

MiniSail

Mark P. Wassell, University of Cambridge

January 2, 2021

Contents

1	Introduction	5
2	Prelude	6
2.1	Lemmas helping with equivariance proofs	6
2.2	Freshness via equivariance	7
2.3	Additional simplification rules	8
2.4	Additional equivariance lemmas	8
2.5	Freshness lemmas	10
2.6	Freshness and support for subsets of variables	11
2.7	The set of free variables of an expression	11
2.8	Other useful lemmas	13
3	Syntax	19
3.1	Program Syntax	19
3.1.1	Datatypes	19
3.1.2	Lemmas	22
3.2	Context Syntax	32
3.2.1	Datatypes	32
3.2.2	Functions and Lemmas	32
3.2.3	Immutable Variable Context Lemmas	35
3.2.4	Mutable Variable Context Lemmas	39
3.2.5	Lookup Functions	39
4	Immutable Variable Substitution	42
4.1	Class	42
4.2	Values	43
4.3	Expressions	44
4.4	Expressions in Constraints	47
4.5	Constraints	49
4.6	Variable Context	52
4.7	Types	53
4.8	Mutable Variable Context	60
4.9	Statements	62
4.10	Type Definition	67
4.11	Variable Context	71
4.12	Lookup	71

5	Base Type Variable Substitution	73
5.1	Class	73
5.2	Base Type	74
5.3	Value	77
5.4	Constraints Expressions	79
5.5	Constraints	81
5.6	Types	82
5.7	Expressions	85
5.8	Statements	87
5.9	Function Type	97
5.10	Contexts	99
	5.10.1 Immutable Variables	99
	5.10.2 Mutable Variables	102
6	Wellformed Terms	107
6.1	Definitions	107
7	Refinement Constraint Logic	118
7.1	Evaluation and Satisfiability	118
	7.1.1 Valuation	118
	7.1.2 Evaluation base-types	120
	7.1.3 Wellformed Evaluation	120
	7.1.4 Evaluating Terms	120
	7.1.5 Satisfiability	121
7.2	Validity	122
7.3	Lemmas	122
7.4	Syntax Lemmas	123
7.5	Type Definitions	128
7.6	Function Definitions	129
8	Wellformedness Lemmas	132
8.1	Prelude	132
8.2	Strong Elimination	132
8.3	Context Extension	133
8.4	Context	133
8.5	Converting between wb forms	136
8.6	Support	139
8.7	Freshness	150
8.8	Misc	154
8.9	Context Strengthening	155
8.10	Type Definitions	160
	8.10.1 Simple	164
	8.10.2 Polymorphic	166
8.11	Equivariance Lemmas	171
8.12	Lookup	172
8.13	Function Definitions	180
8.14	Weakening	190

8.15	Forms	201
8.16	Replacing	204
8.17	Substitution	211
9	Type System	231
9.1	Subtyping	231
9.2	Literals	231
9.3	Values	232
9.4	Introductions	234
9.5	Expressions	234
9.6	Statements	237
9.7	Programs	240
10	Operational Semantics	243
10.1	Reduction Rules	243
10.2	Reduction Typing	246
11	Refinement Constraint Logic Lemmas	249
11.1	Lemmas	249
11.2	Existence of evaluation	250
11.3	Satisfiability	258
11.4	Substitution for Evaluation	259
11.5	Validity	265
11.5.1	Weakening and Strengthening	266
11.5.2	Updating valuation	270
11.6	Base Type Substitution	274
11.7	Expression Operator Lemmas	290
12	Typing Lemmas	302
12.1	Subtyping	302
12.2	Literals	317
12.3	Values	319
12.4	Expressions	328
12.5	Statements	334
12.6	Replacing Variables	334
12.7	Additional Elimination and Intros	346
12.7.1	Values	346
12.7.2	Expressions	347
12.8	Weakening	347
12.8.1	Weakening Immutable Variable Context	356
13	Context Subtyping Lemmas	361
13.1	Replace Type of Variable in Context	361
13.2	Validity and Subtyping	367
13.3	Literals	370
13.4	Values	370
13.5	Expressions	373

13.6 Statements	377
14 Immutable Variable Substitution Lemmas	384
14.1 Proof Methods	384
14.2 Misc	384
14.3 Context	385
14.4 Satisfiability	386
14.5 Validity	387
14.6 Subtyping	389
14.7 Values	393
14.8 Expressions	398
14.9 Statements	407
15 Base Type Variable Substitution Lemmas	419
16 Safety	434
16.1 Operational Semantics	434
16.2 Preservation	436
16.2.1 Function Application	436
16.2.2 Operators	443
16.2.3 Let Statements	444
16.2.4 Other Statements	448
16.2.5 Main Lemma	457
16.3 Progress	466
16.4 Safety	471

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-Utills.

2.1 Lemmas helping with equivariance proofs

lemma *perm-rel-lemma*:

assumes $\bigwedge \pi \ x \ y. \ r \ (\pi \cdot x) \ (\pi \cdot y) \implies r \ x \ y$
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 $\bigwedge \pi \ x \ y. \ r \ x \ y \implies r \ (\pi \cdot x) \ (\pi \cdot y)$
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-equivI*:

assumes *f-equiv*[*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-equiv*[*eqvt*]: $p \cdot \text{size-list } f \ x = \text{size-list } (p \cdot f) \ (p \cdot x)$

proof (*induction* x)

```

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

```

2.2 Freshness via equivariance

```

lemma eqvt-fresh-cong1: ( $\bigwedge p x. p \cdot (f x) = f (p \cdot x)$ )  $\implies a \# x \implies a \# f x$ 
  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:  $a \# x$  and fresh2:  $a \# y$ 
  shows  $a \# f x y$ 
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  $a \# (x, y)$  using fresh1 fresh2 by auto
ultimately
  have  $a \# (\lambda (x,y). f x y) (x, y)$  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:  $a \#* x$ 
  shows  $a \#* f x$ 
  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:  $a \#* x$  and fresh2:  $a \#* y$ 
  shows  $a \#* f x y$ 
  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:  $a \# x$  and fresh2:  $a \# y$  and fresh3:  $a \# z$ 

```



```

shows a # f x y z
proof-
  have eqvt (λ (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 # (x, y, z) using fresh1 fresh2 fresh3 by auto
ultimately
  have a # (λ (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: (λ p x y z. p · (f x y z) = f (p · x) (p · y) (p · z))
  and fresh1: a #* x and fresh2: a #* y and fresh3: a #* z
  shows a #* 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 # x ⟷ False
  by (metis fresh-at-base(2))

lemma fresh-star-singleton: { x } #* e ⟷ x # e
  by (simp add: fresh-star-def)

```

2.4 Additional equivariance lemmas

```

lemma eqvt-cases:
  fixes f x π
  assumes eqvt: λ x. π · f x = f (π · x)
  obtains f x f (π · x) | ¬ f x ¬ f (π · x)
  using assms[symmetric]
  by (cases f x) auto

lemma range-eqvt: π · range Y = range (π · Y)
  unfolding image-eqvt UNIV-eqvt ..

lemma case-option-eqvt[eqvt]:
  π · case-option d f x = case-option (π · d) (π · f) (π · x)
  by (cases x)(simp-all)

lemma supp-option-eqvt:
  supp (case-option d f x) ⊆ supp d ∪ supp f ∪ supp x
  apply (cases x)
  apply (auto simp add: supp-Some)
  apply (metis (mono-tags) Un-iff subsetCE supp-fun-app)

```

done

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

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

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

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

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

```

apply perm-simp
apply auto
done

```

```

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

```

```

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

```

```

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

```

```

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

```

```

lemma Sigma-eqvt'[eqvt]:  $\pi \cdot \text{Sigma} = \text{Sigma}$ 
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]:
 $\pi \cdot (\text{override-on } m1 \ m2 \ S) = \text{override-on } (\pi \cdot m1) \ (\pi \cdot m2) \ (\pi \cdot S)$ 
by (auto simp add: override-on-def )

```

```

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

```

```

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

```

```

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

```

2.5 Freshness lemmas

```

lemma fresh-list-elem:
assumes a  $\# \Gamma$ 
and e  $\in \text{set } \Gamma$ 

```

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

lemma *set-not-fresh*:
 $x \in \text{set } L \implies \neg(\text{atom } x \# L)$
 by (metis fresh-list-elem not-self-fresh)

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

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

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

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

2.6 Freshness and support for subsets of variables

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

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

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

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

2.7 The set of free variables of an expression

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

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

lemma *fv-Nil[simp]*: $\text{fv } [] = \{\}$
 by (auto simp add: *fv-def* *supp-Nil*)

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

```

lemma fv-Pair[simp]:  $fv\ x\ y = fv\ x \cup fv\ y$ 
  by (auto simp add: fv-def supp-Pair)
lemma fv-append[simp]:  $fv\ (x\ @\ y) = fv\ x \cup fv\ y$ 
  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:  $a \notin fv\ x \implies b \notin fv\ x \implies (a \leftrightarrow b) \cdot x = x$ 
  by (metis flip-def fresh-def fv-def mem-Collect-eq swap-fresh-fresh)

lemma fv-not-fresh:  $atom\ x \# e \longleftrightarrow x \notin fv\ e$ 
  unfolding fv-def fresh-def by blast

lemma fresh-fv:  $finite\ (fv\ e :: 'a\ set) \implies atom\ (x :: ('a::at-base)) \# (fv\ e :: 'a\ set) \longleftrightarrow atom\ x \# e$ 
  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 ::  $'a::fs \Rightarrow 'b::at-base\ list$ 
  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]:
   $a \# (fv-list\ e :: 'b::at-base\ list) \longleftrightarrow a \# (fv\ e :: 'b::at-base\ set)$ 
proof–
  have  $a \# (fv-list\ e :: 'b::at-base\ list) \longleftrightarrow a \# set\ (fv-list\ e :: 'b::at-base\ list)$ 

```

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

2.8 Other useful lemmas

lemma pure-permute-id: $permute\ p = (\lambda\ x.\ (x::'a::pure))$
 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: $eqvt\ f \implies eqvt\ (case-prod\ f)$
 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 $\text{supp } (\text{case-prod } f \ (x,y) \ z) \subseteq \text{supp } (x,y) \cup \text{supp } z.$
thus *?thesis* **by** (*simp add: supp-Pair*)
qed

lemma *permute-0[simp]*: $\text{permute } 0 = (\lambda x. x)$
by *auto*

lemma *permute-comp[simp]*: $\text{permute } x \circ \text{permute } y = \text{permute } (x + y)$ **by** *auto*

lemma *map-permute*: $\text{map } (\text{permute } p) = \text{permute } p$
apply *rule*
apply (*induct-tac x*)
apply *auto*
done

lemma *fresh-star-restrictA[intro]*: $a \# \Gamma \implies a \# AList.restrict \ V \ \Gamma$
by (*induction \Gamma*) (*auto simp add: fresh-star-Cons*)

lemma *Abs-lst-Nil-eq[simp]*: $[\] \text{lst}. (x :: 'a :: fs) = [xs] \text{lst}. x' \longleftrightarrow (([\], x) = (xs, x'))$
apply *rule*
apply (*frule Abs-lst-fcb2* **where** $f = \lambda x \ y. (x, y)$ **and** $as = [\]$ **and** $bs = xs$ **and** $c = ()$)
apply (*auto simp add: fresh-star-def*)
done

lemma *Abs-lst-Nil-eq2[simp]*: $[xs] \text{lst}. (x :: 'a :: fs) = [\] \text{lst}. x' \longleftrightarrow ((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*]:
 $(\bigwedge a \ b \ c \ d \ e \ f \ g \ h. P \ (a, b, c, d, e, f, g, h)) \implies P \ x$
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*]:
 $(\bigwedge a \ b \ c \ d \ e \ f \ g \ h \ i. P \ (a, b, c, d, e, f, g, h, i)) \implies P \ x$
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]*: $x \# (a, b) = (x \# a \wedge x \# b)$

