ *sa* ⇒ *sa* ⇒ *nat* ⇒ *bool* **where**
as-unpack i xa [] sa sa i
| ⟦
(i2,xa) = mk-fresh-x i1;
as-unpack i2 xa xas sa sa′ i3
⟧ ⟹ *as-unpack i1 xa1 (xa2#xas) sa*
(AS-let xa2 (AE-fst (V-var xa1)) (AS-let xa (AE-snd (V-var xa1)) sa′)) i3

code-pred (*modes*: *i* ⇒ *i* ⇒ *i* ⇒ *i* ⇒ *o* ⇒ *o* ⇒ *bool*) [*show-steps*, *show-mode-inference*, *show-invalid-clauses*] *as-unpack* .

values { (*sa*,*i*) . *as-unpack 3 (mk-x 0) [mk-x 1*, *mk-x 2*] *s-unit sa i*}

17.7.4 Convert Pattern Matrix

fun *mk-switch2* :: *xa* ⇒ (*pm* ∗ (*la option*)) *list* ⇒ *sa list* ⇒ *sa* **where**
mk-switch2 xa [(- , -)] [sa] = sa
| *mk-switch2 xa ((- , Some la)#ps) (sa#sas) = LET xa = (AE-app (″eq″) (V-pair (V-var xa) (V-lit la))) IN IF (V-var xa) THEN sa ELSE (mk-switch2 xa ps sas)*

fun *mk-switch* :: *xa* ⇒ (*pm* ∗ ((*lctx* ∗ *xa*) *option*)) *list* ⇒ *sa list* ⇒ *sa* **where**
mk-switch xa ((- , Some (L,xb))#[]) [sa] = L [IF (V-var xb) THEN sa ELSE s-unit]$_L$
| *mk-switch xa ((- , None)#[]) [sa] = sa*
| *mk-switch xa ((- , Some (L,xb))#ps) (sa#sas) = L [IF (V-var xb) THEN sa ELSE (mk-switch xa*

ps sas)]$_L$

fun *mk-match-aux* :: (*pm* ∗ (*dc* ∗ *xa*) *option*) *list* ⇒ *sa list* ⇒ *branch-list* **where**
mk-match-aux [(- , *Some* (*dc*,*xa*))] [*sa*] = *AS-final* (*AS-branch dc xa sa*)
| *mk-match-aux* ((- , *Some* (*dc*,*xa*))#*bs*) (*sa*#*sas*) = *AS-cons* (*AS-branch dc xa sa*) (*mk-match-aux bs sas*)

fun *mk-match* :: *xa* ⇒ (*pm* ∗ (*dc* ∗ *xa*) *option*) *list* ⇒ *sa list* ⇒ *sa* **where**
mk-match xa [(- , *None*)] [*sa*] = *sa*
| *mk-match xa bs sa* = *AS-match* (*V-var xa*) (*mk-match-aux bs sa*)

fun *mk-fresh-many* :: *nat* ⇒ *nat* ⇒ (*nat* ∗ (*xa list*)) **where**
mk-fresh-many i1 0 = (*i1*,[])
| *mk-fresh-many i1 n* = (*let* (*i2*,*xa*) = *mk-fresh-x i1 in*
let (*i3*,*xas*) = *mk-fresh-many i2* (*n*−*1*) *in*
(*i3*,*xa*#*xas*))

definition *mk-pm-list* :: *env* ⇒ *SailAST.id* ⇒ (*pm* ∗ ((*dc* ∗ *xa*) *option*)) *list* ⇒(*typ*∗*xa*) *list* ⇒ (*pm*∗ (*typ*∗*xa*) *list*) *list* **where**
mk-pm-list env tyid pmctor bss = *List.map* (λ(*pm*,*xx*). *case xx of*
Some (*dc*,*xa*) ⇒ (*case lookup-variant-env env tyid* (*id* (*String.implode dc*)) *of*
(*Some typ*) ⇒(*pm*, (*typ*,*xa*)#*bss*)
| *None* ⇒ (*case lookup-enum-env env* (*id* (*String.implode dc*)) *of*
Some - ⇒ (*pm* , (*unit-typ*, *xa*) # *bss*))
)
| *None* ⇒ (*pm*,*bss*)) *pmctor*

inductive *conv-pattern-matrix* :: *nat* ⇒ *env* ⇒ *xmap* ⇒ *pm* ⇒ (*typ*∗*xa*) *list* ⇒ *sa* ⇒ *nat* ⇒ *bool* **and**
conv-pattern-matrix-list ::*nat* ⇒ *env* ⇒ *xmap* ⇒ (*pm*∗ (*typ*∗*xa*) *list*) *list* ⇒ *sa list* ⇒ *nat* ⇒ *bool*
where

conv-pm-list-nilI: ⟦
trace ''*conv-pm-list-nilI*''
⟧ ⟹ *conv-pattern-matrix-list i* - - [] [] *i*

| *conv-pm-list-consI*:⟦
trace ''*conv-pm-list-consI*'';
conv-pattern-matrix i1 env xmap pm bss sa i2;
trace (''*conv-pm-consI i2*=''@*string-of-nat i2*);
conv-pattern-matrix-list i2 env xmap pmlist salist i3;
trace (''*conv-pm-consI i3*=''@*string-of-nat i3*)
⟧ ⟹
conv-pattern-matrix-list i1 env xmap ((*pm*,*bss*)#*pmlist*) (*sa*#*salist*) *i3*

```
|conv-tupleI: ⟦
   trace ''conv-pm-tupleI'';
   (i2,xas) = mk-fresh-many i1 (List.length bs);
   trace (''conv-pm-tupleI i2='' @ (show i2));
   expand-tuple i2 pm (List.length bs) xa  pm' xalist i3 ;
   trace (''conv-pm-tupleI i3='' @ (show i3) @ (show bs));
   conv-pattern-matrix i3 env xmap pm' ( (List.zip bs xas) @ bss) sa i4;
   trace (''conv-pm-tupleI i4='' @ (show i4) @ (show (List.length xalist)));
   as-unpack i4 xa xas sa sa' i5
⟧ ⟹
   conv-pattern-matrix i1 env xmap pm ( ( ( (Typ-tup bs, xa) ) # bss)) sa' i5
```

```
| conv-litI: ⟦
   trace (''conv-pm-litI b='' @ (show b));
   is-literal-base b;
   trace (''conv-pm-litI i1=''@string-of-nat i1);
   expand-lit i1 env xmap pm xa pmlits ms i3 ;
   trace (''conv-pm-litI i3=''@string-of-nat i3);
   conv-pattern-matrix-list i3 env xmap (map (λpm. (pm,bss)) (map fst pmlits)) salist i4;
   trace (''conv-pm-litI i4=''@string-of-nat i4)
⟧ ⟹
   conv-pattern-matrix i1 env xmap pm (Cons (b,xa)  bss) (mk-switch xa pmlits salist)  i4
```

```
| conv-ctorI: ⟦
  trace ''conv-pm-ctorI'';
  ¬ (is-literal-base (Typ-id tyid));
  expand-ctor i1 env pm tyid xa pmctor ms i2;
  trace (''conv-pm-ctorI i2'' @ (show i2));
  pmlist = mk-pm-list env tyid  pmctor bss;
  conv-pattern-matrix-list i2 env xmap pmlist salist i3 ;
  trace (''conv-pm-ctorI i3'' @ (show i3))
⟧ ⟹  conv-pattern-matrix i1 env xmap pm (Cons ( (Typ-id tyid,xa)  )  bss) (mk-match xa pmctor salist
)  i3
```

```
| conv-emptyI: ⟦
   trace ''conv-pm-emptyI''
⟧ ⟹   conv-pattern-matrix i1 env xmap (( ( [], sa )  ) # pm) [] sa i1
```

code-pred (*modes*: $i \Rightarrow i \Rightarrow i \Rightarrow i \Rightarrow i \Rightarrow o \Rightarrow o \Rightarrow bool$) [*show-steps*, *show-mode-inference*, *show-invalid-clauses*] *conv-pattern-matrix* .

values { (*pm'*,*xalist*,*i*). *expand-tuple 0* ([([(*P-wild* (*None*))] , *s-unit*)]) *2* (*mk-x 0*) *pm' xalist i*}

values { (*sa*,*i*). *conv-pattern-matrix 0 emptyEnv* [] ([([] , *s-unit*)]) ([]) *sa i*}

values { (*sa*,*i*,*ms*). *expand-lit 0 emptyEnv* [] ([([(*P-wild unk*)] , *s-unit*)]) (*mk-x 0*) *sa ms i*}

values { (*sa*,*i*,*ms*). *expand-lit 0 emptyEnv* [] ([([(((*P-lit unk* (*SailAST.L-true*))))] , *s-unit*)]) (*mk-x 0*) *sa ms i*}

values {(*x*,*y*). *conv-pattern-matrix 0 emptyEnv* [] [([], *s-unit*)] [] *x y* }

value *nc-true*

value *is-literal-base* (*bool-typ True*)

values {(*x*,*y*). *conv-pattern-matrix 0 emptyEnv* [] ([([(((*P-lit unk* (*SailAST.L-true*))))] , *s-unit*),
([(((*P-lit unk* (*SailAST.L-false*))))] , *s-unit*)
]) [(*bool-typ True*,(*mk-x 0*))] *x y* }

definition *ctor-typ* :: *char list* ⇒ *typ* **where**
ctor-typ s = *Typ-id* (*id* (*String.implode s*))

values {(*x*,*y*). *conv-pattern-matrix 0 emptyEnv* [] [([((*P-lit unk* (*SailAST.L-true*)))], *s-unit*)] [(*bool-typ True*,(*mk-x 0*))] *x y* }

values {(*x*,*y*). *conv-pattern-matrix 0 emptyEnv* [(*id STR ''x''*, *mk-x 1*)] [([(*P-id unk* (*id STR ''x''*))
] , *s-unit*)] [(*ctor-typ ''foo''*,(*mk-x 0*))] *x y* }

values {(*x*,*y*). *conv-pattern-matrix 0 emptyEnv* [(*id STR ''x''*, *mk-x 1*)] [([(*P-id unk* (*id STR ''x''*))
] , *s-unit*)] [(*ctor-typ ''foo''*,(*mk-x 0*))] *x y* }

definition *env3* :: *env* **where**
env3 = *emptyEnv* (| *variants* := [(*id STR ''foo''*, *TypQ-no-forall* , [
(*Tu-ty-id unit-typ* (*id STR ''R''*))])] |)

values {(*x*,*y*). *conv-pattern-matrix 0 env3* [(*id STR ''x''*, *mk-x 1*)] [([
(*P-app unk* (*id STR ''R''*) [*P-id unk* (*id STR ''x''*)])] , *s-unit*)] [(*ctor-typ ''foo''*,(*mk-x 0*))] *x y* }

value ¬ (*is-literal-base* (*ctor-typ ''foo''*))