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

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

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

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

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

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

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

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

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

lemma *fresh-prod12[nominal-prod-simps,ms-fresh]*: $x \# (a,b,c,d,e,f,g,h,i,j,k,l) = (x \# a \wedge x \# b \wedge x \# c \wedge x \# d \wedge x \# e \wedge x \# f \wedge x \# g \wedge x \# h \wedge x \# i \wedge x \# j \wedge x \# k \wedge x \# l)$
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 $x \# x1$ **and** $x \# x2$
shows $x \# (x1,x2)$ **using** *fresh-prod2 assms* **by** *auto*

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

lemma *fresh-prod4I*:

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

lemma *fresh-prod5I*:

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

lemma *flip-collapse[simp]*:

fixes $b1::'a::pt$ and $bv1::'b::at$ and $bv2::'b::at$
 assumes $atom\ bv2 \# b1$ and $atom\ c \# (bv1, bv2, b1)$ 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-eqv[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-eqv* 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 $(\forall c. atom\ c \# (t2, t2') \longrightarrow atom\ c \# (x, x', t1, t1') \longrightarrow (x \leftrightarrow c) \cdot t1 = (x' \leftrightarrow c) \cdot t1')$

proof(*rule, rule, rule*)

fix $c::'a$

assume $atom\ c \# (t2, t2')$ and $atom\ c \# (x, x', t1, t1')$

hence $atom\ c \# (x, x', (t1, t2), (t1', t2'))$ using *fresh-prod2* by *simp*

thus $(x \leftrightarrow c) \cdot t1 = (x' \leftrightarrow c) \cdot t1'$ using *assms Abs1-eq-iff-all(3)* *Pair-eqv* 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 $(\forall c. atom\ c \# (t1, t1') \longrightarrow atom\ c \# (x, x', t2, t2') \longrightarrow (x \leftrightarrow c) \cdot t2 = (x' \leftrightarrow c) \cdot t2')$

```

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)

```

sledgehammer-params[debug=true,timeout=600]

lemma lst-supp:

```

assumes [[atom x1]]lst. t1 = [[atom x2]]lst. t2
shows supp t1 - {atom x1} = supp t2 - {atom x2}
proof -
have supp ([[atom x1]]lst.t1) = supp ([[atom x2]]lst.t2) using assms by auto
thus ?thesis using Abs-finite-supp
  by (metis assms empty-set list.simps(15) supp-lst.simps)

```

qed

lemma *lst-supp-subset*:

assumes $[[atom\ x1]]lst.\ t1 = [[atom\ x2]]lst.\ t2$ **and** $supp\ t1 \subseteq \{atom\ x1\} \cup B$

shows $supp\ t2 \subseteq \{atom\ x2\} \cup B$

using *assms lst-supp* **by** *fast*

lemma *projl-inl-eqvt*:

fixes $\pi :: perm$

shows $\pi \cdot (projl\ (Inl\ x)) = projl\ (Inl\ (\pi \cdot x))$

unfolding *projl-def Inl-eqvt* **by** *simp*

end

sledgehammer-params $[debug=true, timeout=600, provers= cvc4\ spass\ e\ vampire\ z3, isar-proofs=true, smt-proofs=false]$

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. We include a type identifier, `tyid`, in the constructors to make typing and well-formedness checking easier

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

Binary Operations

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

Expressions

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

Expressions for Constraints

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

Constraints

nominal-datatype $c =$
 $C\text{-true} \quad (TRUE \ [] \ 50)$
 $| C\text{-false} \quad (FALSE \ [] \ 50)$
 $| C\text{-conj } c \ c \quad (- \ AND \ - \ [50, 50] \ 50)$
 $| C\text{-disj } c \ c \quad (- \ OR \ - \ [50, 50] \ 50)$
 $| C\text{-not } c \quad (\neg \ - \ [] \ 50)$
 $| C\text{-imp } c \ c \quad (- \ IMP \ - \ [50, 50] \ 50)$
 $| C\text{-eq } ce \ ce \quad (- \ == \ - \ [50, 50] \ 50)$

Refined type

nominal-datatype $\tau =$
 $T\text{-refined-type } x::x \ b \ c::c \ \text{binds } x \ \text{in } c \quad (\{ - : - \ | \ - \} \ [50, 50] \ 1000)$

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::s binds x in s ( ( - - ⇒ - ) )
and branch-list =
  AS-final branch-s ( { - } )
| AS-cons branch-s branch-list ( ( - | - ) )

```

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 (PROG - - -)

```

```

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* \notin *atom*'*us*
 by (simp add: image-iff)

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

lemma *x-not-in-b-set*[simp]:
 fixes *x::x* and *bs::bv* fset
 shows *atom* *x* \notin *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* \sharp *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* \sharp *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* \notin *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* \notin *atom*'*xs*
 by (simp add: image-iff)

lemma *u-not-in-b-atoms*[simp]:
 fixes *b :: b* and *u::u*
 shows *atom* *u* \notin *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 $\text{atom } u \notin \text{supp } bs$
by(*induct* bs , *auto simp add: supp-at-base supp-finsert*)

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

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

lemma *supp-b-u-disjoint[simp]*:
fixes $b::b$ **and** $u::u$
shows $\text{supp } u \cap \text{supp } b = \{\}$
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 $\text{atom } u \# b$
using *fresh-at-base* **by** *simp*

Base Types

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

lemma *supp-b-empty[simp]*:
fixes $b :: b$ **and** $x::x$
shows $\text{atom } x \notin \text{supp } b$
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* $b.\text{eq-iff}(\gamma)$ $b.\text{perm-simps}(\gamma)$ *flip-b-id*)

lemma *flip-bv-x-cancel[simp]*:
fixes $bv::bv$ **and** $z::bv$ **and** $x::x$
shows $(bv \leftrightarrow z) \cdot x = x$ **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 $(bv \leftrightarrow z) \cdot x = x$ **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 $supp\ (V\text{-}lit\ l) = \{\}$
by(*nominal-induct* l *rule*: *l.strong-induct*,
auto *simp* *add*: *l.strong-exhaust* *pure-supp* *v.fv-defs* *supp-bitvec-empty*)

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

lemma *flip-bitvec0*:
 fixes $x::bit\ list$
 assumes $atom\ c \# (z, x, z')$
 shows $(z \leftrightarrow c) \cdot x = (z' \leftrightarrow c) \cdot x$
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 $atom\ c \# (z, L\text{-}bitvec\ x, z')$
 shows $(z \leftrightarrow c) \cdot x = (z' \leftrightarrow c) \cdot x$
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 *type-l-eq*:

shows $\{ \{ z : b \mid [[z]^v]^{ce} == [V\text{-lit } l]^{ce} \} = (\{ z' : b \mid [[z']^v]^{ce} == [V\text{-lit } l]^{ce} \})$
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 # x and atom c # x and atom z' # 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 # x and atom c # x' and atom c # x and atom z' # 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 $\exists z. \text{atom } z \# (c', t) \wedge (\{ z' : b \mid c' \}) = (\{ z : b \mid (z \leftrightarrow z') \cdot c' \})$

proof –

obtain $z::x$ **where** *fr: atom z # (c', t)* **using** *obtain-fresh* **by** *blast*
moreover **hence** $(\{ z' : b \mid c' \}) = (\{ z : b \mid (z \leftrightarrow z') \cdot c' \})$
using *$\tau.\text{eq-iff Abs1-eq-iff}$*
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 $\exists z b c. \text{atom } z \# t \wedge \tau = \{ z : b \mid c \}$

proof –

obtain z' **and** b **and** c' **where** *teq: $\tau = (\{ z' : b \mid c' \})$* **using** *$\tau.\text{exhaust}$* **by** *blast*
obtain $z::x$ **where** *fr: atom z # (t, c')* **using** *obtain-fresh* **by** *blast*
hence $(\{ z' : b \mid c' \}) = (\{ z : b \mid (z \leftrightarrow z') \cdot c' \})$ **using** *$\tau.\text{eq-iff Abs1-eq-iff}$*
flip-commute flip-fresh-fresh fresh-PairD(1) **by** (*metis fresh-PairD(2)*)
hence $\text{atom } z \# t \wedge \tau = (\{ z : b \mid (z \leftrightarrow z') \cdot c' \})$ **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 $\#$ t \wedge $\tau = \{ \{ z : b \mid c \} \}$
using has-fresh-z by blast

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

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

Values

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 $\#$ (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  $(x \leftrightarrow c) \cdot v = (xa \leftrightarrow c) \cdot va$ 
shows  $((v = V\text{-lit } l \longrightarrow (\exists l'. va = V\text{-lit } l' \wedge (x \leftrightarrow c) \cdot l = (xa \leftrightarrow c) \cdot l')) \wedge$ 
 $((v = V\text{-var } y \longrightarrow (\exists y'. va = V\text{-var } y' \wedge (x \leftrightarrow c) \cdot y = (xa \leftrightarrow c) \cdot y')) \wedge$ 
 $((v = V\text{-pair } vone\ vtwo \longrightarrow (\exists v1'\ v2'. va = V\text{-pair } v1'\ v2' \wedge (x \leftrightarrow c) \cdot vone = (xa \leftrightarrow c) \cdot v1'$ 
 $\wedge (x \leftrightarrow c) \cdot vtwo = (xa \leftrightarrow c) \cdot v2')) \wedge$ 
 $((v = V\text{-cons } tyid\ dc\ vone \longrightarrow (\exists v1'. va = V\text{-cons } tyid\ dc\ v1' \wedge (x \leftrightarrow c) \cdot vone = (xa \leftrightarrow c) \cdot$ 
 $v1')) \wedge$ 
 $((v = V\text{-consp } tyid\ dc\ b\ vone \longrightarrow (\exists v1'. va = V\text{-consp } tyid\ dc\ b\ v1' \wedge (x \leftrightarrow c) \cdot vone = (xa$ 
 $\leftrightarrow c) \cdot v1'))$ 
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  $(V\text{-pair } v1\ v2 = V\text{-pair } vone\ vtwo \longrightarrow (\exists v1'\ v2'. va = V\text{-pair } v1'\ v2' \wedge (x \leftrightarrow c) \cdot vone = (xa$ 
 $\leftrightarrow c) \cdot v1' \wedge (x \leftrightarrow c) \cdot vtwo = (xa \leftrightarrow c) \cdot v2'))$  proof
    assume  $V\text{-pair } v1\ v2 = V\text{-pair } vone\ vtwo$ 
    thus  $(\exists v1'\ v2'. va = V\text{-pair } v1'\ v2' \wedge (x \leftrightarrow c) \cdot vone = (xa \leftrightarrow c) \cdot v1' \wedge (x \leftrightarrow c) \cdot vtwo = (xa$ 
 $\leftrightarrow c) \cdot v2')$ 
    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  $(V\text{-cons } tyid\ dc\ v1 = V\text{-cons } tyid\ dc\ vone \longrightarrow (\exists v1'. va = V\text{-cons } tyid\ dc\ v1' \wedge (x \leftrightarrow c) \cdot$ 
 $vone = (xa \leftrightarrow c) \cdot v1'))$  proof
    assume as:  $V\text{-cons } tyid\ dc\ v1 = V\text{-cons } tyid\ dc\ vone$ 
    hence  $(x \leftrightarrow c) \cdot (V\text{-cons } tyid\ dc\ vone) = V\text{-cons } tyid\ dc\ ((x \leftrightarrow c) \cdot vone)$  proof –
      have  $(x \leftrightarrow c) \cdot dc = dc$  using pure-permute-id by metis
      moreover have  $(x \leftrightarrow c) \cdot tyid = tyid$  using pure-permute-id by metis
      ultimately show ?thesis using v.perm-simps(4) by simp
    qed
    then obtain v1' where  $(xa \leftrightarrow c) \cdot va = V\text{-cons } tyid\ dc\ v1' \wedge (x \leftrightarrow c) \cdot vone = v1'$  using assms
V-cons
    using as by fastforce
    hence  $va = V\text{-cons } tyid\ dc\ ((xa \leftrightarrow c) \cdot v1') \wedge (x \leftrightarrow c) \cdot vone = v1'$  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-consp 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  $\langle d \neq x \rangle$  by fastforce+
next
case (V-cons tyid dc v)
show ?case using v.supp(4) pure-supp
using V-cons.hyps V-cons.premis 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.premis 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  $\text{supp } ((z \leftrightarrow d) \cdot e) \subseteq \text{supp } e - \{ \text{atom } z \} \cup \{ \text{atom } d \}$ 
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  $\text{supp } ((z \leftrightarrow d) \cdot (\text{AE-appP } b f v)) = \text{supp } (\text{AE-appP } b f ((z \leftrightarrow d) \cdot v))$  using e.supp
by (metis b.eq-iff(3) b.perm-simps(3) e.perm-simps(3) flip-b-id)
also have ... =  $\text{supp } b \cup \text{supp } f \cup \text{supp } ((z \leftrightarrow d) \cdot v)$  using e.supp by auto
also have ...  $\subseteq \text{supp } b \cup \text{supp } f \cup \text{supp } v - \{ \text{atom } z \} \cup \{ \text{atom } d \}$  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  $d \# v1 \wedge \text{atom } d \# v2$  using fresh-def e.supp by force
have  $((z \leftrightarrow d) \cdot (\text{AE-op } opp v1 v2)) = \text{AE-op } opp ((z \leftrightarrow d) \cdot v1) ((z \leftrightarrow d) \cdot v2)$  using
e.perm-simps flip-commute opp.perm-simps AE-op opp.strong-exhaust pure-supp
by (metis (full-types))

hence  $\text{supp } ((z \leftrightarrow d) \cdot \text{AE-op } opp v1 v2) = \text{supp } (\text{AE-op } opp ((z \leftrightarrow d) \cdot v1) ((z \leftrightarrow d) \cdot v2))$  by simp
also have ... =  $\text{supp } ((z \leftrightarrow d) \cdot v1) \cup \text{supp } ((z \leftrightarrow d) \cdot v2)$  using e.supp
by (metis (mono-tags, hide-lams) opp.strong-exhaust opp.supp sup-bot.left-neutral)
also have ...  $\subseteq (\text{supp } v1 - \{ \text{atom } z \} \cup \{ \text{atom } d \}) \cup (\text{supp } v2 - \{ \text{atom } z \} \cup \{ \text{atom } d \})$  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-supply 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 supply-eqv
  swap-set-in-eq
  by (metis sort-of-atom-eq)
next
  case (AE-len v)
then show ?case using swap-v-supply by auto
next
  case (AE-concat v1 v2)
then show ?case using swap-v-supply by auto
next
  case (AE-split v1 v2)
then show ?case using swap-v-supply by auto
qed