17.8 Statements

type-synonym *pexp* = *tannot pexp*
type-synonym *pat* = *tannot pat*

inductive *pat-conv* :: *env* ⇒ *pat* ⇒ *string* ⇒ *xa* ⇒ *bool* **where**
Some xa = *mk-xa G x* ⟹ *pat-conv G* ((*P-app* - ((*id ctor*)) [(*P-id* - *x*)])) *ctor xa*

fun *mk-str'* :: *'a* ⇒ *string* **where**
mk-str' s = *STR ''''*

fun *mk-str* :: *'a* ⇒ *char list* **where**

mk-str t = String.explode (mk-str' t)

code-printing
constant *mk-str'* ⇀ (*Eval*) (@{*make'-string*} -) — note indirection via antiquotation

value *trace* (*mk-str* [*True*,*False*])

inductive
e-conv :: *nat* ⇒ *xmap* ⇒ *tannot SailAST.exp* ⇒ *xa* ⇒ Θ ⇒ Φ ⇒ $\mathcal{B}$ ⇒ Γ ⇒ Δ ⇒ *lctx* ⇒ *nat* ⇒ *bool* (- ; - ⊢ - ; - ⇝ - ; - ; - ; - ; - ; - ; -) **and**
e-conv-list :: *nat* ⇒ *xmap* ⇒ *tannot SailAST.exp list* ⇒ *xa* ⇒ Θ ⇒ Φ ⇒ $\mathcal{B}$ ⇒ Γ ⇒ Δ ⇒ *lctx* ⇒ *nat* ⇒ *bool* **and**
s-conv :: *nat* ⇒ *xmap* ⇒ *tannot SailAST.exp* ⇒ Θ ⇒ Φ ⇒ $\mathcal{B}$ ⇒ Γ ⇒ Δ ⇒ *s* ⇒ *nat* ⇒ *bool* (- ⊢ - ⇝ -) **and**
pexp-list-conv :: *nat* ⇒ *env* ⇒ *xmap* ⇒ *pexp list* ⇒ *xa* ⇒ *pm* ⇒ *nat* ⇒ *bool*

where

e-conv-valI: ⟦
Some E = get-env-exp e;
trace (''e-conv-valI n='' @ show n) ;
E ; xm ⊢ e ⇝ T ; G ; va;
trace (''e-conv-valI len='' @ (show (List.length T)))
⟧ ⟹ *n ; xm ⊢ e ; xa ⇝ T ; [] ; {||} ; G ; DNil ; L-let xa (AE-val va) L-hole ; n*

| *e-conv-regI*: ⟦
trace (''e-conv-regI n='' @ show n) ;
Some env = get-env tan;
Some regi = lookup-register-index-env env reg;
Some typ = lookup-register-env env reg;
t-conv [] typ ta;
trace (''e-conv-regI n='' @ (show regi))
⟧ ⟹ *n ; xm ⊢ (E-id tan reg) ; xa ⇝ [] ; [] ; {||} ; GNil ; ((mk-u regi, ta)#$_\Delta$DNil) ; L-let xa (AE-mvar (mk-u regi)) L-hole ; n*

| *e-conv-appI*: ⟦
trace ''e-conv-appI'';
(n2,xa') = mk-fresh-x n1;
e-conv-list n2 xm es xa' T P B G D L n3
⟧ ⟹ *n1 ; xm ⊢ (E-app tan (id fid) es) ; xa ⇝ T ; P ; B ; G ; D ; L ∘ (L-let xa (AE-app (String.explode fid) (V-var xa')) L-hole) ; n3*

| *e-conv-tupleI*: ⟦
trace ''e-conv-tupleI'';
e-conv-list n1 xm es xa T P B G D L n2
⟧ ⟹ *n1 ; xm ⊢ (E-tuple tan es) ; xa ⇝ T ; P ; B ; G ; D ; L ; n2*

| *e-conv-list-singleI*: ⟦

trace (''*e-conv-list-singleI n1=*'' @ (*show n1*));
e-conv n1 xm e xa T P B G D L n2;
trace (''*e-conv-list-singleI n2=*'' @ (*show n2*))
⟧ ⟹ *e-conv-list n1 xm* [*e*] *xa T P B G D L n2*

| *e-conv-list-consI*: ⟦
trace (''*e-conv-list-consI n1=*'' @ (*show n1*));
(*n2*,*xa'*) = *mk-fresh-x n1*;
e-conv n2 xm e xa' T' P' B' G' D' L1 n3;
trace (''*e-conv-list-consI n3=*'' @ (*show n3*));
(*n4*,*xa''*) = *mk-fresh-x n3*;
e-conv-list n4 xm (*es*) *xa''* *T P B G D L2 n5*;
trace (''*e-conv-list-consI n5=*'' @ (*show n5*))
⟧ ⟹ *e-conv-list n1 xm* (*e*#*es*) *xa T P B G D* (*L-compose L1* (*L-compose L2* (*L-let xa* (*AE-val* (*V-pair* (*V-var xa'*) (*V-var xa''*))) *L-hole*))) *n5*

| *s-conv-expI*: ⟦
trace (''*s-conv-expI n1=*'' @ (*show n1*));
(*n2*,*xa*) = *mk-fresh-x n1*;
n2 ; *xm* ⊢ *e* ; *xa* ⇝ *T* ; *P* ; *B* ; *G* ; *D* ; *L* ; *n3*;
trace (''*s-conv-expI n3=*'' @ (*show n3*))
⟧ ⟹ *s-conv n1 xm e T P B G D* (*L* [*AS-val* (*V-var xa*)]$_L$) *n3*

| *s-conv-block-single*: ⟦
trace (''*s-conv-blockI n1=*'' @ (*show n1*));
s-conv n1 xm e T P B G D L n2;
trace (''*s-conv-blockI n2=*'' @ (*show n2*))
⟧ ⟹ *s-conv n1 xm* (*E-block* - [*e*]) *T P B G D L n2*

| *s-conv-assignI*: ⟦
(*n2*,*xa*) = *mk-fresh-x n1*;
e-conv n2 xm e xa T P B G D L n3;
Some env = *get-env tan*;
Some regi = *lookup-register-index-env env regid*
⟧ ⟹ *s-conv n1 xm* (*E-assign tan* (*LEXP-id* - *regid*) *e*) *T P B G D L*[*AS-assign* (*mk-u regi*) (*V-var xa*)]$_L$ *n3*

| *s-conv-ifI*: ⟦
(*n2*,*xa*) = *mk-fresh-x n1*;
e-conv n2 xm e1 xa T1 P1 B1 G1 D1 L n3;
s-conv n3 xm e2 T2 P2 B2 G2 D2 sa2 n4;
s-conv n4 xm e3 T3 P3 B3 G3 D3 sa3 n5
⟧ ⟹
s-conv n1 xm ((*E-if* - *e1 e2 e3*)) *T3 P3 B3 G3 D3* (*L* [*AS-if* [*xa*]^v *sa2 sa3*]$_L$) *n4*

| *s-conv-letI*: ⟦
trace ''*letI*'';
(*i2*,*xa*) = *mk-fresh-x i1*;
e-conv i2 xm e1 xa T1 P1 B1 G1 D1 L i3;

```
   Some t = type-of-exp e1;
   trace (''t='' @ (show t));
   Some env = get-env tan;
   pexp-list-conv i3 env xm [ (Pat-exp None pat e2) ] xa pm i4;
   trace (''done pexp list conv'' @ (mk-str pm) @ (string-of-nat i4));
   conv-pattern-matrix i4 env xm pm [(t, xa)] sa i5;
   trace (''elab i5='' @ (string-of-nat i5))
⟧ ⟹
   s-conv i1 xm ( (E-let tan ( (LB-val lc pat e1 ) ) e2)) T1 P1 B1 G1 D1 (L [ sa ]_L) i5
```

```
| ep-s-elab-matchI: ⟦
  trace (''ep-s-elab-matchI i1='' @ (show i1));
  Some env = get-env tan;
  (i2, xa) = mk-fresh-x i1;
  e-conv i2 xm ep xa T P B G D L i3;
  trace (''ep-s-elab-matchI i3='' @ (show i3));
  pexp-list-conv i3 env xm pexps xa pm i4;
  trace (''ep-s-elab-matchI i4='' @ (show i4));
  conv-pattern-matrix i4 env xm pm [(t,xa)] sa i5;
  trace (''ep-s-elab-matchI i5='' @ (show i5));
  Some t = type-of-exp ep
⟧ ⟹
  s-conv i1 xm ( (E-case tan ep pexps) ) T P B G D (L [ sa ]_L) i5
```

```
|  ep-s-elab-pexp-singletonI: ⟦
  trace (''ep-s-elab-pexp-singletonI i1='' @ (show i1));
  s-conv i1 xm ep T P B G D sa i2;
  trace (''ep-s-elab-pexp-singletonI i2='' @ (show i2))
⟧ ⟹
  pexp-list-conv i1  env xm [ ( (Pat-exp - patp ep)  ) ] xa [ ([ (patp ) ], sa)] i2
```

```
|  ep-s-elab-pexp-consI: ⟦
  trace ''ep-s-elab-pexp-consI'';
  s-conv i1 xm ep T P B G D sa i2;
  pexp-list-conv i2  env xm (pexp#pexps) xa pm i3
⟧ ⟹
  pexp-list-conv i1 env xm (( (Pat-exp  - patp ep)) # pexp # pexps) xa (([ (patp ) ], sa)#pm) i3
```

code-pred (*modes*:
  e-conv : $i \Rightarrow i \Rightarrow i \Rightarrow i \Rightarrow o \Rightarrow o \Rightarrow o \Rightarrow o \Rightarrow o \Rightarrow o \Rightarrow o \Rightarrow$ *bool* **and**
  e-conv-list : $i \Rightarrow i \Rightarrow i \Rightarrow i \Rightarrow o \Rightarrow o \Rightarrow o \Rightarrow o \Rightarrow o \Rightarrow o \Rightarrow o \Rightarrow$ *bool* **and**
  s-conv : $i \Rightarrow i \Rightarrow i \Rightarrow o \Rightarrow o \Rightarrow o \Rightarrow o \Rightarrow o \Rightarrow o \Rightarrow o \Rightarrow$ *bool* **and**
   pexp-list-conv : $i \Rightarrow i \Rightarrow i \Rightarrow i \Rightarrow i \Rightarrow o \Rightarrow o \Rightarrow$ *bool*) [*show-steps*, *show-mode-inference*, *show-invalid-clauses*] *s-conv* .

```
values { (T,P,B,G,D,sa,n) . s-conv 0 [] ( (E-if (set-type None unit-typ)
                  ( (E-lit (set-type None (bool-typ True)) ( SailAST.L-true )) )
                      ( (E-lit (set-type None unit-typ) ( SailAST.L-unit )) )
                      ( (E-lit (set-type None unit-typ) ( SailAST.L-unit )) )
                  )) T P B G D sa n}

values { (T,P,B,G,D,sa,n) . s-conv 0 [] ( (E-let (set-type None unit-typ)
            ( (LB-val None ( (P-id (set-type None (bool-typ True)) ( (id (STR ''x'')) )) )
                  ( (E-lit (set-type None (bool-typ True))) ( SailAST.L-true ))       ))
                  ( (E-lit (set-type None unit-typ) ( SailAST.L-unit )))
                                  )) T P B G D sa n}


values  { (T,P,B,G,D,sa,n) . s-conv 0 [] (e-let e-unit unit-typ) T P B G D sa n}



values { (T,P,B,G,D,sa,n) . s-conv 0  [] ( (E-let (set-type None unit-typ)
            ( (LB-val None ( (P-id (set-type None (bool-typ True)) ( (id (STR ''x'')) )) )
                  ( (E-lit (set-type None (bool-typ True))) ( SailAST.L-true ))       ))
                  ( (E-lit (set-type None unit-typ) ( SailAST.L-unit )))
                                  )) T P B G D sa n}

value  ( (LB-val None (pat-id (STR ''x'')) (e-unit)))

values { (T,P,B,G,D,sa,n) . s-conv 0  [] ( (E-let (set-type None unit-typ)
            ( (LB-val None (pat-unit) (e-unit))) e-unit
                                  )) T P B G D sa n}


values { (T,P,B,G,D,sa,n) . s-conv 0 [] ( (E-let (set-type None unit-typ)
            ( (LB-val None (pat-id (STR ''x'')) (e-unit))) e-unit
                                  )) T P B G D sa n}

values { (T,P,B,G,D,sa,n) . s-conv 0 [] ( (E-let (set-type None unit-typ)
            ( (LB-val None (pat-pair) (e-pair))) e-unit

                                  )) T P B G D sa n}


values { (x,y) . pexp-list-conv 0 emptyEnv [] [] (mk-x 0) x y }

values { (x,y) . pexp-list-conv 0 emptyEnv [] [
              ( Pat-exp  (set-type None (bool-typ True))
                ( (P-id  (set-type None (bool-typ True))) ( (id (STR ''x'')) ))
                      ( (E-lit (set-type None unit-typ)) ( SailAST.L-unit )) ] (mk-x 0) x y }



value (bool-typ True)
```

17.9 Definitions

primrec *option-map* :: (*'a* ⇒ *'b option*) ⇒ *'a list* ⇒ *'b list* **where**
option-map f [] = []
| *option-map f* (*x*#*xs*) = (*case f x of Some y* ⇒ *y* # *option-map f xs* | - ⇒ *option-map f xs*)

fun *mk-xxx* :: *quant-item list* ⇒ (*kid*∗*kind*) *list* **where**
mk-xxx qi-list = (*option-map* (λ*qi*. *case qi of* ((*QI-id* ((*KOpt-kind kind kid*)))) ⇒ *Some* (*kid*,*kind*)
| - ⇒ *None*) *qi-list*)

fun *mk-tq-map* :: *typquant* ⇒ *type-vars* **where**
mk-tq-map (*TypQ-no-forall*) = []
| *mk-tq-map* ((*TypQ-tq qi-list*)) = *mk-type-vars* (*mk-xxx qi-list*)
(*CE-fst* (*CE-val* (*V-var* (*mk-x 1*))))

inductive *mk-tq-c-aux* :: *type-vars* ⇒ *quant-item list* ⇒ *c* ⇒ *bool* **where**
mk-tq-c-aux - [] (*C-true*)
| *mk-tq-c-aux m qis c* ⟹ *mk-tq-c-aux m* ((*QI-id* -)#*qis*) *c*
| ⟦ *c-conv m nc T G c1*; *mk-tq-c-aux m qis c2* ⟧ ⟹ *mk-tq-c-aux m* ((*QI-constraint nc*)#*qis*) (*c1 AND c2*)

inductive *mk-tq-c* :: *type-vars* ⇒ *typquant* ⇒ *c* ⇒ *bool* **where**
mk-tq-c - (*TypQ-no-forall*) *C-true*
| *mk-tq-c-aux m qi-list c* ⟹ *mk-tq-c m* ((*TypQ-tq qi-list*)) *c*

code-pred (*modes*: *i* ⇒ *i* ⇒ *o* ⇒ *bool*)
[*show-steps*, *show-mode-inference*, *show-invalid-clauses*]
mk-tq-c .

inductive *def-funtyp* :: *typquant* ⇒ *typ* ⇒ *ba* ⇒ *ca* ⇒ *τ* ⇒ *bool* **where**
⟦ *tv-map* = *mk-tq-map tyq* ;
k-typ = *List.map* (λ(-,*k*,-). *case k of*
K-int ⇒ *int-typ* | *K-bool* ⇒ *bool-all-typ*) *tv-map*;
k-typ' = (*if length k-typ* = *0 then unit-typ else*
(*if length k-typ* = *1 then hd k-typ else*
Typ-tup k-typ));
mk-tq-c tv-map tyq ca;
t-conv tv-map out-typ ta ;
trace (''*def-funtyp out-typ*='' @ (*show out-typ*) @ (*show in-typs*));
t-conv-raw tv-map k-typ' (*CE-fst* (*CE-val* (*V-var* (*mk-x 1*)))) *T1 G1 ba1 ca1*;
t-conv-raw tv-map (*Typ-tup in-typs*) (*CE-snd* (*CE-val* (*V-var* (*mk-x 1*)))) *T2 G2 ba2 ca2*
⟧ ⟹ *def-funtyp tyq* ((*Typ-fn in-typs out-typ* -)) (*B-pair ba1 ba2*) (*C-conj ca* (*C-conj ca1 ca2*)) *ta*

code-pred (*modes*: *i* ⇒ *i* ⇒ *o* ⇒ *o* ⇒ *o* ⇒ *bool*) [*show-steps*, *show-mode-inference*, *show-invalid-clauses*]
def-funtyp .

values {(*ba*,*ca*,*ta*) . *def-funtyp* (*TypQ-no-forall*) ((*Typ-fn* [*unit-typ*] *unit-typ* ((*Effect-set Nil*))))
ba ca ta}

```
fun mk-id-map :: pexp list ⇒ (id ∗ xa) list where
  mk-id-map ps = mapi (λn i. (i,mk-x (2∗n+1))) (concat (map collect-pexp ps))

inductive funcl-conv :: env ⇒ type-vars ⇒ (tannot pexp-funcl) list ⇒ s ⇒ bool where
⟦
  trace (''funcl-conv'' @ (show t));
  (i1,xa) = mk-fresh-x 0;
  pexps = List.map (λfcls. case fcls of  (PEXP-funcl pexp) ⇒ pexp) funcls;

  xm = mk-id-map pexps;

  pexp-list-conv i1 env xm pexps xa pm i4;
  trace (''funcl-conv i4=''@string-of-nat i4);
  conv-pattern-matrix i4 env xm pm [(t,xa)] sa i5;
  trace (''funcl-conv i5=''@string-of-nat i5);
  Some t = type-of-pexp (hd pexps   )

⟧ ⟹ funcl-conv env tvars funcls (AS-let xa (AE-snd (V-var (mk-farg))) sa)

code-pred (modes: i ⇒ i ⇒ i ⇒ o ⇒ bool) [show-steps, show-mode-inference, show-invalid-clauses]
funcl-conv .

inductive variant-conv :: env ⇒ type-union list ⇒ (char list  ∗ τ) list ⇒ bool where

variant-conv - Nil Nil

| ⟦
   trace (''variant-convI'' @ (show typ));
   t-conv Nil typ ta;
   variant-conv env tu-list dclist
⟧ ⟹ variant-conv env (( (Tu-ty-id typ ( (id ctor) )) ) # tu-list) ((String.explode ctor, ta) # dclist)

code-pred (modes: i ⇒ i ⇒ o ⇒ bool) [show-steps, show-mode-inference, show-invalid-clauses]
variant-conv .



fun lookup-fun-typ :: env ⇒ string ⇒ (typquant ∗ typ) option where
  lookup-fun-typ env f = get-val-spec-env env (SailAST.id f)

datatype ms-def =
  MS-type-def MiniSailAST.type-def
  | MS-fun-def MiniSailAST.fun-def
  | MS-val-spec f x b c τ
  | MS-register u τ v

definition extract-pexp :: (tannot funcl) list ⇒ (tannot pexp-funcl) list where
  extract-pexp fcls =  List.map (λpe. case pe of (FCL-Funcl - - pe) ⇒ pe) fcls

fun extract-tan ::  tannot ⇒ (tannot funcl) list ⇒ tannot where
 extract-tan tan [] = tan
| extract-tan tan ((FCL-Funcl tan' - - )#fcls) = extract-tan tan' fcls
```

inductive *def-conv* :: *env* ⇒ *tannot def* ⇒ *ms-def option* ⇒ *bool* **where**

⟦
tan′ = *extract-tan tan fcls*;
Some env = *get-env tan′*;
trace (*″def-conv-annot-none env=″* @ (*show-env env*));
Some (*tyq*,*fntyp*) = *lookup-fun-typ env f* ;
trace (*″def-conv-annot-none typ=″* @ (*show fntyp*));
def-funtyp tyq fntyp ba ca ta;
type-vars = *mk-tq-map tyq*;
pexp-funcls = *extract-pexp fcls*;
funcl-conv env type-vars (*pexp-funcl*#*pexp-funcls*) *sa* ⟧
⟹ *def-conv* - (*DEF-fundef* ((*FD-function* - *rec-opt* ((-)) *effect-opt*
((*FCL-Funcl tan* ((*id f*)) *pexp-funcl*)#*fcls*))))
(*Some* (*MS-fun-def* (*AF-fundef* (*String.explode f*) (*MiniSailAST.AF-fun-typ-none* (*AF-fun-typ*
(*mk-x 1*) *ba ca ta* (*sa*))))))

| *variant-conv env tu-list dclist* ⟹ *def-conv env*
(*DEF-type* (*TD-aux* (*TD-variant* (*SailAST.id tyid*) - *tu-list* -)))
(*Some* (*MS-type-def* (*AF-typedef* (*String.explode tyid*) *dclist*)))

| ⟦
tu-list = *List.map* (λ*i*. *Tu-ty-id unit-typ i*) *id-list*;
variant-conv env tu-list dclist
⟧ ⟹ *def-conv env*
(*DEF-type* (*TD-aux* (*TD-enum* (*SailAST.id tyid*) *id-list* -)))
(*Some* (*MS-type-def* (*AF-typedef* (*String.explode tyid*) *dclist*)))

| ⟦
def-funtyp tyq typ ba ca ta
⟧ ⟹ *def-conv env* (*DEF-spec* (*VS-aux* ((*TypSchm-ts tyq typ*) ,*id f*, -, -)))
(*Some* (*MS-val-spec* (*String.explode f*) (*mk-x 0*) *ba ca ta*))

| ⟦
t-conv [] *typ t*;
u = *mk-u 0* ;
v = *V-lit* (*L-bitvec* [])
⟧ ⟹ *def-conv env* ((*DEF-reg-dec* ((*DEC-reg* - - - *typ rid*))))
(*Some* (*MS-register u t v*))

| *def-conv env* (*DEF-overload* - -) *None*
| *def-conv env* (*DEF-default* -) *None*
| *def-conv env* (*DEF-type* (*TD-aux* (*TD-abbrev* - - -))) *None*