```

```

lemma swap-ce-supply:
  fixes e::ce and d::x and z::x
  assumes atom d  $\#$  e
  shows supply ((z  $\leftrightarrow$  d)  $\cdot$  e)  $\subseteq$  supply e - { atom z }  $\cup$  { atom d }
  using assms
proof(nominal-induct e rule:ce.strong-induct)
  case (CE-val v)
  then show ?case using swap-v-supply ce.fresh ce.supply by simp
next
  case (CE-op opp v1 v2)
  hence df: atom d  $\#$  v1  $\wedge$  atom d  $\#$  v2 using fresh-def e.supply by force
  have ((z  $\leftrightarrow$  d)  $\cdot$  (CE-op opp v1 v2)) = CE-op opp ((z  $\leftrightarrow$  d)  $\cdot$  v1) ((z  $\leftrightarrow$  d)  $\cdot$  v2) using
  ce.perm-simps flip-commute opp.perm-simps CE-op opp.strong-exhaust x-fresh-b pure-supply
  by (metis (full-types))

  hence supply ((z  $\leftrightarrow$  d)  $\cdot$  CE-op opp v1 v2) = supply (CE-op opp ((z  $\leftrightarrow$  d)  $\cdot$  v1) ((z  $\leftrightarrow$  d)  $\cdot$  v2)) by simp
  also have ... = supply ((z  $\leftrightarrow$  d)  $\cdot$  v1)  $\cup$  supply ((z  $\leftrightarrow$  d)  $\cdot$  v2) using ce.supply
  by (metis (mono-tags, hide-lams) opp.strong-exhaust opp.supply sup-bot.left-neutral)
  also have ...  $\subseteq$  (supply v1 - { atom z }  $\cup$  { atom d })  $\cup$  (supply v2 - { atom z }  $\cup$  { atom d }) using
  swap-v-supply CE-op df by blast
  finally show ?case using ce.supply opp.supply by blast
next
  case (CE-fst v)
  then show ?case using ce.supply ce.fresh swap-v-supply by auto
next
  case (CE-snd v)
then show ?case using ce.supply ce.fresh swap-v-supply by auto
next
  case (CE-len v)
  then show ?case using ce.supply ce.fresh swap-v-supply by auto
next

```

case (*CE-concat* *v1* *v2*)
then show ?*case* **using** *ce.supp ce.fresh swap-v-supp ce.perm-simps*
proof –
 have $\forall x v xa. \neg \text{atom } (x::x) \# (v::v) \vee \text{supp } ((xa \leftrightarrow x) \cdot v) \subseteq \text{supp } v - \{\text{atom } xa\} \cup \{\text{atom } x\}$
 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* $\#$ *c*
 shows $\text{supp } ((z \leftrightarrow d) \cdot c) \subseteq \text{supp } c - \{\text{atom } z\} \cup \{\text{atom } d\}$
 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* $\#$ *e* and *atom z'* $\#$ *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* $\#$ *e* and *atom z'* $\#$ *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* $\#$ (*e, ea*) and *atom c* $\#$ (*x, xa, e, ea*) and $(x \leftrightarrow c) \cdot e = (xa \leftrightarrow c) \cdot ea$
 shows $(e = \text{AE-val } w \longrightarrow (\exists w'. ea = \text{AE-val } w' \wedge (x \leftrightarrow c) \cdot w = (xa \leftrightarrow c) \cdot w')) \vee$
 $(e = \text{AE-op } opp \ v1 \ v2 \longrightarrow (\exists v1' \ v2'. ea = \text{AS-op } opp \ v1' \ v2' \wedge (x \leftrightarrow c) \cdot v1 = (xa \leftrightarrow c) \cdot$
 $v1') \wedge (x \leftrightarrow c) \cdot v2 = (xa \leftrightarrow c) \cdot v2')) \vee$
 $(e = \text{AE-fst } v \longrightarrow (\exists v'. ea = \text{AE-fst } v' \wedge (x \leftrightarrow c) \cdot v = (xa \leftrightarrow c) \cdot v')) \vee$
 $(e = \text{AE-snd } v \longrightarrow (\exists v'. ea = \text{AE-snd } v' \wedge (x \leftrightarrow c) \cdot v = (xa \leftrightarrow c) \cdot v')) \vee$
 $(e = \text{AE-len } v \longrightarrow (\exists v'. ea = \text{AE-len } v' \wedge (x \leftrightarrow c) \cdot v = (xa \leftrightarrow c) \cdot v')) \vee$
 $(e = \text{AE-concat } v1 \ v2 \longrightarrow (\exists v1' \ v2'. ea = \text{AS-concat } v1' \ v2' \wedge (x \leftrightarrow c) \cdot v1 = (xa \leftrightarrow c) \cdot v1'$
 $\wedge (x \leftrightarrow c) \cdot v2 = (xa \leftrightarrow c) \cdot v2')) \vee$
 $(e = \text{AE-app } f \ v \longrightarrow (\exists v'. ea = \text{AE-app } f \ v' \wedge (x \leftrightarrow c) \cdot v = (xa \leftrightarrow c) \cdot v'))$
by (*metis assms e.perm-simps permute-flip-cancel2*)

lemma *fresh-opp-all*:
 fixes *opp::opp*
 shows *z* $\#$ *opp*
 using *e.fresh opp.exhaust opp.fresh* **by** *metis*

lemma *fresh-e-opp-all*:
 shows $(z \# v1 \wedge z \# v2) = z \# \text{AE-op } opp \ v1 \ v2$

using *e.fresh opp.exhaust opp.fresh fresh-opp-all* **by** *simp*

lemma *fresh-e-opp*:

fixes *z::x*

assumes *atom z # v1 ∧ atom z # v2*

shows *atom z # AE-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 # (v, va) and atom c # (x, xa, v, va) and (x ↔ c) · s = (xa ↔ c) · sa*

shows *(s = AS-val w → (∃ w'. sa = AS-val w' ∧ (x ↔ c) · w = (xa ↔ c) · w')) ∨*

(s = AS-seq s1 s2 → (∃ s1' s2'. sa = AS-seq s1' s2' ∧ (x ↔ c) · s1 = (xa ↔ c) · s1') ∧ (x ↔ c) · s2 = (xa ↔ c) · s2') ∨

(s = AS-if v s1 s2 → (∃ v' s1' s2'. sa = AS-if seq s1' s2' ∧ (x ↔ c) · s1 = (xa ↔ c) · s1') ∧ (x ↔ c) · s2 = (xa ↔ c) · s2' ∧ (x ↔ c) · c = (xa ↔ c) · v')

by (*metis assms s-branch-s-branch-list.perm-simps permute-flip-cancel2*)

3.2 Context Syntax

3.2.1 Datatypes

Type and function/type definition contexts

type-synonym $\Phi = \text{fun-def list}$

type-synonym $\Theta = \text{type-def list}$

type-synonym $\mathcal{B} = \text{bv fset}$

datatype $\Gamma =$

GNil

| *GCons x*b*c* Γ (**infixr** $\#_{\Gamma}$ 65)

datatype $\Delta =$

DNil ($\llbracket \Delta \rrbracket$)

| *DCons u*τ* Δ (**infixr** $\#_{\Delta}$ 65)