code-pred (*modes*: *i* ⇒ *i* ⇒ *o* ⇒ *bool*) [*show-steps*, *show-mode-inference*, *show-invalid-clauses*]

```
def-conv .

values { dc . def-conv emptyEnv (DEF-type (TD-aux (TD-variant (id (STR ''ast''))
           TypQ-no-forall [(Tu-ty-id (bool-all-typ ) (id (STR ''bob'')))] False))) dc }

values { dc . def-conv emptyEnv (DEF-type (TD-aux (TD-variant (id (STR ''ast''))
           TypQ-no-forall [(Tu-ty-id (bv-typ2 2 ) (id (STR ''bob'')))] False))) dc }

values { dc . def-conv emptyEnv (DEF-type (TD-aux (TD-variant (id (STR ''ast''))
           TypQ-no-forall [
               (Tu-ty-id (Typ-tup [unit-typ,unit-typ,unit-typ] ) (id (STR ''bob''))),
               (Tu-ty-id (Typ-tup [unit-typ,unit-typ,unit-typ] ) (id (STR ''bill'')))

] False))) dc }


values { dc. def-conv emptyEnv (DEF-spec (VS-aux (
    (TypSchm-ts (TypQ-tq [
          (QI-id (KOpt-kind K-int  (var (STR ''n'')))),
          (QI-constraint (NC-bounded-ge (Nexp-var (var (STR ''n''))) (Nexp-constant 0)))
      ])
        (Typ-fn [ Typ-app (id (STR ''atom'')) [ A-nexp (Nexp-var (var (STR ''n'')))]] int-typ
        (Effect-set []))
    ),
    (id (STR ''foo'')), (λ-. None), False))
    ) dc }


definition tq1 where
  tq1 =  (TypQ-tq [(QI-id (KOpt-kind K-int  (var (STR ''n'')))),
                   (QI-constraint (NC-bounded-ge (Nexp-var (var (STR ''n''))) (Nexp-constant 0)))
])

value mk-tq-map tq1

values { (T,G,c) . c-conv (mk-tq-map tq1)  (NC-bounded-ge (Nexp-constant 0) (Nexp-constant 0)) T
G c }

values { (T,G,c) . ce-conv (mk-tq-map tq1)  (Nexp-var (var  (STR ''n''))) T G c }

values { (T,G,c) . c-conv (mk-tq-map tq1)  (NC-bounded-ge (Nexp-var (var (STR ''n''))) (Nexp-constant
0)) T G c }

values { c . mk-tq-c (mk-tq-map tq1) tq1 c }



values {ta. t-conv [] unit-typ ta }

values {(t,g,ba,ca). t-conv-raw Nil ( (Typ-tup [unit-typ]) ) (CE-val (V-var (mk-x 1))) t g ba ca }

abbreviation
```

env1 ≡ *emptyEnv* (| *locals* := [(((*id* (*STR* ''*x*''))),*Immutable*, (*bool-typ True*))] , *typ-vars* := [((*var STR* ''*n*'') ,
K-int)] |)

abbreviation *tan1* ≡ (*add-local* (*set-type None unit-typ*) ((*id* (*STR* ''*x*''))) (*unit-typ*))

abbreviation *tan2* ≡ (*add-local* (*set-type None int-typ*) ((*id* (*STR* ''*x*''))) (*unit-typ*))

abbreviation *pexp1* ≡ ((*Pat-exp unk* ((*P-id* (*set-type None unit-typ*) ((*id* (*STR* ''*x*'')))) ((*E-id tan1* ((*id* (*STR* ''*x*''))))
)))

abbreviation *pexp2* ≡ ((*Pat-exp unk* ((*P-typ* (*set-type None unit-typ*) *unit-typ*
(*P-id* (*set-type None unit-typ*) ((*id* (*STR* ''*x*'')))))
(*E-block tan1* [(*E-lit tan1* (*SailAST.L-unit*))])))

abbreviation *pexp3* ≡ ((*Pat-exp unk* ((*P-typ* (*set-type None unit-typ*) *unit-typ*
(*P-id* (*set-type None unit-typ*) ((*id* (*STR* ''*x*'')))))
(*E-block tan2* [(*E-lit tan1* (*SailAST.L-num 42*))])))

abbreviation *funcl1* ≡ (*FCL-Funcl unk* ((*id* (*STR* ''*f*''))) (*PEXP-funcl pexp1*))

abbreviation *funcl2* ≡ (*FCL-Funcl unk* ((*id* (*STR* ''*f*''))) (*PEXP-funcl pexp2*))

abbreviation *funcl3* ≡ (*FCL-Funcl unk* ((*id* (*STR* ''*f*''))) (*PEXP-funcl pexp3*))

values { (*va*,*t*,*g*) . *v-conv env1* [] ((*E-id tan1* ((*id* (*STR* ''*x*''))))) *t g va* }

values { (*t*,*p*,*b*,*g*,*d*,*sa*,*i*) . *s-conv 0* [] ((*E-id tan1* ((*id* (*STR* ''*x*''))))) *t p b g d sa i* }

values { (*pm*,*i*) . *pexp-list-conv 0 emptyEnv* [] [*pexp1*] (*mk-x 1*) *pm i* }

values { *sa* . *funcl-conv emptyEnv* [] [(*PEXP-funcl pexp1*)] *sa* }

values { *fa* . *def-conv emptyEnv* (*DEF-fundef* ((*FD-function unk Rec-nonrec* ((*Typ-annot-opt-some*
((*TypQ-tq* [(*QI-id* ((*KOpt-kind* (*K-int*) ((*var* (*STR* ''*x*''))))))]))
((*Typ-fn* [*unit-typ*] *unit-typ* ((*Effect-set Nil*)))))) (*Effect-opt-none*) [*funcl1*]))) *fa* }

values { *fa* . *def-conv emptyEnv* (*DEF-fundef* ((*FD-function unk Rec-nonrec*
(*Typ-annot-opt-some TypQ-no-forall unit-typ*) *Effect-opt-none*
[*funcl2*]))) *fa* }

values { *fa* . *def-conv emptyEnv* (*DEF-fundef* ((*FD-function unk Rec-nonrec*
(*Typ-annot-opt-some TypQ-no-forall int-typ*) *Effect-opt-none*
[*funcl3*]))) *fa* }

abbreviation *pexp4* ≡ ((*Pat-exp unk*
(*P-tup* (*set-type None* (*Typ-tup* [*int-typ*,*int-typ*]))
[*P-id* (*set-type None int-typ*) ((*id* (*STR* ''*x*'')),
P-id (*set-type None int-typ*) ((*id* (*STR* ''*y*'')))])

```
( E-block tan2  [(E-if tan1 (E-lit tan1 SailAST.L-true)
    (E-lit tan1 (SailAST.L-num 42)) (E-lit tan1 (SailAST.L-num 42))
                )] )))
```

```
abbreviation env4 ≡ emptyEnv (| top-val-specs :=
                  [ (id (STR ''f''), TypQ-no-forall ,
              ( (Typ-fn [int-typ,int-typ] int-typ ( (Effect-set Nil))))) ] |)

abbreviation tan4 ≡ (| tannot-env = env4 ,   tannot-typ = int-typ,
       tannot-effect = ( (Effect-set [])), tannot-expected = None, tannot-instantiations = None  |)

abbreviation funcl4 ≡ (FCL-Funcl (Some tan4) ( (id (STR ''f'')) ) (PEXP-funcl pexp4))

value collect-ids funcl4


values { fa . def-conv env4 (DEF-fundef ( (FD-function unk Rec-nonrec
 (Typ-annot-opt-none) Effect-opt-none
   [funcl4]) )) fa }

value mk-fresh-many 0 2

value int-typ
value unit-typ

values { fa . def-conv emptyEnv (DEF-fundef ( (FD-function unk Rec-nonrec  ( (Typ-annot-opt-some
  TypQ-no-forall
 ( (Typ-fn [int-typ] int-typ ( (Effect-set Nil)  )) )  ) ) ( Effect-opt-none ) [funcl1]) )) fa }
```

17.10 Programs

17.11 Examples

```
values { G. g-conv env1 []  G }

values { (sa,T,P,B,G,D,i) . s-conv 0  [( id STR ''x'', mk-x 1)]
         (E-id
         (add-local (set-type None (bool-typ True)) ( (id (STR ''x''))) (bool-typ True)  )
         (id (STR ''x'')) )
         T P B G D sa i}


values {  (sa,T,P,B,G,D,i) . s-conv 0  [( id STR ''x'', mk-x 1)]
             ( (E-id
                 (add-local (set-type None (bool-typ True)) ( (id (STR ''x''))) (bool-typ True)  )
                ( (id (STR ''xx'')) )))
    T P B G D sa i }
```

values { (*sa*,*T*,*P*,*B*,*G*,*D*,*i*) . *s-conv 0* [(*id STR ''x''*, *mk-x 1*)]
(*E-case* (*set-type None* (*bool-typ True*)) *e-unit* [*Pat-exp unk* (*pat-id* (*STR ''x''*)) *e-true*])
T P B G D sa i }

values { (*sa*,*T*,*P*,*B*,*G*,*D*,*i*) . *s-conv 0* [] (*e-let* (*bv-lit* (*STR ''0101''*)) (*bv-typ 4*)) *T P B G D sa i* }

values { (*sa*,*T*,*P*,*B*,*G*,*D*,*i*) . *s-conv 0* []
(*bv-lit* (*STR ''0101''*))
T P B G D sa i }

definition *match1* **where**
match1 = *E-case int-tannot* (*bv-lit* (*STR ''0101''*)) [
Pat-exp unk (*pat-lit-bv* (*STR ''0101''*)) *e-true*,
Pat-exp unk (*pat-lit-bv* (*STR ''0100''*)) *e-false*]

values { (*sa*,*T*,*P*,*B*,*G*,*D*,*i*) . *s-conv 0* [] *match1 T P B G D sa i* }

value *x-pp* (*mk-x 0*)

end
theory *Validator*
imports *SailEnv Native-Word.Uint32 SailASTUtils ShowAST HOL−Library.Debug*
begin

Chapter 18

Sail Validator

18.1 Wellformedness

Use mutual recursion to make this align with MiniSail's equivalent

inductive *wfNE* :: *env* ⇒ *nexp* ⇒ *kind* ⇒ *bool* **and**
wfNC :: *env* ⇒ *n-constraint* ⇒ *bool* **and**
wfTyp :: *env* ⇒ *typ* ⇒ *bool* **and**
wfLocals :: *env* ⇒ (*id*∗*mut*∗*typ*) *list* ⇒ *bool* **and**
wfE :: *env* ⇒ *bool*
where

wfNE-constI: *wfNE env* ((*Nexp-constant n*)) *K-int*

| *wfNE-timesI*:⟦
wfNE env nexp1 (*K-int*);
wfNE env nexp2 (*K-int*)
⟧ ⟹ *wfNE env* ((*Nexp-times nexp1 nexp2*)) (*K-int*)

| *wfNE-sumI*:⟦
wfNE env nexp1 (*K-int*);
wfNE env nexp2 (*K-int*)
⟧ ⟹ *wfNE env* ((*Nexp-sum nexp1 nexp2*)) (*K-int*)

| *wfNE-minusI*:⟦
wfNE env nexp1 (*K-int*);
wfNE env nexp2 (*K-int*)
⟧ ⟹ *wfNE env* ((*Nexp-minus nexp1 nexp2*)) (*K-int*)

| *wfNE-kid*: ⟦ *Some* (*kind*) = *lookup* (*typ-vars env*) *kid*
⟧ ⟹ *wfNE env* ((*Nexp-var kid*)) *kind*

| *wfNC-eqI*: ⟦ *wfNE env nexp1 k1* ; *wfNE env nexp2 k2* ⟧ ⟹ *wfNC env* ((*NC-equal nexp1 nexp2*))

| *wfTyp-idI*: *Some kd* = *lookup* (*typ-vars env*) *x* ⟹ *wfTyp env* ((*Typ-var x*))
| *wfTyp-fnI*: *wfTyp env* ((*Typ-fn typs-in typ-ret* -))
| *wfTyp-bidirI*: *wfTyp env* ((*Typ-bidir t1 t2* -))
| *wfTyp-tup*: *wfTyp env* ((*Typ-tup typs*))
| *wfTyp-app*: *wfTyp env* ((*Typ-app x args*))

| *wfTyp-exist*: *wfTyp env* ((*Typ-exist kids nc t*))

| *wfLocals-nilI*: *wfLocals env* []
| *wfLocals-consI*: ⟦
wfTyp env typ;
wfLocals env locs
⟧ ⟹ *wfLocals env* ((*x*,(*mut*,*typ*))#*locs*)

| *wfEI*: *wfLocals env* (*locals env*) ⟹ *wfE env*

code-pred (*modes*:
wfNE: *i* ⇒ *i* ⇒ *o* ⇒ *bool* **and**
wfNC: *i* ⇒ *i* ⇒ *bool* **and**
wfTyp : *i* ⇒ *i* ⇒ *bool* **and**
wfLocals : *i* ⇒ *i* ⇒ *bool* **and**
wfE : *i* ⇒ *bool*) [*show-steps*, *show-mode-inference*, *show-invalid-clauses*] *wfNC* .

18.2 Subtyping

hide-const *id*

fun *env-of* :: *tannot exp* ⇒ *env option* **where**
env-of exp = *get-env* (*annot-e exp*)

fun *type-of-exp-lb* :: *tannot letbind* ⇒ *typ option* **where**
type-of-exp-lb lb = *get-type* (*annot-letbind lb*)

fun *env-of-lb* :: *tannot letbind* ⇒ *env option* **where**
env-of-lb lb = *get-env* (*annot-letbind lb*)

inductive *eq-id* :: *id* ⇒ *id* ⇒ *bool* **where**
eq-id ((*id x*)) ((*id y*))

inductive *eq-kid* :: *kid* ⇒ *kid* ⇒ *bool* **where**
eq-kid ((*var x*)) ((*var y*))

definition *nc-and* :: *n-constraint* ⇒ *n-constraint* ⇒ *n-constraint* **where**
nc-and nc1 nc2 ≡ (*NC-and nc1 nc2*)

definition *nc-or* :: *n-constraint* ⇒ *n-constraint* ⇒ *n-constraint* **where**
nc-or nc1 nc2 ≡ (*NC-or nc1 nc2*)

definition *mk-id* **where** *mk-id x* = (*id* (*String.implode x*))

definition *arg-bool* **where** *arg-bool nc* = (*A-bool nc*)

definition *nc-not* :: *n-constraint* ⇒ *n-constraint* **where**
nc-not nc ≡ (*NC-app* (*mk-id* ''*not*'') [*arg-bool nc*])

definition *nc-between* :: *nexp* ⇒ *nexp* ⇒ *nexp* ⇒ *n-constraint* **where**
nc-between n1 n n2 = *nc-and* ((*NC-bounded-le n1 n*)) ((*NC-bounded-ge n n1*))

definition *nc-bool-equiv* :: *n-constraint* ⇒ *n-constraint* ⇒ *n-constraint* **where**
nc-bool-equiv nc1 nc2 = (*nc-or* (*nc-and nc1 nc2*) (*nc-and* (*nc-not nc1*) (*nc-not nc2*)))

inductive
match :: *typ* ⇒ *typ* ⇒ *n-constraint list* ⇒ *bool* **and**
match-list :: *typ list* ⇒ *typ list* ⇒ *n-constraint list* ⇒ *bool* **and**
match-arg :: *typ-arg* ⇒ *typ-arg* ⇒ *n-constraint list* ⇒ *bool* **and**
match-arg-list :: *typ-arg list* ⇒ *typ-arg list* ⇒ *n-constraint list* ⇒ *bool* **and**
match-nc :: *n-constraint* ⇒ *n-constraint* ⇒ *n-constraint list* ⇒ *bool* **and**
match-nexp :: *nexp* ⇒ *nexp* ⇒ *n-constraint list* ⇒ *bool*
where

match-arg-typ: *match t1 t2 ms* ⟹ *match-arg* ((*A-typ t1*)) ((*A-typ t2*)) *ms*

| *match-arg-nexpI*: *match-nexp ne1 ne2 ms* ⟹ *match-arg* ((*A-nexp ne1*)) ((*A-nexp ne2*)) *ms*

| *match-arg-ncI*: *match-nc nc1 nc2 ms* ⟹ *match-arg* ((*A-bool nc1*)) ((*A-bool nc2*)) *ms*

| *match-arg-orderI*: *match-arg* ((*A-order ord*)) ((*A-order ord*)) []

| *match-arg-list-nilI*: *match-arg-list* [] [] []

| *match-arg-list-consI*:
⟦
match-arg a1 a2 ms1;
match-arg-list as1 as2 ms2
⟧ ⟹ *match-arg-list* (*a1*#*as1*) (*a2*#*as2*) ((*ms1*@*ms2*))

| *match-appI*: ⟦ *match-arg-list args1 args2 ms* ; *eq-id id1 id2*
⟧ ⟹ *match* ((*Typ-app id1 args1*)) ((*Typ-app id2 args2*)) *ms*

| *match-app1I*: ⟦ *eq-id id1 id2*
⟧ ⟹ *match* ((*Typ-id id1*)) ((*Typ-app id2* [])) []

| *match-app2I*: ⟦ *eq-id id1 id2*
⟧ ⟹ *match* ((*Typ-app id1* [])) ((*Typ-id id2*)) []

| *match-idI*: ⟦ *eq-id id1 id2*
⟧ ⟹ *match* ((*Typ-id id1*)) ((*Typ-id id2*)) []

| *match-varI*: ⟦ *eq-kid kid1 kid2*
⟧ ⟹ *match* ((*Typ-var kid1*)) ((*Typ-var kid2*)) []

| *match-tupleI*: ⟦ *match-list ts1 ts2 bs*
⟧ ⟹ *match* ((*Typ-tup ts1*)) ((*Typ-tup ts2*)) *bs*

| *match-intI*:
match ((*Typ-id* ((*id* (*STR ''int''*))))) ((*Typ-app* ((*id* (*STR ''atom''*))) -)) []

| *match-intI2*:
match ((*Typ-app* ((*id* (*STR ''atom''*))) -))
((*Typ-id* ((*id* (*STR ''int''*))))) [(*NC-true*)]

| *match-nat1I*:
match ((*Typ-app* ((*id* (*STR ''atom''*))) [(*A-nexp nexp*)])) ((*Typ-id* ((*id* (*STR ''nat''*))))) [*nc-pos nexp*]

| *match-nat2I*:
match ((*Typ-id* ((*id* (*STR ''nat''*))))) ((*Typ-app* ((*id* (*STR ''atom''*))) -)) []

| *match-range1I*:
match ((*Typ-app* ((*id* (*STR ''atom''*))) [((*A-nexp ne*))]))
((*Typ-app* ((*id* (*STR ''range''*))) [((*A-nexp ne1*)),((*A-nexp ne2*))])) [*nc-between ne1 ne ne2*]

| *match-range2I*:
match
((*Typ-app* ((*id* (*STR ''range''*))) [((*A-nexp ne1*)),((*A-nexp ne2*))]))
((*Typ-app* ((*id* (*STR ''atom''*))) [((*A-nexp ne*))]))
[*nc-between ne1 ne ne2*]

| *match-range3I*:
match
((*Typ-app* ((*id* (*STR ''range''*))) [((*A-nexp ne1*)),((*A-nexp ne2*))]))
((*Typ-id* ((*id* (*STR ''int''*)))))
[*NC-true*]

| *match-boolI1*:
match ((*Typ-app* ((*id* (*STR ''atom-bool''*))) -)) ((*Typ-id* ((*id* (*STR ''bool''*))))) [(*NC-true*)]

| *match-bool21*:
match ((*Typ-id* ((*id* (*STR ''bool''*))))) ((*Typ-app* ((*id* (*STR ''atom-bool''*))) [(*A-bool nc*)])) [*nc*]

| *match-list-nilI*: *match-list* [] [] []

| *match-list-consI*: ⟦
  match t1 t2 b;
  match-list ts1 ts2 bs
⟧ ⟹ *match-list* (*t1*#*ts1*) (*t2*#*ts2*) (*b*@*bs*)

| *match-nexp*: *match-nexp ne1 ne2* [(*NC-equal ne1 ne2*)]

| *match-ncI*: *match-nc nc1 nc2* [(*nc-bool-equiv nc1 nc2*)]

inductive *normalise* :: *env* ⇒ *typ* ⇒ *typ* ⇒ *bool* **where**
normalise env ((*Typ-exist x y z*)) (*Typ-exist x y z*)
| *normalise env* ((*Typ-id i*)) (*Typ-exist* [] (*NC-true*) ((*Typ-id i*)))
| *normalise env* ((*Typ-var k*)) (*Typ-exist* [] (*NC-true*) ((*Typ-var k*)))
| *normalise env* ((*Typ-tup ts*)) (*Typ-exist* [] (*NC-true*) ((*Typ-tup ts*)))
| *normalise env* ((*Typ-app idd tas*)) (*Typ-exist* [] (*NC-true*) ((*Typ-app idd tas*)))

18.3 Printing

fun *trace* :: *char list* ⇒ *bool* **where**
  trace s = (*let* - = *Debug.trace* (*String.implode s*) *in True*)

fun *nc-and-list* :: *n-constraint list* ⇒ *n-constraint* **where**
nc-and-list ncs = *List.fold nc-and ncs* (*NC-true*)

inductive *subtype* :: *env* ⇒ *typ* ⇒ *typ* ⇒ *bool* **where**
⟦
  normalise env t1 (*Typ-exist x1 y1 t1'*) ;
  normalise env t2 (*Typ-exist x2 y2 t2'*) ;
  trace (*''t1=''* @ *show t1*);
  trace (*''t2=''* @ *show t2*);
  match t1' t2' bs;
  trace (*''ncs=''* @ (*List.concat* (*List.map show bs*))) ;
  prove env (*nc-and-list bs*)
⟧ ⟹ *subtype env t1 t2*

inductive *subtype-exp* :: *tannot exp* ⇒ *typ* ⇒ *bool* **where**
⟦
    Some env = *get-env-exp exp*;
    Some typ1 = *type-of-exp exp*;
    subtype env typ1 typ2