3.2.2 Functions and Lemmas

lemma *Γ-induct* [*case-names GNil GCons*] : $P \text{ GNil} \implies (\bigwedge x \ b \ c \ \Gamma'. P \ \Gamma' \implies P \ ((x,b,c) \#_{\Gamma} \Gamma')) \implies P \ \Gamma$

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  $\Delta :: pt$ 
begin

primrec permute- $\Delta$ 
where
  DNil-eqv:  $p \cdot DNil = DNil$ 
| DCons-eqv:  $p \cdot (x \#_{\Delta} xs) = p \cdot x \#_{\Delta} p \cdot (xs::\Delta)$ 

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

lemmas [eqv] = permute- $\Delta$ .simps

lemma  $\Delta$ -induct [case-names DNil DCons] :  $P \ DNil \implies (\bigwedge u \ t \ \Delta'. \ P \ \Delta' \implies P \ ((u,t) \#_{\Delta} \Delta')) \implies P \ \Delta$ 
proof(induct  $\Delta$  rule:  $\Delta$ .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  $\Phi$ -induct [case-names PNil PConsNone PConsSome] :  $P \ [] \implies (\bigwedge f \ x \ b \ c \ \tau \ s' \ \Phi'. \ P \ \Phi' \implies P \ ((AF-fundef \ f \ (AF-fun-typ-none \ (AF-fun-typ \ x \ b \ c \ \tau \ s')) \ \# \ \Phi')) \implies$ 
 $(\bigwedge f \ bv \ x \ b \ c \ \tau \ s' \ \Phi'. \ P \ \Phi' \implies P \ ((AF-fundef \ f \ (AF-fun-typ-some \ bv \ (AF-fun-typ \ x \ b \ c \ \tau \ s')) \ \# \ \Phi')) \implies P \ \Phi$ 
proof(induct  $\Phi$  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  $\Theta$ -induct [case-names TNil AF-typedef AF-typedef-poly] :  $P \ [] \implies (\bigwedge tid \ dclist \ \Theta'. \ P \ \Theta' \implies P \ ((AF-typedef \ tid \ dclist) \ \# \ \Theta')) \implies$ 
 $(\bigwedge tid \ bv \ dclist \ \Theta'. \ P \ \Theta' \implies P \ ((AF-typedef-poly \ tid \ bv \ dclist) \ \# \ \Theta')) \implies P \ \Theta$ 
proof(induct  $\Theta$  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  $\Gamma :: pt$ 
begin

```

```

primrec permute- $\Gamma$ 
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- $\Gamma$ .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  $x \# (c, \Gamma)$  and atom  $x' \# (c, \Gamma)$ 
  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  $x \# (c, \Gamma)$  and atom  $x' \# (c, \Gamma)$ 
  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  $x \# \Gamma$  and atom  $x' \# \Gamma$ 
  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 *toSet* :: $\Gamma \Rightarrow (x*b*c)$ *set* **where**
 toSet *GNil* = {}
 | *toSet* (*GCons* *xbc* *G*) = {*xbc*} \cup (*toSet* *G*)
apply (*auto*, *simp* *add*: *eqvt-def toSet-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* # _{Γ} *g1*) *g2* = (*xbc* # _{Γ} (*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*' (*toSet* Γ))
apply *auto*
unfolding *eqvt-def dom-graph-aux-def lfp-eqvt toSet.eqvt* **by** *simp*
nominal-termination (*eqvt*) **by** *lexicographic-order*

Use of this is sometimes mixed in with use of freshness and support for the context however it makes it clear that for immutable variables, the context is ‘self-supporting’

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

3.2.3 Immutable Variable Context Lemmas

lemma *append-GNil[simp]*:
 GNil @ *G* = *G*
by *simp*

lemma *append-g-toSetU [simp]*: *toSet* (*G1*@*G2*) = *toSet* *G1* \cup *toSet* *G2*
by(*induct* *G1*, *auto*+))

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

lemma *supp-GCons*:
fixes $xs::\Gamma$
shows $\text{supp } (x \#_{\Gamma} xs) = \text{supp } x \cup \text{supp } xs$
by (*simp add: supp-def Collect-imp-eq Collect-neg-eq*)

lemma *atom-dom-eq[simp]*:
fixes $G::\Gamma$
shows $\text{atom-dom } ((x, b, c) \#_{\Gamma} G) = \text{atom-dom } ((x, b, c') \#_{\Gamma} G)$
using *atom-dom.simps toSet.simps* **by** *simp*

lemma *dom-append[simp]*:
 $\text{atom-dom } (\Gamma @ \Gamma') = \text{atom-dom } \Gamma \cup \text{atom-dom } \Gamma'$
using *image-Un append-g-toSetU atom-dom.simps dom.simps* **by** *metis*

lemma *dom-cons[simp]*:
 $\text{atom-dom } ((x, b, c) \#_{\Gamma} G) = \{ \text{atom } x \} \cup \text{atom-dom } G$
using *image-Un append-g-toSetU 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 $xs::\Gamma$
shows $a \# (x \#_{\Gamma} xs) \longleftrightarrow a \# x \wedge a \# xs$
by (*simp add: fresh-def supp-GCons*)

lemma *dom-supg-g[simp]*:
 $\text{atom-dom } G \subseteq \text{supp } G$
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 $xs::\Gamma$
shows $a \# (xs @ ys) \longleftrightarrow a \# xs \wedge a \# ys$
by (*induct xs*) (*simp-all add: fresh-GNil fresh-GCons*)

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

lemma *append-g-inside*:
fixes $xs::\Gamma$
shows $xs @ (x \#_{\Gamma} ys) = (xs @ (x \#_{\Gamma} GNil)) @ ys$
by(*induct xs, auto+*)

lemma *finite- Γ* :
 $\text{finite } (\text{toSet } \Gamma)$
by(*induct Γ rule: Γ -induct, auto*)

```

lemma supp-Γ:
  supp Γ = supp (toSet Γ)
proof(induct Γ rule: Γ-induct)
  case GNil
  then show ?case using supp-GNil toSet.simps
    by (simp add: supp-set-empty)
next
  case (GCons x b c Γ')
  then show ?case using supp-GCons toSet.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 ⊆ 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 toSet G ⊆ toSet G'
  shows supp G ⊆ supp G'
  using supp-Γ finite-Γ by (simp add: supp-of-subset assms)

```

```

lemma fresh-weakening[ms-fresh]:
  assumes toSet G ⊆ toSet G' and x # G'
  shows x # G
proof(rule ccontr)
  assume ¬ x # G
  hence x ∈ supp G using fresh-def by auto
  hence x ∈ 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 # Γ
  and e ∈ toSet Γ
  shows a # e
using assms by(induct Γ,auto simp add: fresh-GCons)

```

```

lemma fresh-gamma-append:
  fixes xs::Γ
  shows a # (xs @ ys) ⟷ a # xs ∧ a # 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 $\text{supp } (x,y,z) = \text{supp } x \cup (\text{supp } y \cup \text{supp } z)$ **using** *supp-Pair* **by** *metis*
 thus *?thesis* **by** *auto*
qed

lemma *supp-append-g*:
 fixes $xs::\Gamma$
 shows $\text{supp } (xs @ ys) = \text{supp } xs \cup \text{supp } ys$
by(*induct xs, auto simp add: supp-GNil supp-GCons*)

lemma *fresh-in-g[simp]*:
 fixes $\Gamma::\Gamma$ and $x'::x$
 shows $\text{atom } x' \# \Gamma' @ (x, b0, c0) \#_{\Gamma} \Gamma = (\text{atom } x' \notin \text{supp } \Gamma' \cup \text{supp } x \cup \text{supp } b0 \cup \text{supp } c0 \cup \text{supp } \Gamma)$
proof –
 have $\text{atom } x' \# \Gamma' @ (x, b0, c0) \#_{\Gamma} \Gamma = (\text{atom } x' \notin \text{supp } (\Gamma' @ ((x,b0,c0) \#_{\Gamma} \Gamma)))$
 using *fresh-def* **by** *auto*
 also have $\dots = (\text{atom } x' \notin \text{supp } \Gamma' \cup \text{supp } ((x,b0,c0) \#_{\Gamma} \Gamma))$ **using** *supp-append-g* **by** *fast*
 also have $\dots = (\text{atom } x' \notin \text{supp } \Gamma' \cup \text{supp } x \cup \text{supp } b0 \cup \text{supp } c0 \cup \text{supp } \Gamma)$ **using** *supp-GCons*
supp-append-g supp-triple **by** *auto*
 finally show *?thesis* **by** *fast*
qed

lemma *fresh-suffix[ms-fresh]*:
 fixes $\Gamma::\Gamma$
 assumes $\text{atom } x \# \Gamma' @ \Gamma$
 shows $\text{atom } x \# \Gamma$
using *assms* **by**(*induct* Γ' *rule: Γ -induct, auto simp add: append-g.simps fresh-GCons*)

lemma *not-GCons-self [simp]*:
 fixes $xs::\Gamma$
 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 $\text{atom } y \# (\Gamma' @ (x, b, c) \#_{\Gamma} \Gamma)$
 shows $\text{atom } y \# (\Gamma' @ \Gamma)$
using *assms* **by**(*induct* Γ' *rule: Γ -induct, auto simp add: fresh-GCons fresh-GNil*)

lemma *fresh-dom-free*:
 assumes $\text{atom } x \# \Gamma$
 shows $(x,b,c) \notin \text{toSet } \Gamma$
using *assms* **proof**(*induct* Γ *rule: Γ -induct*)

```

  case GNil
  then show ?case by auto
next
  case (GCons x' b' c'  $\Gamma'$ )
  hence  $x \neq x'$  using fresh-def fresh-GCons fresh-Pair supp-at-base by blast
  moreover have  $\text{atom } x \# \Gamma'$  using fresh-GCons GCons by auto
  ultimately show ?case using toSet.simps GCons by auto
qed

lemma  $\Gamma$ -set-intros:  $x \in \text{toSet } (x \#_{\Gamma} xs)$  and  $y \in \text{toSet } xs \implies y \in \text{toSet } (x \#_{\Gamma} xs)$ 
  by simp+

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

```

3.2.4 Mutable Variable Context Lemmas

```

lemma supp-DNil:
  shows  $\text{supp } DNil = \{\}$ 
  by (simp add: supp-def)

lemma supp-DCons:
  fixes  $xs :: \Delta$ 
  shows  $\text{supp } (x \#_{\Delta} xs) = \text{supp } x \cup \text{supp } xs$ 
  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  $xs :: \Delta$ 
  shows  $a \# (x \#_{\Delta} xs) \longleftrightarrow a \# x \wedge a \# xs$ 
  by (simp add: fresh-def supp-DCons)

```

```

instance  $\Delta :: fs$ 
by (standard, induct-tac x, simp-all add: supp-DNil supp-DCons finite-supp)

```

3.2.5 Lookup Functions

```

nominal-function lookup ::  $\Gamma \Rightarrow x \Rightarrow (b * c) \text{ option}$  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 ::  $\Gamma \Rightarrow x \Rightarrow c \Rightarrow \Gamma$  (-[ $\dashrightarrow$ ]) [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  $\Gamma$ .exhaust by metis
nominal-termination (eqvt) by lexicographic-order

```

Functions for looking up data-constructors in the Pi context

```

nominal-function lookup-fun ::  $\Phi \Rightarrow f \Rightarrow \text{fun-def option}$  where
  lookup-fun [] g = None
| lookup-fun ((AF-fundef f ft)# $\Pi$ ) g = (if (f=g) then Some (AF-fundef f ft) else lookup-fun  $\Pi$  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 ::  $\Theta \Rightarrow \text{string} \Rightarrow \text{type-def option}$  where
  lookup-td [] g = None
| lookup-td ((AF-typedef s lst) # ( $\Theta::\Theta$ )) g = (if (s = g) then Some (AF-typedef s lst) else lookup-td  $\Theta$  g)
| lookup-td ((AF-typedef-poly s bv lst) # ( $\Theta::\Theta$ )) g = (if (s = g) then Some (AF-typedef-poly s bv lst) else lookup-td  $\Theta$  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 ::  $\text{type-def} \Rightarrow f$  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 ::  $\text{fun-def} \Rightarrow f$  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  $\Rightarrow$  'a list  $\Rightarrow$  '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 :: $l \Rightarrow b$ 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: $(xs \neq DNil) = (\exists y \ ys. xs = y \#_{\Delta} ys)$
by (induct xs) auto

nominal-function setD :: $\Delta \Rightarrow (u*\tau)$ set where
 setD DNil = {}
 | setD (DCons xbc G) = {xbc} \cup (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 $atom\ y \# (xa, va)$ and $atom\ ya \# (xa, va)$ and
 $\forall c. atom\ c \# (s, sa) \longrightarrow atom\ c \# (y, ya, s, sa) \longrightarrow (y \leftrightarrow c) \cdot s = (ya \leftrightarrow c) \cdot sa$
 and $eqvt\text{-}at\ f\ (s, xa, va)$ and $eqvt\text{-}at\ f\ (sa, xa, va)$ and
 $atom\ c \# (s, va, xa, sa)$ and $atom\ c \# (y, ya, f\ (s, xa, va), f\ (sa, xa, va))$
 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 $\dots = f\ ((y \leftrightarrow c) \cdot s, (y \leftrightarrow c) \cdot xa, (y \leftrightarrow c) \cdot va)$ **by** auto
 also have $\dots = f\ ((ya \leftrightarrow c) \cdot sa, (ya \leftrightarrow c) \cdot xa, (ya \leftrightarrow c) \cdot va)$ **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 $\dots = f\ ((ya \leftrightarrow c) \cdot (sa, xa, va))$ **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 subst-v-simple-commute[simp]:  atom x # c ⟹ (c[z::=[x]v]v)[x::=b]v = c[z::=b]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-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 $(x \leftrightarrow x') \cdot (c1[z1 ::= [x]^v]_v) = c1'[z1' ::= [x']^v]_v$

proof –

have $atom\ x' \# c1[z1 ::= [x]^v]_v$ **using** *assms fresh-subst-v-if* **by** *simp*

hence $(x \leftrightarrow x') \cdot (c1[z1 ::= [x]^v]_v) = c1[z1 ::= [x]^v]_v[x ::= [x']^v]_v$ **using** *flip-subst-v[of x' c1[z1 ::= [x]^v]_v]*
x] flip-commute **by** *metis*

also have $\dots = c1[z1 ::= [x']^v]_v$ **using** *subst-v-simple-commute fresh-prod4 assms* **by** *auto*

also have $\dots = c1'[z1' ::= [x']^v]_v$ **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 :: $v \Rightarrow x \Rightarrow v \Rightarrow v$ **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 :: $v \Rightarrow x \Rightarrow v \Rightarrow v$ ($[- ::= -]_{vv} [1000, 50, 50] 1000$)

where

$v[x ::= v']_{vv} \equiv \text{subst-vv } v \ x \ v'$

lemma *fresh-subst-vv-if [simp]*:

$j \# t[i ::= x]_{vv} = ((atom\ i \# t \wedge j \# t) \vee (j \# x \wedge (j \# t \vee j = atom\ i)))$

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]*: $atom\ a \# tm \implies tm[a ::= x]_{vv} = tm$

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]*:

$atom\ j \# tm \implies tm[i ::= t]_{vv}[j ::= u]_{vv} = tm[i ::= t[j ::= u]_{vv}]_{vv}$

by (*induct* *tm* *rule: v.induct*) (*auto simp: fresh-Pair*)

```

lemma subst-vv-commute-full [simp]:
  atom j # t  $\implies$  atom i # u  $\implies$  i  $\neq$  j  $\implies$  tm[i::=t]vv[j::=u]vv = tm[j::=u]vv[i::=t]vv
  by (induct tm rule: v.induct) auto

lemma subst-vv-var-flip [simp]:
  fixes v::v
  assumes atom y # v
  shows (y  $\leftrightarrow$  x)  $\cdot$  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  $\wedge$  j # t)  $\vee$  (j # x  $\wedge$  (j # t  $\vee$  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  $\implies$  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  $\cdot$  subst-v t1 x1 v = subst-v (p  $\cdot$  t1) (p  $\cdot$  x1) (p  $\cdot$  v)
    using subst-v-v-def by simp

  fix x::x and c::v and z::x
  show atom x # c  $\implies$  ((x  $\leftrightarrow$  z)  $\cdot$  c) = c[z::=[x]]v
    using subst-v-v-def by simp

  fix x::x and c::v and z::x
  show atom x # c  $\implies$  c[z::=[x]]v[x::=v]v = c[z::=v]v
    using subst-v-v-def by simp
qed

end