$]\!] \Longrightarrow$ *subtype-exp exp typ2*

code-pred (*modes*:
subtype : $i \Rightarrow i \Rightarrow i \Rightarrow$ *bool* **and**
subtype-exp : $i \Rightarrow i \Rightarrow$ *bool* **and**
match : $i \Rightarrow i \Rightarrow o \Rightarrow$ *bool* **and**
match-list : $i \Rightarrow i \Rightarrow o \Rightarrow$ *bool* **and**
match-nc : $i \Rightarrow i \Rightarrow o \Rightarrow$ *bool* **and**
match-arg : $i \Rightarrow i \Rightarrow o \Rightarrow$ *bool* **and**
match-arg-list : $i \Rightarrow i \Rightarrow o \Rightarrow$ *bool*

) [*show-steps*, *show-mode-inference*, *show-invalid-clauses*] *subtype* **.**

value *bool-typ True*

values {*ms* . *match unit-typ unit-typ ms*}

values {*ms* . *match* (*bool-typ True*) (*bool-typ True*) *ms*}

values {*ms* . *match-nexp* ((*Nexp-constant 10*)) ((*Nexp-constant 10*)) *ms* }

values {*ms* . *match-arg-list* [((*A-nexp* ((*Nexp-constant 10*)))] [((*A-nexp* ((*Nexp-constant 10*)))] *ms* }

values {*ms* . *match* ((*Typ-app* ((*id* (*STR ''atom''*))) [(*A-nexp* ((*Nexp-constant 10*)))])) ((*Typ-app* ((*id* (*STR ''atom''*))) [(*A-nexp* ((*Nexp-constant 11*)))])) *ms* }

values {*ms* . *match* ((*Typ-app* ((*id* (*STR ''atom-bool''*))) [*arg-bool* (*NC-true*)])) ((*Typ-app* ((*id* (*STR ''atom-bool''*))) [*arg-bool* (*NC-false*)])) *ms* }

18.4 Checking

18.4.1 Literals

fun *integer-of-int2* :: *int* $\Rightarrow$ *integer* **where**
integer-of-int2 x = *integer-of-int x*

inductive *check-lit* :: *env* $\Rightarrow$ *lit* $\Rightarrow$ *typ* $\Rightarrow$ *bool* **where**
check-lit-unitI: *check-lit env* (*L-unit*) ((*Typ-id* ((*id* (*STR ''unit''*)))))

| *check-lit-numI*: *check-lit env* ((*L-num num*)) ((*Typ-app* ((*id* (*STR ''atom''*)))
[((*A-nexp* ((*Nexp-constant num*))))]))

| *check-lit-trueI*: *check-lit env* (*L-true*) ((*Typ-app* ((*id* (*STR ''atom-bool''*)))
[((*A-bool* (*NC-true*)))]))

| *check-lit-falseI*: *check-lit env* (*L-false*) ((*Typ-app* ((*id* (*STR ''atom-bool''*)))
[((*A-bool* (*NC-false*)))]))

| *check-lit-bitoneI*: *check-lit env* (*L-one*) ((*Typ-id* ((*id* (*STR ''bit''*)))))

```
| check-lit-bitzeroI: check-lit env ( L-zero ) ( (Typ-id ( (id (STR ''bit'')) ) ))


| check-lit-binI: check-lit env ( (L-bin s) ) ( (Typ-app ( (id (STR ''bitvector'')) )
    [( (A-nexp ( (Nexp-constant (integer-of-int2 ( (int (length (String.explode s)))))) )) ) , (A-order (
- )) ]) )


| check-lit-hexI: check-lit env ( (L-hex s) ) ( (Typ-app ( (id (STR ''bitvector'')) )
   [( (A-nexp ( (Nexp-constant (4*( (integer-of-int2 (int (length (String.explode s)))))))) )) ) , (A-order
( - )) ]) )

| check-lit-stringI: check-lit env ( (L-string -)) ( (Typ-id ( (id (STR ''string'')))))

code-pred (modes: i ⇒ i ⇒ i ⇒ bool) check-lit .

values { True. check-lit emptyEnv ( L-unit ) unit-typ }

values { True. check-lit emptyEnv ( (L-num 43) ) (num-typ 43) }




fun print-type :: typ ⇒ bool where
  print-type - = True

code-printing
  constant subtype ⇀ (OCaml) Type'-check.subtype'-check

code-printing
  constant print-type ⇀ (OCaml) Utils2.print'-type

type-synonym bindings = (id*mut*typ) list

fun locals-in :: env ⇒ bindings ⇒ bool where
  locals-in - [] = True
| locals-in env ((x,mut,typ)#gs) = (case lookup-local-id-env env x of
                                    Some t ⇒ locals-in env gs | None ⇒ False)
```

18.4.2 Patterns

```
inductive check-pat :: tannot pat ⇒ bindings ⇒ bool ( ⊢ - ⇝ - ) and
  check-pat-list :: tannot pat list ⇒ bindings ⇒ bool
  where
 check-pat-litI: ⟦ Some (env,t) = get tan ; check-lit env lit t ⟧ ⟹ check-pat ( (P-lit tan lit)) []

| check-pat-wildI: check-pat ( P-wild - ) []
| check-pat-orI: ⟦
   check-pat p1 bs1;
   check-pat p2 bs2
⟧ ⟹ check-pat ( (P-or - p1 p2) ) (bs1@bs2)
```

```
| check-pat-notI: ⟦
   check-pat p1 bs1
⟧ ⟹ check-pat ( (P-not - p1)  ) (bs1)

| check-pat-asI:check-pat pat bindings ⟹ check-pat ( (P-as  - pat -)  ) bindings

| check-pat-typI:check-pat pat bindings ⟹ check-pat ( (P-typ - - pat)  ) bindings

| check-pat-idI: ⟦
      Some (-,t) = get tan;
      None = lookup-enum tan x
 ⟧ ⟹ check-pat ( (P-id tan x)  ) [(x,Immutable,t)]

| check-pat-enumI: ⟦
      Some (env,t1) = get tan;
      Some t2 = lookup-enum tan x ;
      subtype env t2 t1
⟧ ⟹ check-pat ( (P-id tan x)  ) []


| check-pat-varI:⟦
   check-pat pat bindings
⟧ ⟹ check-pat ( (P-var - pat - )  ) bindings


| check-pat-appI:⟦
   check-pat parg bindings
⟧ ⟹ check-pat ( (P-app - ctor [ parg ])  ) bindings

| check-pat-vectorI: ⟦
   check-pat-list pats bs
⟧ ⟹ check-pat ( (P-vector - pats) ) bs

| check-pat-vector-concatI: ⟦
   check-pat-list pats bs
⟧ ⟹ check-pat ( (P-vector-concat - pats) ) bs

| check-pat-tupI: ⟦
   check-pat-list pat-list bindings
⟧ ⟹ check-pat ( (P-tup - pat-list) ) bindings

| check-pat-listI: ⟦
   check-pat-list pat-list bindings
⟧ ⟹ check-pat ( (P-list - pat-list) ) bindings

| check-pat-consI: ⟦
   check-pat p1 bs1;
   check-pat p2 bs2
⟧ ⟹ check-pat (  (P-cons - p1 p2)  ) (bs1@bs2)

| check-pat-string-appendI: ⟦
   check-pat-list pat-list bindings
```

$\rrbracket \Longrightarrow$ *check-pat* ((*P-string-append* - *pat-list*)) *bindings*

| *check-pat-list-nilI*: *check-pat-list* [] []
| *check-pat-list-consI*: ⟦
check-pat pat bindings1;
check-pat-list pats bindings2
⟧ ⟹ *check-pat-list* (*pat*#*pats*) (*bindings1*@*bindings2*)

code-pred (*modes*: *i* ⇒ *o* ⇒ *bool*) [*show-steps*, *show-mode-inference*, *show-invalid-clauses*] *check-pat*
.

The type we get from the type annotation on a node is a subtype of the supplied type

inductive *subtype-tan* :: *typ* ⇒ *tannot* ⇒ *bool* **where**
⟦ *Some env* = *get-env tan* ; *Some t'* = *get-type tan* ; *subtype env t t'* ⟧ ⟹ *subtype-tan t tan*

18.4.3 L-values

18.4.4 Expressions

inductive *check-lexp-vector-list* :: (*tannot lexp*) *list* ⇒ *order* ⇒ *typ* ⇒ *bool* **where**

check-lexp-vector-list-nilI: *check-lexp-vector-list* [] - -

| *check-lexp-vector-list-consI*: ⟦
Some t = *type-of-lexp lexp*;
Some (-, *order*, *typ*) = *deconstruct-vector-type t*;
check-lexp-vector-list lexps order typ
⟧ ⟹ *check-lexp-vector-list* (*lexp*#*lexps*) *order typ*

inductive *check-local-binds* :: (*tannot exp*) *list* ⇒ *bindings* ⇒ *bool* **where**
check-local-binds [] -

| ⟦ *Some env* = *get-env-exp exp*;
locals-in env bindings;
check-local-binds exps bindings
⟧ ⟹ *check-local-binds* (*exp*#*exps*) *bindings*

fun *add-locals* :: *env* ⇒ *bindings* ⇒ *env* **where**
add-locals env - = *env*

Second environment contains at the least the bindings that the first does and the constraints FIXME

inductive *subenv* :: *env* ⇒ *env* ⇒ *bool* **where**

subenv e1 e2

inductive *check-exp* :: *tannot exp* ⇒ *bindings* ⇒ *bool* (⊢ - ⇝ -) **and**
check-exp-typ :: *tannot exp* ⇒ *typ* ⇒ *bool* (⊢ - : -) **and**

check-exp-typ-env :: *env* ⇒ *tannot exp* ⇒ *typ* ⇒ *bool* (- ⊢ - : -) **and**
check-exp-list :: (*tannot exp*) *list* ⇒ *typ list* ⇒ *bool* **and**
check-letbind :: *tannot letbind* ⇒ *bindings* ⇒ *bool* (⊢ - ⇝ -) **and**
check-fexp :: *tannot fexp* ⇒ *typ* ⇒ *bool* (⊢ - <: -) **and**
check-fexp-list :: *tannot fexp list* ⇒ *typ* ⇒ *bool* **and**
check-pexp :: *tannot pexp* ⇒ *bool* (⊢ -) **and**
check-pexps :: *tannot pexp list* ⇒ *bool* (⊢ -) **and**
check-lexp :: *tannot lexp* ⇒ *bindings* ⇒ *bool* (⊢ - ⇝ -) **and**
check-lexp-list :: (*tannot lexp*) *list* ⇒ *bindings* ⇒ *bool* **and**
check-exp-bindings :: *tannot exp* ⇒ *bindings* ⇒ *bool*
where

check-lexp-id-notbI:⟦
Some (-,*t*) = *get tan*;
trace ''*check-lexp-id-notbI*'';
None = *lookup-mutable tan x*
⟧ ⟹ ⊢ ((*LEXP-id tan x*)) ⇝ [(*x*,*Mutable*,*t*)]