```

4.3 Expressions

nominal-function *subst-ev* :: *e* \Rightarrow *x* \Rightarrow *v* \Rightarrow *e* **where**

```

  subst-ev ( (AE-val v') ) x v = ( (AE-val (subst-vv v' x v)) )
| subst-ev ( (AE-app f v') ) x v = ( (AE-app f (subst-vv v' x v)) )
| subst-ev ( (AE-appP f b v') ) x v = ( (AE-appP f b (subst-vv v' x v)) )
| subst-ev ( (AE-op opp v1 v2) ) x v = ( (AE-op opp (subst-vv v1 x v) (subst-vv v2 x v)) )
| subst-ev [#1 v]e x v = [#1 (subst-vv v' x v)]e
| subst-ev [#2 v]e x v = [#2 (subst-vv v' x v)]e
| subst-ev ( (AE-mvar u) ) x v = AE-mvar u
| subst-ev [| v' |]e x v = [| (subst-vv v' x v) |]e
| subst-ev ( AE-concat v1 v2 ) x v = AE-concat (subst-vv v1 x v) (subst-vv v2 x v)
| subst-ev ( AE-split v1 v2 ) x v = AE-split (subst-vv v1 x v) (subst-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-ev-abbrev :: $e \Rightarrow x \Rightarrow v \Rightarrow e$ $(-[:=-])_{ev}$ [1000,50,50] 500)

where

$e[x::=v]_{ev} \equiv \text{subst-ev } e \ x \ v'$

lemma *size-subst-ev* [simp]: $\text{size } (\text{subst-ev } A \ i \ x) = \text{size } A$

apply (nominal-induct A avoiding: i x rule: e.strong-induct)

apply auto

done

lemma *forget-subst-ev* [simp]: $\text{atom } a \ \sharp \ A \implies \text{subst-ev } A \ a \ x = A$

apply (nominal-induct A avoiding: a x rule: e.strong-induct)

apply(auto simp: fresh-at-base)

done

lemma *subst-ev-id* [simp]: $\text{subst-ev } A \ a \ (V\text{-var } a) = A$

by (nominal-induct A avoiding: a rule: e.strong-induct) (auto simp: fresh-at-base)

lemma *fresh-subst-ev-if* [simp]:

$j \ \sharp \ (\text{subst-ev } A \ i \ x) = ((\text{atom } i \ \sharp \ A \wedge j \ \sharp \ A) \vee (j \ \sharp \ x \wedge (j \ \sharp \ A \vee j = \text{atom } i)))$

apply (induct A rule: e.induct)

unfolding *subst-ev.simps fresh-subst-vv-if* **apply** auto+

using *pure-fresh fresh-opp-all* **apply** metis+

done

lemma *subst-ev-commute* [simp]:

$\text{atom } j \ \sharp \ A \implies (A[i::=t]_{ev})[j::=u]_{ev} = A[i::=t[j::=u]_{vv}]_{ev}$

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 $\text{atom } y \ \sharp \ e$

shows $(y \leftrightarrow x) \cdot e = e \ [x::=V\text{-var } y]_{ev}$

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 $\text{atom } c \# (e, ea)$ **and** $\text{atom } c \# (x, xa, e, ea)$ **and** $(x \leftrightarrow c) \cdot e = (xa \leftrightarrow c) \cdot ea$
shows $e[x::=v]_{ev} = ea[xa::=v]_{ev}$

proof –

have $e[x::=v]_{ev} = (e[x::=V\text{-var } c]_{ev})[c::=v]_{ev}$ **using** *subst-ev-commute* **assms** **by** *simp*
also have $\dots = ((c \leftrightarrow x) \cdot e)[c::=v]_{ev}$ **using** *subst-ev-var-flip* **assms** **by** *simp*
also have $\dots = ((c \leftrightarrow xa) \cdot ea)[c::=v]_{ev}$ **using** *assms flip-commute* **by** *metis*
also have $\dots = ea[xa::=v]_{ev}$ **using** *subst-ev-var-flip* **assms** **by** *simp*
finally show *?thesis* **by** *auto*

qed

lemma *subst-ev-var*[*simp*]:

$(AE\text{-val } (V\text{-var } x))[x::=[z]^v]_{ev} = AE\text{-val } (V\text{-var } z)$

by *auto*

instantiation $e :: \text{has-subst-v}$

begin

definition

$\text{subst-v} = \text{subst-ev}$

instance proof

fix $j::\text{atom}$ **and** $i::x$ **and** $x::v$ **and** $t::e$
show $(j \# \text{subst-v } t \ i \ x) = ((\text{atom } i \# t \wedge j \# t) \vee (j \# x \wedge (j \# t \vee j = \text{atom } i)))$
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 $\text{atom } a \# tm \implies \text{subst-v } tm \ a \ x = tm$

using *forget-subst-ev* *subst-v-e-def* **by** *simp*

fix $a::x$ **and** $tm::e$

show $\text{subst-v } tm \ a \ (V\text{-var } a) = tm$ **using** *subst-ev-id* *subst-v-e-def* **by** *simp*

fix $p::\text{perm}$ **and** $x1::x$ **and** $v::v$ **and** $t1::e$

show $p \cdot \text{subst-v } t1 \ x1 \ v = \text{subst-v } (p \cdot t1) \ (p \cdot x1) \ (p \cdot v)$

using *subst-ev-commute* *subst-v-e-def* **by** *simp*

fix $x::x$ **and** $c::e$ **and** $z::x$

show $\text{atom } x \# c \implies ((x \leftrightarrow z) \cdot c) = c[z::=[x]^v]_v$

using *subst-v-e-def* **by** *simp*

fix $x::x$ **and** $c::e$ **and** $z::x$

show $\text{atom } x \# c \implies c[z::=[x]^v]_v[x::=v]_v = c[z::=v]_v$

using *subst-v-e-def* **by** *simp*

qed
end

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

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

4.4 Expressions in Constraints

nominal-function *subst-cev* :: $ce \Rightarrow x \Rightarrow v \Rightarrow ce$ **where**
 $\text{subst-cev } ((CE\text{-val } v')) \ x \ v = ((CE\text{-val } (\text{subst-vv } v' \ x \ v)))$
 $\text{subst-cev } ((CE\text{-op } \text{opp } v1 \ v2)) \ x \ v = ((CE\text{-op } \text{opp } (\text{subst-cev } v1 \ x \ v) (\text{subst-cev } v2 \ x \ v)))$
 $\text{subst-cev } ((CE\text{-fst } v')) \ x \ v = CE\text{-fst } (\text{subst-cev } v' \ x \ v)$
 $\text{subst-cev } ((CE\text{-snd } v')) \ x \ v = CE\text{-snd } (\text{subst-cev } v' \ x \ v)$
 $\text{subst-cev } ((CE\text{-len } v')) \ x \ v = CE\text{-len } (\text{subst-cev } v' \ x \ v)$
 $\text{subst-cev } (CE\text{-concat } v1 \ v2) \ x \ v = CE\text{-concat } (\text{subst-cev } v1 \ x \ v) (\text{subst-cev } v2 \ x \ v)$
apply (*simp* add: *eqvt-def* *subst-cev-graph-aux-def*, *auto*)
by (*meson* $ce.\text{strong-exhaust}$)

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

abbreviation

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

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

lemma *forget-subst-cev* [*simp*]: $\text{atom } a \# A \implies \text{subst-cev } A \ a \ x = A$
by (*nominal-induct* A *avoiding*: $a \ x$ rule: $ce.\text{strong-induct}, \text{auto}$ *simp*: *fresh-at-base*)

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

lemma *fresh-subst-cev-if* [*simp*]:
 $j \# (\text{subst-cev } A \ i \ x) = ((\text{atom } i \# A \wedge j \# A) \vee (j \# x \wedge (j \# A \vee j = \text{atom } i)))$
proof(*nominal-induct* A *avoiding*: $i \ x$ rule: $ce.\text{strong-induct}$)
case ($CE\text{-op } \text{opp } v1 \ v2$)
then show ?*case* **using** *fresh-subst-vv-if* *subst-ev.simps* $e.\text{supp}$ *pure-fresh* *opp.fresh*

fresh-e-opp
using *fresh-opp-all* **by** *auto*
qed(*auto*)**+**

lemma *subst-cev-commute* [*simp*]:
 $atom\ j \# A \implies (subst-cev\ (subst-cev\ A\ i\ t)\ j\ u) = subst-cev\ A\ i\ (subst-vv\ t\ j\ u)$
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 $(y \leftrightarrow x) \cdot e = e\ [x::=V-var\ y]_{cev}$
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*: $atom\ y \# v1 \wedge atom\ y \# v2$ **using** *ce.fresh* **by** *blast*
have $(y \leftrightarrow x) \cdot (CE-op\ opp\ v1\ v2) = CE-op\ ((y \leftrightarrow x) \cdot opp)\ ((y \leftrightarrow x) \cdot v1)\ ((y \leftrightarrow x) \cdot v2)$
using *opp.perm-simps ce.perm-simps permute-pure ce.fresh opp.strong-exhaust* **by** *presburger*
also have ... = $CE-op\ ((y \leftrightarrow x) \cdot opp)\ (v1\ [x::=V-var\ y]_{cev})\ (v2\ [x::=V-var\ y]_{cev})$ **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)*)
qed((*auto simp add: permute-pure subst-vv-var-flip*)**+**)

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** $(x \leftrightarrow c) \cdot e = (xa \leftrightarrow c) \cdot ea$
shows $e[x::=v]_{cev} = ea[xa::=v]_{cev}$
proof –
have $e[x::=v]_{cev} = (e[x::=V-var\ c]_{cev})[c::=v]_{cev}$ **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 $(j \# \text{subst-}v \ t \ i \ x) = ((\text{atom } i \# t \wedge j \# t) \vee (j \# x \wedge (j \# t \vee j = \text{atom } i)))$
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 $\text{atom } a \# tm \implies \text{subst-}v \ tm \ a \ x = tm$
using *forget-subst-cev* *subst-v-ce-def* **by** *simp*

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

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

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

fix *x::x* **and** *c::ce* **and** *z::x*
show $\text{atom } x \# c \implies c \ [z::=V\text{-var } x]_v[x::=v]_v = c[z::=v]_v$
using *subst-v-ce-def* **by** *simp*

qed

end

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

lemma *subst-cev-v-flip1*[*simp*]:
fixes *e::ce*
assumes $\text{atom } z1 \# (z, e)$ **and** $\text{atom } z1' \# (z, e)$
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**
 $\text{subst-cv} \ (C\text{-true}) \ x \ v = C\text{-true}$
 $\text{subst-cv} \ (C\text{-false}) \ x \ v = C\text{-false}$
 $\text{subst-cv} \ (C\text{-conj } c1 \ c2) \ x \ v = C\text{-conj} \ (\text{subst-cv } c1 \ x \ v) \ (\text{subst-cv } c2 \ x \ v)$
 $\text{subst-cv} \ (C\text{-disj } c1 \ c2) \ x \ v = C\text{-disj} \ (\text{subst-cv } c1 \ x \ v) \ (\text{subst-cv } c2 \ x \ v)$
 $\text{subst-cv} \ (C\text{-imp } c1 \ c2) \ x \ v = C\text{-imp} \ (\text{subst-cv } c1 \ x \ v) \ (\text{subst-cv } c2 \ x \ v)$
 $\text{subst-cv} \ (e1 == e2) \ x \ v = ((\text{subst-}cev \ e1 \ x \ v) == (\text{subst-}cev \ e2 \ x \ v))$
 $\text{subst-cv} \ (C\text{-not } c) \ x \ v = C\text{-not} \ (\text{subst-cv } c \ x \ v)$
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 :: $c \Rightarrow x \Rightarrow v \Rightarrow c \text{ } (-[::=])_{cv} [1000, 50, 50] 1000$

where

$c[x::=v']_{cv} \equiv \text{subst-cv } c \ x \ v'$

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

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

apply *auto*

done

lemma *forget-subst-cv* [*simp*]: $\text{atom } a \ \sharp \ A \Longrightarrow \text{subst-cv } A \ a \ x = A$

apply (*nominal-induct* *A* *avoiding*: *a x* *rule*: *c.strong-induct*)

apply (*auto simp*: *fresh-at-base*)

done

lemma *subst-cv-id* [*simp*]: $\text{subst-cv } A \ a \ (V\text{-var } a) = A$

by (*nominal-induct* *A* *avoiding*: *a* *rule*: *c.strong-induct*) (*auto simp*: *fresh-at-base*)

lemma *fresh-subst-cv-if* [*simp*]:

$j \ \sharp \ (\text{subst-cv } A \ i \ x) \longleftrightarrow (\text{atom } i \ \sharp \ A \wedge j \ \sharp \ A) \vee (j \ \sharp \ x \wedge (j \ \sharp \ A \vee j = \text{atom } i))$

by (*nominal-induct* *A* *avoiding*: *i x* *rule*: *c.strong-induct*, (*auto simp add*: *pure-fresh*)+)

lemma *subst-cv-commute* [*simp*]:

$\text{atom } j \ \sharp \ A \Longrightarrow (\text{subst-cv } (\text{subst-cv } A \ i \ t) \ j \ u) = \text{subst-cv } A \ i \ (\text{subst-vv } t \ j \ u)$

by (*nominal-induct* *A* *avoiding*: *i j t u* *rule*: *c.strong-induct*) (*auto simp*: *fresh-at-base*)

lemma *let-s-size* [*simp*]: $\text{size } s \leq \text{size } (AS\text{-let } x \ e \ s)$

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 $\text{atom } y \ \sharp \ c$

shows $(y \leftrightarrow x) \cdot c = c[x::=V\text{-var } y]_{cv}$

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 $(j \# \text{subst-}v \ t \ i \ x) = ((\text{atom } i \# t \wedge j \# t) \vee (j \# x \wedge (j \# t \vee j = \text{atom } i)))$
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 $\text{atom } a \# tm \implies \text{subst-}v \ tm \ a \ x = tm$
using *forget-subst-cv* *subst-v-c-def* **by** *simp*

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

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

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

fix $x::x$ **and** $c::c$ **and** $z::x$
show $\text{atom } x \# c \implies c[z::=[x]^v]_v[x::=v]_v = c[z::=v]_v$
using *subst-cv-var-flip* *subst-v-c-def* **by** *simp*

qed

end

lemma *subst-cv-var-flip1* [*simp*]:
fixes $c::c$
assumes $\text{atom } y \# c$
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-flip3* [*simp*]:
fixes $c::c$
assumes $\text{atom } z1 \# c$ **and** $\text{atom } z1' \# c$
shows $(z1 \leftrightarrow z1') \cdot c[z::=[z1]^v]_{cv} = c[z::=[z1']^v]_{cv}$
proof –
consider $z1' = z \mid z1 = z \mid \text{atom } z1 \# z \wedge \text{atom } z1' \# z$ **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** *assms* **by** *auto*
qed
qed

lemma *subst-cv-v-flip*[simp]:
fixes $c::c$
assumes $\text{atom } x \# c$
shows $((x \leftrightarrow z) \cdot c)[x::=v]_{cv} = c [z::=v]_{cv}$
using *assms subst-v-c-def* **by** *auto*

lemma *subst-cv-commute-full*:
fixes $c::c$
assumes $\text{atom } z \# v$ **and** $\text{atom } x \# w$ **and** $x \neq z$
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-full* **by** *simp*
qed(*force+*)

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

4.6 Variable Context

nominal-function *subst-gv* :: $\Gamma \Rightarrow x \Rightarrow v \Rightarrow \Gamma$ **where**
 $\text{subst-gv } GNil \ x \ v = GNil$
 $| \text{subst-gv } ((y,b,c) \#_{\Gamma} \Gamma) \ x \ v = (\text{if } x = y \text{ then } \Gamma \text{ else } ((y,b,c[x::=v]_{cv}) \#_{\Gamma} (\text{subst-gv } \Gamma \ x \ v)))$
proof(*goal-cases*)
case 1
then show $?case$ **by**(*simp add: eqvt-def subst-gv-graph-aux-def*)
next
case ($\exists P \ x$)
then show $?case$ **by** (*metis neq-GNil-conv prod-cases3*)
qed(*fast+*)
nominal-termination (*eqvt*) **by** *lexicographic-order*

abbreviation
 $\text{subst-gv-abbrev} :: \Gamma \Rightarrow x \Rightarrow v \Rightarrow \Gamma \text{ } (-[::=]_{\Gamma v} [1000,50,50] \ 1000)$
where
 $g[x::=v]_{\Gamma v} \equiv \text{subst-gv } g \ x \ v$

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

lemma *forget-subst-gv* [simp]: $\text{atom } a \# G \implies \text{subst-gv } G \ a \ x = G$
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*: $\text{atom } a \# G \implies \text{atom } a \# v \implies \text{atom } a \# \text{subst-gv } G \ x \ v$

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 $\text{atom } a \# G$ 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 $\Gamma::\Gamma$

assumes $\text{atom } xa \# ((x, b, c[z::=[x]^v]_{cv}) \#_{\Gamma} \Gamma)$ and $\text{atom } xa \# \Gamma$ and $\text{atom } x \# \Gamma$ and $\text{atom } x \# (z, c)$ and $\text{atom } xa \# (z, c)$

shows $(x \leftrightarrow xa) \cdot ((x, b, c[z::=[x]^v]_{cv}) \#_{\Gamma} \Gamma) = (xa, b, c[z::=V\text{-var } xa]_{cv}) \#_{\Gamma} \Gamma$

proof –

have $(x \leftrightarrow xa) \cdot ((x, b, c[z::=[x]^v]_{cv}) \#_{\Gamma} \Gamma) = ((x \leftrightarrow xa) \cdot x, b, (x \leftrightarrow xa) \cdot c[z::=[x]^v]_{cv}) \#_{\Gamma} ((x \leftrightarrow xa) \cdot \Gamma)$

using *subst Cons-eqvt flip-fresh-fresh* using *G-cons-flip* by *simp*

also have ... = $((xa, b, (x \leftrightarrow xa) \cdot c[z::=[x]^v]_{cv}) \#_{\Gamma} ((x \leftrightarrow xa) \cdot \Gamma))$ using *assms* by *fastforce*

also have ... = $((xa, b, c[z::=V\text{-var } xa]_{cv}) \#_{\Gamma} ((x \leftrightarrow xa) \cdot \Gamma))$ using *assms* *subst-cv-var-flip* by *fastforce*

also have ... = $((xa, b, c[z::=V\text{-var } xa]_{cv}) \#_{\Gamma} \Gamma)$ 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

$\text{atom } z \# (x, v) \implies \text{subst-tv } \{z : b \mid c\} \ x \ v = \{z : b \mid c[x::=v]_{cv}\}$

apply (*simp* *add: eqvt-def subst-tv-graph-aux-def*)

apply *auto*

apply(*rule-tac* $y=a$ and $c=(aa, b)$ in $\tau.\text{strong-exhaust}$)

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 $a1: \text{atom } za \# va$ and $a2: \text{atom } z \# va$ and $a3: \forall cb. \text{atom } cb \# c \wedge \text{atom } cb \# ca \longrightarrow cb \neq z \wedge cb \neq za \longrightarrow c[z::=V\text{-var } cb]_{cv} = ca[za::=V\text{-var } cb]_{cv}$

assume $a4: \text{atom } cb \# c$ and $a5: \text{atom } cb \# ca$ and $a6: cb \neq z$ and $a7: cb \neq za$ and $\text{atom } cb \# va$ 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  $c[z::=V\text{-var } cb]_{cv} = ca[za::=V\text{-var } cb]_{cv}$  using assms by auto
hence  $c[z::=V\text{-var } cb]_{cv}[xa::=va]_{cv} = ca[za::=V\text{-var } cb]_{cv}[xa::=va]_{cv}$  by simp
moreover have  $c[z::=V\text{-var } cb]_{cv}[xa::=va]_{cv} = c[xa::=va]_{cv}[z::=V\text{-var } cb]_{cv}$  using subst-cv-commute-full[of
z va xa V-var cb ] assms fresh-def v.supp by fastforce
moreover have  $ca[za::=V\text{-var } cb]_{cv}[xa::=va]_{cv} = ca[xa::=va]_{cv}[za::=V\text{-var } cb]_{cv}$ 
using subst-cv-commute-full[of za va xa V-var cb ] assms fresh-def v.supp by fastforce

ultimately show  $c[xa::=va]_{cv}[z::=V\text{-var } cb]_{cv} = ca[xa::=va]_{cv}[za::=V\text{-var } cb]_{cv}$  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
   $t[x::=v]_{\tau v} \equiv \text{subst-tv } t \ x \ v$ 

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

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

lemma subst-tv-id [simp]:  $\text{subst-tv } A \ a \ (V\text{-var } a) = A$ 
by (nominal-induct A avoiding: a rule: \tau.strong-induct) (auto simp: fresh-at-base)

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

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

lemma subst-tv-var-flip [simp]:
  fixes  $x::x$  and  $xa::x$  and  $\tau::\tau$ 
assumes  $\text{atom } xa \ \sharp \ \tau$ 
shows  $(x \leftrightarrow xa) \cdot \tau = \tau[x::=V\text{-var } xa]_{\tau v}$ 
proof –
  obtain  $z::x$  and  $b$  and  $c$  where  $zbc: \text{atom } z \ \sharp \ (x, xa, V\text{-var } xa) \wedge \tau = \llbracket z : b \mid c \rrbracket$ 
using obtain-fresh-z by (metis prod.inject subst-tv.cases)
hence  $\text{atom } xa \notin \text{supp } c - \{ \text{atom } z \}$  using  $\tau.\text{supp}[of \ z \ b \ c]$  fresh-def supp-b-empty assms
by auto
moreover have  $xa \neq z$  using zbc fresh-prod3 by force