| *check-lexp-id-bI*:⟦
Some (*env*,*t1*) = *get tan*;
trace ''*check-lexp-id-bI*'';
Some t2 = *lookup-mutable tan x*;
trace (''*check-lexp-id-bI found mut*'' @ *show t2*);
subtype env t1 t2;
trace (''*check-lexp-id-bI subtype ok*'')
⟧ ⟹ ⊢ ((*LEXP-id tan x*)) ⇝ []

| *check-lexp-cast-notbI*:⟦
Some (*env*,*t'*) = *get tan*;
None = *lookup-mutable tan x*;
subtype env t2 t'
⟧ ⟹ ⊢ ((*LEXP-cast tan t2 x*)) ⇝ [(*x*,*Mutable*,*t2*)]

| *check-lexp-cast-bI*:⟦
Some (*env*,*t'*) = *get tan*;
Some t'' = *lookup-mutable tan x*;
subtype env t t'';
subtype env t' t
⟧ ⟹ *check-lexp* ((*LEXP-cast tan t x*)) []

| *check-lexp-derefI*: ⟦
Some (*env*,*t2*) = *get tan*;
check-exp exp bs;
Some t = *type-of-exp exp*;
Some t1 = *deconstruct-register-type t*;
subtype-tan t1 tan
⟧ ⟹ *check-lexp* ((*LEXP-deref tan exp*)) []

| *check-lexp-list-nilI*: ⟦
trace ''*check-lexp-list-nilI*''
⟧ ⟹

```
   check-lexp-list [] []

| check-lexp-list-consI: ⟦
    trace ''check-lexp-list-consI'';
    check-lexp lexp binding;
    check-lexp-list lexps bindings
⟧ ⟹   check-lexp-list (lexp#lexps) (binding@bindings)


| check-lexp-tupI: ⟦
    trace ''check-lexp-tupI'';
    Some (env,t2) = get tan;
    check-lexp-list lexps bindings;
    Some ts1 = those (List.map (λl. type-of-lexp l) lexps);
    trace (''check-lexp-tupI types='' @ show ts1);
    subtype-tan ( (Typ-tup ts1) ) tan
⟧ ⟹ check-lexp ( (LEXP-tup tan lexps)) bindings


| check-lexp-vector-concatI: ⟦
    trace ''check-lexp-vector-concatI'';
    Some (nexp, order, typ) = is-vector-type tan;
    check-lexp-list lexps bindings;
    check-lexp-vector-list lexps order typ
⟧ ⟹ check-lexp ( (LEXP-vector-concat tan lexps)) bindings


| check-lexp-vectorI: ⟦
   trace ''check-lexp-vectorI'';
   Some (-,typ) = get tan;
   check-lexp lexp bindings;
   trace ''check-lexp-vectorI 2'';
   Some t = type-of-lexp lexp;
   trace (''check-lexp-vectorI vectype='' @ show t);
   Some (nexp', order, typ) = deconstruct-vector-type t;
   trace (''check-lexp-vectorI typ='' @ show typ);
  ⊢ exp : int-typ
⟧ ⟹ check-lexp ( (LEXP-vector tan lexp exp) ) bindings

| check-lexp-vector-rangeI: ⟦
    Some (-,t2) = get tan;
   ⊢ exp1 : int-typ;
   ⊢ exp2 : int-typ;
    check-lexp lexp bindings ;
    Some t1 = type-of-lexp lexp;
    Some (nexp', order, typ) = deconstruct-vector-type t1;
    Some (nexp , order, typ) = deconstruct-vector-type t2
⟧ ⟹ check-lexp ( (LEXP-vector-range tan lexp exp1 exp2) ) []



| check-fexpI: ⟦
```

Some recid = deconstruct-record-type rtyp;
Some env = get-env-exp exp;
Some t2 = lookup-record-field-env env recid x;
check-exp-typ exp t2
⟧ ⟹ *check-fexp* ((*FE-Fexp tan x exp*)) *rtyp*

| *check-fexp-list-nillI*: *check-fexp-list* [] -

| *check-fexp-list-consI*: ⟦
check-fexp fexp typ;
check-fexp-list fexp-list typ
⟧ ⟹ *check-fexp-list* (*fexp*#*fexp-list*) *typ*

| *check-lexp-fieldI*: *check-lexp* ((*LEXP-field tan lexp fid*)) []

| *check-litI*: ⟦
Some (*env*,*t*) = *get tan* ;
check-lit env lit t
⟧ ⟹ ⊢ ((*E-lit tan lit*)) ⇝ []

| *check-idI*: ⟦
trace ''*check-idI*'';
Some (*env*,*t2*) = *get tan*;
Some t1 = lookup-id tan x;
subtype env t1 t2
⟧ ⟹ ⊢ ((*E-id tan x*)) ⇝ []

| *check-tupleI*: ⟦
Some (-,*t*) = *get tan*;
((*Typ-tup typs*)) = *t*;
check-exp-list exps typs
⟧ ⟹ ⊢ ((*E-tuple tan exps*)) ⇝ []

| *check-castI*: ⟦
check-exp exp bs;
subtype-exp exp t
⟧ ⟹ ⊢ ((*E-cast tan t exp*)) ⇝ *bs*

| *check-exp-env-typI*:
⟦ ⊢ *exp* ⇝ *bs*;
Some (*e*,*t*) = *get-e exp*;
subtype env t typ;
subenv env e
⟧ ⟹ *check-exp-typ-env env exp typ*

| *check-exp-typI*:
⟦ ⊢ *exp* ⇝ *bs*;

```
    subtype-exp  exp typ
⟧ ⟹ check-exp-typ exp typ

| check-exp-list-nilI: check-exp-list  [] []

| check-exp-list-consI: ⟦
  check-exp-typ  exp typ;
  check-exp-list  exps typs
⟧ ⟹ check-exp-list (exp#exps) (typ#typs)


| check-appI: ⟦
    Some (env,t) = get tan;
    Some (in-typs,rett-typ ) = lookup-fun tan fid;
    trace (''E-app '' @ (show-tannot tan));
    Some in-typs2 = subst-inst-list tan in-typs;
    trace ''E-app after subst-inst in args'';
    check-exp-list exps in-typs2;
    Some ret-typ2 = subst-inst tan rett-typ;
    trace ''E-app after subst-inst ret'';
    trace (''E-app subtype'' @ ''t1='' @ show ret-typ2 @ ''t2='' @ show t);
    subtype-tan ret-typ2 tan
⟧ ⟹ check-exp ( (E-app tan fid exps)  ) []



| check-recordI: ⟦
  Some (-,typ) = get tan;
  check-fexp-list fexp-list typ
⟧ ⟹ check-exp ( (E-record tan fexp-list)  ) []

| check-record-updateI: ⟦
 ⊢ exp ⇝ bs ;
 check-fexp-list fexp-list typ;
 Some (-,typ) = get tan
⟧ ⟹ check-exp ( (E-record-update tan exp fexp-list)  ) []

| check-fieldI: ⟦
  Some (env,t1) = get tan;
  ⊢ exp ⇝ bs;
  Some rtype = type-of-exp exp;
  Some recid = deconstruct-record-type rtype;
  Some t2 = lookup-record-field-env env recid fid;
  subtype env t1 t2
⟧ ⟹ check-exp ( (E-field tan exp fid)  ) []



| check-returnI: ⟦
  Some r-typ = ret-type tan;
  check-exp-typ exp r-typ
```

⟧ ⟹ *check-exp* ((*E-return tan exp*)) []

| *check-exitI*: ⟦
  check-exp-typ exp unit-typ
⟧ ⟹ *check-exp* ((*E-exit tan exp*)) []

| *check-throwI*: ⟦
  check-exp-typ exp ((*Typ-id* ((*id* (*STR ''exception''*)))))
⟧ ⟹ *check-exp* ((*E-throw tan exp*)) []

| *check-tryI*: ⟦
  check-exp exp bs;
  check-pexps pexps
⟧ ⟹ *check-exp* ((*E-try tan exp pexps*)) []

| *check-refI*: ⟦
  Some t1 = *lookup-register tan x*;
  subtype-tan t1 tan
⟧ ⟹ *check-exp* ((*E-ref tan x*)) []

| *check-vectorI*: ⟦
  Some (*env*,-) = *get tan*;
  Some (*len*,*ord*,*typ*) = *is-vector-type tan*;
  check-exp-list exps (*replicate* (*length exps*) *typ*);
  prove env (*nc-eq* (*nint* (*integer-of-int2* (*int* (*length exps*)))) (*len*))
⟧ ⟹ *check-exp* ((*E-vector tan exps*)) []

| *check-listI*: ⟦
  Some elem-typ = *is-list-type tan*;
  check-exp-list exps (*replicate* (*length exps*) *elem-typ*)
⟧ ⟹ *check-exp* ((*E-list tan exps*)) []

| *check-list-consI*: ⟦
  Some (-,*t*) = *get tan*;
  Some elem-typ = *is-list-type tan*;
  check-exp-list [*exp1*] [*elem-typ*];
  check-exp-typ exp2 t
⟧ ⟹ *check-exp* ((*E-cons tan exp1 exp2*)) []

| *check-letI*: ⟦
    trace ''check-letI'';
    Some (*env*,*t1*) = *get tan*;
    check-letbind lb bindings;
    check-exp-typ-env (*add-locals env bindings*) *exp t1*
⟧ ⟹ *check-exp* ((*E-let tan lb exp*)) []

| *check-ifI*: ⟦
    Some (*env*,*t*) = *get tan*;
    check-exp exp1 bs1;
    Some t-exp1 = *type-of-exp exp1* ;
    Some nc = *deconstruct-bool-type t-exp1*;
    (*add-constraint nc env*) ⊢ *exp2* : *t*;
    (*add-constraint* (*nc-not nc*) *env*) ⊢ *exp3* : *t*
⟧ ⟹ *check-exp* ((*E-if tan exp1 exp2 exp3*)) []

| *check-varI*: ⟦
    Some (*env*,*t1*) = *get tan*;
    check-exp exp1 bs1;
    check-exp exp2 bs2;
    Some t2 = *type-of-exp exp1*;
    subtype env t1 t2
⟧ ⟹ *check-exp* ((*E-var tan* ((*LEXP-cast typ mvar*) -) *exp1 exp2*)) []

| *check-assignI*:⟦
    check-exp exp bs;
    check-lexp lexp bindings
⟧ ⟹ *check-exp* ((*E-assign tan lexp exp*)) *bindings*

| *check-caseI*:⟦
   trace ''check-caseI'';
   check-exp exp bs;
   check-pexps pexps
⟧ ⟹ *check-exp* ((*E-case tan exp pexps*)) []

| *check-loop1I*: ⟦
  Some (*env*,-) = *get tan*;
  check-exp exp1 bs1;
  check-exp exp2 bs2;
  Some t1 = *type-of-exp exp1* ;
  Some nc = *deconstruct-bool-type t1*;
  add-constraint nc env ⊢ *exp2* : *unit-typ*
⟧ ⟹ *check-exp* ((*E-loop tan lp lm exp1 exp2*)) []

| *check-forI*: ⟦
   ⊢ *exp1* : *int-typ*;
   ⊢ *exp2* : *int-typ*;
   ⊢ *exp3* : *int-typ*;
   ⊢ *exp4* : *unit-typ*
⟧ ⟹ *check-exp* ((*E-for tan x exp1 exp2 exp3 ord exp4*)) []

```
| check-pexps-nilI: check-pexps []
| check-pexps-conI: ⟦ check-pexp pexp; check-pexps pexps ⟧ ⟹ check-pexps (pexp#pexps)

| check-block-singleI: ⟦
   Some (-,t) = get tan;
   ⊢ exp : t ;
   trace (''block single t='' @ show t)
⟧ ⟹ check-exp ( (E-block tan [exp]) ) []


| check-block-consI:⟦
     check-exp exp1 bindings ;
     subtype-exp exp1 unit-typ;
     check-local-binds (exp2#exps) bindings;
     check-exp ( (E-block tan (exp2#exps)) ) bs
⟧ ⟹ check-exp ( (E-block tan (exp1#exp2#exps)) ) []


| check-assertI: ⟦
   check-exp assert-exp bs;
   Some t = type-of-exp assert-exp
⟧ ⟹ check-exp ( (E-assert tan assert-exp msg-exp) ) []


| check-letbindI:⟦
     check-pat pat bindings;
     check-exp exp bs
⟧ ⟹
     check-letbind ( (LB-val tan pat exp) ) bindings


| check-pexpI:⟦
     trace ''check-pexpI'';
     Some env = env-of exp;
     check-pat pat bindings;
     locals-in env bindings;
     check-exp exp bs
⟧ ⟹ check-pexp ( (Pat-exp tan pat exp) )


| check-pexp-whenI:⟦
     Some env = env-of exp;
     check-pat pat bindings;
     locals-in env bindings;
     check-exp exp bs;
     locals-in envg bindings;
     Some envg = env-of expg;
     check-exp expg bsg

⟧ ⟹ check-pexp ( (Pat-when tan pat expg exp) )
```

code-pred (*modes*:
check-exp: $i \Rightarrow o \Rightarrow bool$ **and**
check-letbind: $i \Rightarrow o \Rightarrow bool$ **and**
check-pexp : $i \Rightarrow bool$ **and**
check-pexps : $i \Rightarrow bool$ **and**
check-fexp : $i \Rightarrow i \Rightarrow bool$ **and**
check-fexp-list : $i \Rightarrow i \Rightarrow bool$ **and**
check-exp-list : $i \Rightarrow i \Rightarrow bool$ **and**
check-lexp : $i \Rightarrow o \Rightarrow bool$ **and**
check-lexp-list : $i \Rightarrow o \Rightarrow bool$ **and**
check-exp-typ : $i \Rightarrow i \Rightarrow bool$ **and**
check-exp-typ-env : $i \Rightarrow i \Rightarrow i \Rightarrow bool$
) [*show-steps*, *show-mode-inference*, *show-invalid-clauses*] *check-exp* **.**

value *True*

values {*x* . *subtype emptyEnv unit-typ unit-typ* }

values { *x*. *check-exp*
((*E-lit* (*set-type None unit-typ*) *L-unit*)) *x* }

value *add-local* (*set-type None* (*bool-typ True*))
((*id* (*STR* ''*x*''))) (*bool-typ True*)

inductive *check-funcls* :: *tannot funcl list* $\Rightarrow$ *tannot-opt* $\Rightarrow$ *bool* **where**
check-funcls-emptyI: *trace* ''*check-funcls-emptyI*'' $\Longrightarrow$ *check-funcls* [] -

| *check-funcls-consI*: ⟦
trace ''*check-funcls-consI*'';
check-funcls fs to;
trace ''*check-funcl*'';
check-pexp pexp
⟧ $\Longrightarrow$ *check-funcls* (((*FCL-Funcl* - *fid* ((*PEXP-funcl pexp*))))#*fs*) *to*

18.4.5 Definitions

inductive *check-sd* :: *env* $\Rightarrow$ *tannot scattered-def* $\Rightarrow$ *bool* **where**
check-sd env ((*SD-function ro to eo fid*) *tan*)
| ⟦
check-pexp pexp
⟧ $\Longrightarrow$ *check-sd env* ((*SD-funcl tan* ((*FCL-Funcl ftan fid* (*PEXP-funcl pexp*)))))

| *check-sd env* ((*SD-variant tan tyid tq*))
| *check-sd env* ((*SD-unioncl tan tyid tu*))

| *check-sd env* ((*SD-mapping tan mapid to*))
| *check-sd env* ((*SD-mapcl tan mapid mapcl*))

| *check-sd env* ((*SD-end tan tid*))

code-pred (*modes*: $i \Rightarrow i \Rightarrow bool$) [*show-steps*, *show-mode-inference*, *show-invalid-clauses*] *check-sd*
.

inductive *check-def* :: $env \Rightarrow tannot\ def \Rightarrow bool$ (- $\vdash$ -) **where**

check-typedefI:
check-def env (*DEF-type tdef*)

|*check-fundefI*:
⟦ *trace* ''*check-fundefI*''; *check-funcls funcls tannot-opt* ⟧ $\Longrightarrow$ *check-def e* (*DEF-fundef* ((*FD-function* - *rec-opt tannot-opt effect-top funcls*)))

|*check-mapdefI*: *check-def env* (*DEF-mapdef md*)

| *check-letbindI*:
⟦
check-letbind letbind bindings
⟧ $\Longrightarrow$ *check-def e* (*DEF-val letbind*)

| *check-valspecI*: *check-def env* (*DEF-spec* -)

| *check-fixityI*: *check-def env* (*DEF-fixity* - - -)
| *check-overloadI*: *check-def env* (*DEF-overload* - -)
| *check-set-orderI*: *check-def env* (*DEF-default* (-))
| *check-sdI*: *check-sd env sd* $\Longrightarrow$ *check-def env* (*DEF-scattered sd*)
| *check-measureI*: *check-def env* (*DEF-measure* - - -)
| *check-loop-measureI*: *check-def env* (*DEF-loop-measures* - -)
| *check-reg-decI*: *check-def env* (*DEF-reg-dec* -)
| *check-internal-mutrecI*: *check-def env* (*DEF-internal-mutrec* -)
| *check-pragmaI*: *check-def env* (*DEF-pragma* - -)

code-pred (*modes*: $i \Rightarrow i \Rightarrow bool$) [*show-steps*, *show-mode-inference*, *show-invalid-clauses*] *check-def*
.

definition *env2* $\equiv$ *add-local* (*set-type None unit-typ*) ((*id* (*STR* ''*z*''))) *unit-typ*

code-printing
  constant *Debug.trace* ⇀ (*OCaml*) *Utils2.trace* -

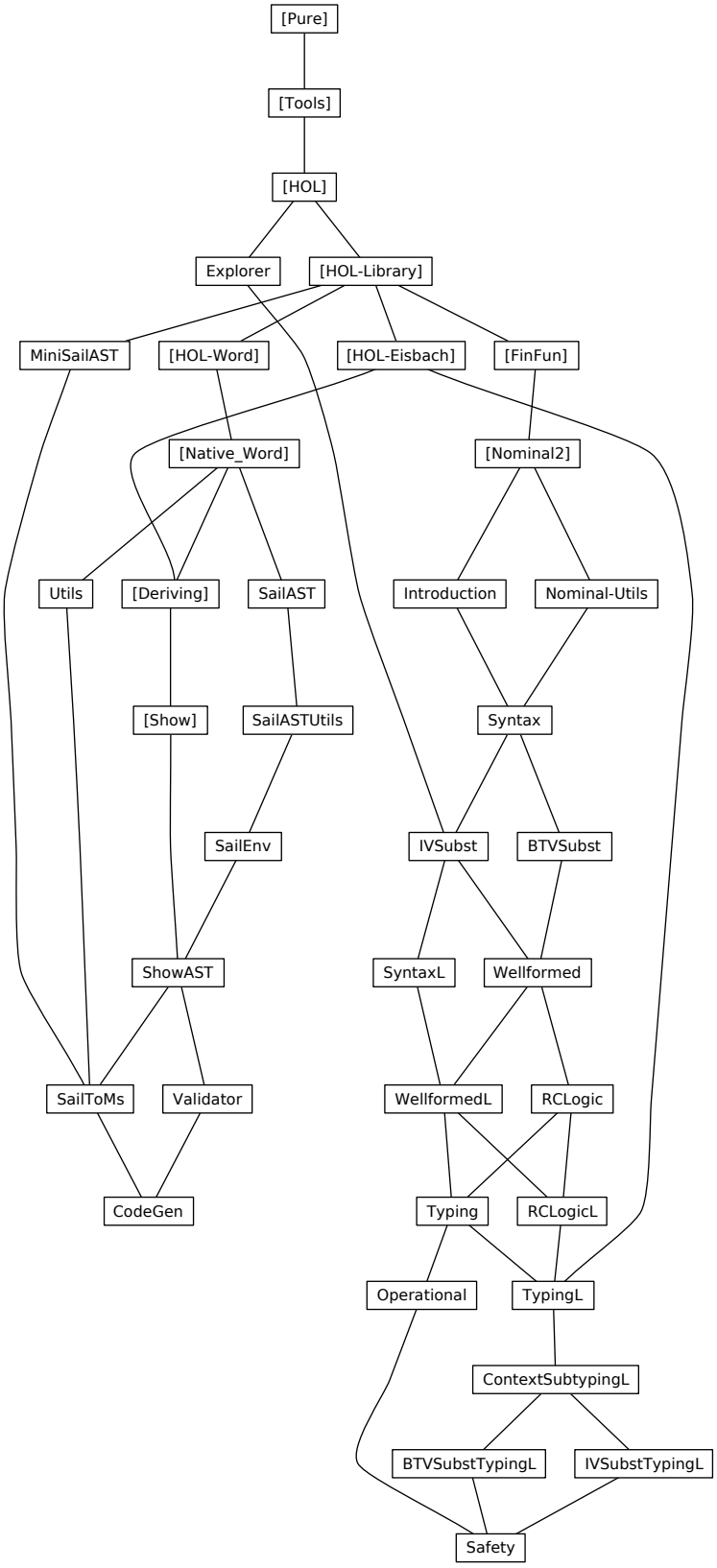
18.5 Examples

end
theory *CodeGen*
imports *SailToMs Validator*
begin

export-code open *check-def def-conv-i-i-i-o s-conv-i-i-i-o-o-o-o-o-o-o set-of-pred emptyEnv SailEnv.constraints*

in *OCaml* **file** *minisail-isa.ml*

end


[Pure]
[Tools]
[HOL]
Explorer
[HOL-Library]
MiniSailAST
[HOL-Word]
[HOL-Eisbach]
[FinFun]
[Native_Word]
[Nominal2]
Utils
[Deriving]
SailAST
Introduction
Nominal-Utils
[Show]
SailASTUtils
Syntax
SailEnv
IVSubst
BTVSubst
ShowAST
SyntaxL
Wellformed
SailToMs
Validator
WellformedL
RCLogic
CodeGen
Typing
RCLogicL
Operational
TypingL
ContextSubtypingL
BTVSubstTypingL
IVSubstTypingL
Safety