```

ultimately have $xaf: atom\ xa \# c$ **using** *fresh-def* **by** *auto*
 have $(x \leftrightarrow xa) \cdot \tau = \llbracket z : b \mid (x \leftrightarrow xa) \cdot c \rrbracket$
by (*metis* $\tau.perm\text{-}simps\ empty\text{-}iff\ flip\text{-}at\text{-}base\text{-}simps(\mathcal{I})\ flip\text{-}fresh\text{-}fresh\ fresh\text{-}PairD(1)\ fresh\text{-}PairD(2)$
fresh-def not-self-fresh supp-b-empty v.fresh(2) zbc)
 also have $\dots = \llbracket z : b \mid c[x::=V\text{-}var\ xa]_{cv} \rrbracket$ **using** *subst-cv-v-flip* xaf
by (*metis* *permute-flip-cancel permute-flip-cancel2 subst-cv-var-flip*)
 finally show *?thesis* **using** *subst-tv.simps zbc*
using *fresh-PairD(1) not-self-fresh* **by** *force*
 qed

instantiation $\tau :: has\text{-}subst\text{-}v$
 begin

definition

subst-v = *subst-tv*

instance proof

fix $j::atom$ and $i::x$ and $x::v$ and $t::\tau$

show $(j \# subst\text{-}v\ t\ i\ x) = ((atom\ i \# t \wedge j \# t) \vee (j \# x \wedge (j \# t \vee j = atom\ i)))$

proof(*nominal-induct t avoiding: i x rule:\tau.strong-induct*)

case (*T-refined-type z b c*)

hence $j \# \llbracket z : b \mid c \rrbracket[i::=x]_v = j \# \llbracket z : b \mid c[i::=x]_{cv} \rrbracket$ **using** *subst-tv.simps subst-v-\tau-def*
fresh-Pair **by** *simp*

also have $\dots = (atom\ i \# \llbracket z : b \mid c \rrbracket \wedge j \# \llbracket z : b \mid c \rrbracket \vee j \# x \wedge (j \# \llbracket z : b \mid c \rrbracket \vee j = atom\ i))$

unfolding $\tau.fresh$ **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 $tm::\tau$ and $x::v$

show $atom\ a \# tm \implies subst\text{-}v\ tm\ a\ x = tm$

apply(*nominal-induct tm avoiding: a x rule:\tau.strong-induct*)

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

fix $a::x$ and $tm::\tau$

show $subst\text{-}v\ tm\ a\ (V\text{-}var\ a) = tm$

apply(*nominal-induct tm avoiding: a rule:\tau.strong-induct*)

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

fix $p::perm$ and $x1::x$ and $v::v$ and $t1::\tau$

show $p \cdot subst\text{-}v\ t1\ x1\ v = subst\text{-}v\ (p \cdot t1)\ (p \cdot x1)\ (p \cdot v)$

apply(*nominal-induct tm avoiding: a x rule:\tau.strong-induct*)

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

fix $x::x$ and $c::\tau$ and $z::x$

show $atom\ x \# c \implies ((x \leftrightarrow z) \cdot c) = c[z::=[x]^v]_v$

apply(*nominal-induct c avoiding: z x rule:\tau.strong-induct*)

using *subst-v-c-def flip-subst-v subst-tv.simps subst-v-\tau-def fresh-Pair* **by** *auto*

fix $x::x$ and $c::\tau$ and $z::x$


```

show  $atom\ x \# c \implies c[z ::= [x]^v]_v[x ::= v]_v = c[z ::= v]_v$ 
  apply(nominal-induct c avoiding:  $x\ v\ z$  rule:  $\tau$ .strong-induct)
  using subst-v-c-def subst-tv.simps subst-v- $\tau$ -def fresh-Pair
  by (metis flip-commute subst-tv-commute subst-tv-var-flip subst-v- $\tau$ -def subst-vv.simps(2))
qed

```

end

lemma *subst-tv-commute-full*:

```

fixes  $c :: \tau$ 
assumes  $atom\ z \# v$  and  $atom\ x \# w$  and  $x \neq z$ 
shows  $(c[z ::= w]_{\tau v})[x ::= v]_{\tau v} = (c[x ::= v]_{\tau v})[z ::= w]_{\tau v}$ 
using assms proof(nominal-induct c avoiding:  $x\ v\ z\ w$  rule:  $\tau$ .strong-induct)
case (T-refined-type  $x1a\ x2a\ x3a$ )
then show ?case using subst-cv-commute-full by simp
qed

```

lemma *type-eq-subst-eq*:

```

fixes  $v :: v$  and  $c1 :: c$ 
assumes  $\{ z1 : b1 \mid c1 \} = \{ z2 : b2 \mid c2 \}$ 
shows  $c1[z1 ::= v]_{cv} = c2[z2 ::= v]_{cv}$ 
using subst-v-flip-eq-two[of  $z1\ c1\ z2\ c2\ v$ ]  $\tau$ .eq-iff assms subst-v-c-def by simp

```

nominal-function *c-of* $:: \tau \Rightarrow x \Rightarrow c$ **where**

$atom\ z \# x \implies c\text{-of}\ (T\text{-refined-type}\ z\ b\ c)\ x = c[z ::= [x]^v]_{cv}$

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  $x1 :: \tau$  and  $x2 :: x$  where  $*:x = (x1, x2)$  by force
obtain  $z'$  and  $b'$  and  $c'$  where  $x1 = \{ z' : b' \mid c' \} \wedge atom\ z' \# x2$  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  $\tau$ .eq-iff by (metis prod.inject type-eq-subst-eq)
qed

```

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

lemma *c-of-eq*:

```

shows  $c\text{-of}\ \{ x : b \mid c \} x = c$ 
proof(nominal-induct  $\{ x : b \mid c \}$  avoiding:  $x$  rule:  $\tau$ .strong-induct)
case (T-refined-type  $x'\ c'$ )

```

moreover **hence** $c\text{-of}\ \{ x' : b \mid c' \} x = c'[x' ::= V\text{-var}\ x]_{cv}$ **using** *c-of.simps* **by** *auto*

moreover **have** $\{ x' : b \mid c' \} = \{ x : b \mid c \}$ **using** *T-refined-type* τ .*eq-iff* **by** *metis*

moreover have $c'[x'::=V\text{-var } x]_{cv} = c$ **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** $atom\ z \# t \wedge \tau = \llbracket z : b\text{-of } \tau \mid c\text{-of } \tau\ z \rrbracket$

proof –

obtain z **and** c **where** $atom\ z \# t \wedge \tau = \llbracket z : b\text{-of } \tau \mid c \rrbracket$ **using** *obtain-fresh-z2* **by** *metis*

moreover hence $c = c\text{-of } \tau\ z$ **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 $atom\ x \# c\text{-of } t\ z$

proof –

obtain z' **and** c' **where** $z:t = \llbracket z' : b\text{-of } t \mid c' \rrbracket \wedge atom\ z' \# (x, z)$ **using** *obtain-fresh-z-c-of* **by** *metis*

hence $*:c\text{-of } t\ z = c'[z'::=V\text{-var } z]_{cv}$ **using** *c-of.simps fresh-Pair* **by** *metis*

have $(atom\ x \# c' \vee atom\ x \in set\ [atom\ z']) \wedge atom\ x \# b\text{-of } t$ **using** $\tau.fresh\ assms\ z\ fresh\text{-Pair}$ **by** *metis*

hence $atom\ x \# c'$ **using** *fresh-Pair z fresh-at-base(2)* **by** *fastforce*

moreover have $atom\ x \# V\text{-var } z$ **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 $(c\text{-of } t\ z)[z::=V\text{-var } x]_{cv} = c\text{-of } t\ x$

proof –

obtain z' **and** c' **where** $z:t = \llbracket z' : b\text{-of } t \mid c' \rrbracket \wedge atom\ z' \# (x, z)$ **using** *obtain-fresh-z-c-of* **by** *metis*

hence $(atom\ z \# c' \vee atom\ z \in set\ [atom\ z']) \wedge atom\ z \# b\text{-of } t$ **using** $\tau.fresh[of\ atom\ z\ z'\ b\text{-of } t\ c']$

assms by metis

moreover have $atom\ z \notin set\ [atom\ z']$ **using** *z fresh-Pair by force*

ultimately have $** : atom\ z \# c'$ **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 $\dots = c'[z'::=V\text{-var } x]_{cv}$ **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

thm *type-eq-subst-eq*

lemma *type-eq-subst-eq1*:

fixes $v::v$ **and** $c1::c$
assumes $\{ \{ z1 : b1 \mid c1 \} = (\{ z2 : b2 \mid c2 \}) \text{ and } \text{atom } z1 \# c2$
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** $\tau.\text{eq-iff}$ *assms* **by** *blast*
have $z1 = z2 \wedge c1 = c2 \vee z1 \neq z2 \wedge c1 = (z1 \leftrightarrow z2) \cdot c2 \wedge \text{atom } z1 \# c2$
using $\tau.\text{eq-iff}$ *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** $[[\text{atom } z1]]\text{lst. } c1 = [[\text{atom } z2]]\text{lst. } c2$
proof –
show $c1[z1::=v]_{cv} = c2[z2::=v]_{cv}$ **using** *type-eq-subst-eq* *assms* **by** *blast*
show $b1=b2$ **using** $\tau.\text{eq-iff}$ *assms* **by** *blast*
show $[[\text{atom } z1]]\text{lst. } c1 = [[\text{atom } z2]]\text{lst. } c2$
using $\tau.\text{eq-iff}$ *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** $\text{atom } z1 \# c2$
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 $\text{atom } x \# c$
shows $\{ \{ z : b \mid c \} = \{ \{ x : b \mid (x \leftrightarrow z) \cdot c \} \}$
using $\tau.\text{eq-iff}$ *Abs1-eq-iff* *assms*
by (*metis* (*no-types*, *lifting*) *flip-fresh-fresh*)

lemma *c-of-true*:
 $c\text{-of } \{ \{ z' : B\text{-bool} \mid \text{TRUE} \} = x = C\text{-true}$
proof(*nominal-induct* $\{ \{ z' : B\text{-bool} \mid \text{TRUE} \}$ *avoiding*: x *rule*: $\tau.\text{strong-induct}$)
case (*T-refined-type* $x1a\ x3a$)
hence $\{ \{ z' : B\text{-bool} \mid \text{TRUE} \} = \{ \{ x1a : B\text{-bool} \mid x3a \} \}$ **using** $\tau.\text{eq-iff}$ **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 $\text{atom } x \# c$
shows $\{ \{ z : b \mid c \} = \{ \{ x : b \mid c[z::=[x]^v]_{cv} \} \}$

using $\tau.eq\text{-iff}$ *Abs1-eq-iff* *assms*
 using *subst-cv-var-flip type-eq-flip* **by** *auto*

lemma *type-e-subst-fresh*:

fixes $x::x$ **and** $z::x$
assumes $atom\ z \# (x,v)$ **and** $atom\ x \# e$
shows $\{ z : b \mid CE\text{-val}\ (V\text{-var}\ z) == e \} [x::=v]_{\tau v} = \{ z : b \mid CE\text{-val}\ (V\text{-var}\ z) == e \}$
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 $\{ z : b \mid CE\text{-val}\ (V\text{-var}\ z) == CE\text{-val}\ v' \} [x::=v]_{\tau v} = \{ z : b \mid CE\text{-val}\ (V\text{-var}\ z) == CE\text{-val}\ v' \}$
using *assms subst-tv.simps subst-cv.simps* **by** *simp*

lemma *subst-tbase-eq*:

$b\text{-of}\ \tau = b\text{-of}\ \tau[x::=v]_{\tau v}$

proof –

obtain z **and** b **and** c **where** $zbc: \tau = \{ z:b|c \} \wedge atom\ z \# (x,v)$ **using** $\tau.exhaust$
by (*metis prod.inject subst-tv.cases*)
hence $b\text{-of}\ \{ z:b|c \} = b\text{-of}\ \{ z:b|c \} [x::=v]_{\tau v}$ **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 $\{ z1 : b \mid CE\text{-val}\ (v'[x::=v]_{vv}) == CE\text{-val}\ (V\text{-lit}\ l) \text{ IMP } (c'[x::=v]_{cv})[z'::=[z1]^v]_{cv} \} =$
 $\{ z1 : b \mid CE\text{-val}\ v' == CE\text{-val}\ (V\text{-lit}\ l) \text{ IMP } c'[z'::=[z1]^v]_{cv} \} [x::=v]_{\tau v}$
using *subst-cv-commute-full[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 $\{ za : B\text{-id}\ tid \mid TRUE \} = \{ za : B\text{-id}\ tid \mid TRUE \} [x::=v]_{\tau v}$
using *assms subst-tv.simps subst-cv.simps* **by** *presburger*

lemma *b-of-subst*:

$b\text{-of}\ (\tau[x::=v]_{\tau v}) = b\text{-of}\ \tau$

proof –

obtain $z\ b\ c$ **where** $*:\tau = \{ z : b \mid c \} \wedge atom\ z \# (x,v)$ **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** $atom\ x \# (v,\tau)$ **and** $atom\ x' \# (v,\tau)$
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' ↔ x) τ' x v]* *assms* **by** *auto*

qed

lemma *subst-cv-true*:

$\llbracket z : B\text{-id } tid \mid TRUE \rrbracket = \llbracket z : B\text{-id } tid \mid TRUE \rrbracket[x::=v]_{\tau v}$

proof –

obtain $za::x$ **where** $atom\ za \# (x,v)$ **using** *obtain-fresh* **by** *auto*

hence $\llbracket z : B\text{-id } tid \mid TRUE \rrbracket = \llbracket za : B\text{-id } tid \mid TRUE \rrbracket$ **using** $\tau.eq\text{-iff}$ *Abs1-eq-iff* **by** *fastforce*

moreover have $\llbracket za : B\text{-id } tid \mid TRUE \rrbracket = \llbracket za : B\text{-id } tid \mid TRUE \rrbracket[x::=v]_{\tau v}$

using *subst-cv.simps* *subst-tv.simps* **by** (*simp add*: $\langle atom\ za \# (x, v) \rangle$)

ultimately show *?thesis* **by** *argo*

qed

lemma *t-eq-supp*:

assumes $(\llbracket z : b \mid c \rrbracket) = (\llbracket z1 : b1 \mid c1 \rrbracket)$

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** $\tau.eq\text{-iff}$ **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 $(\llbracket z : b \mid c \rrbracket) = (\llbracket zz : b \mid cc \rrbracket)$ **and** $atom\ x \# c$ **and** $x \neq zz$

shows $atom\ x \# cc$

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 $DNil\ x\ v = DNil$

\mid *subst-dv* $((u,t) \#_{\Delta} \Delta)\ x\ v = ((u,t[x::=v]_{\tau v}) \#_{\Delta} (subst-dv\ \Delta\ x\ v))$

apply (*simp add*: *eqvt-def subst-dv-graph-aux-def,auto*)

using *delete-aux.elims* **by** (*metis* $\Delta.exhaust\ surj\text{-pair}$)

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

abbreviation

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

where

$$\Delta[x::=v]_{\Delta v} \equiv \text{subst-dv } \Delta \ x \ v$$

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 ($\text{simp add: eqvt-def dmap-graph-aux-def, auto}$)

using delete-aux.elims **by** ($\text{metis } \Delta.\text{exhaust surj-pair}$)

nominal-termination (eqvt) **by** $\text{lexicographic-order}$

lemma subst-dv-iff :

$\Delta[x::=v]_{\Delta v} = dmap \ (\lambda(u,t). (u, t[x::=v]_{\tau v})) \ \Delta$

by($\text{induct } \Delta, \text{ auto}$)

lemma size-subst-dv [simp]: $\text{size} \ (\ \text{subst-dv } G \ i \ x) \leq \text{size } G$

by ($\text{induct } G, \text{ auto}$)

lemma forget-subst-dv [simp]: $\text{atom } a \ \# \ G \Longrightarrow \text{subst-dv } G \ a \ x = G$

apply ($\text{induct } G, \text{ auto}$)

using $\text{fresh-DCons fresh-PairD}(1) \text{ not-self-fresh}$ **apply** fastforce

apply ($\text{simp add: fresh-DCons}$)**+**

done

lemma subst-dv-member :

assumes $(u, \tau) \in \text{setD } \Delta$

shows $(u, \tau[x::=v]_{\tau v}) \in \text{setD } (\Delta[x::=v]_{\Delta v})$

using assms **by**($\text{induct } \Delta \text{ rule: } \Delta\text{-induct, auto}$)

lemma fresh-subst-dv :

fixes $x::x$

assumes $\text{atom } xa \ \# \ \Delta$ **and** $\text{atom } xa \ \# \ v$

shows $\text{atom } xa \ \# \ \Delta[x::=v]_{\Delta v}$

using assms **proof**($\text{induct } \Delta \text{ rule: } \Delta\text{-induct}$)

case $DNil$

then show $?case$ **by** auto

next

case ($DCons \ u \ t \ \Delta$)

then show $?case$ **using** $\text{subst-dv.simps subst-v-}\tau\text{-def fresh-DCons fresh-Pair}$ **by** simp

qed

lemma fresh-subst-dv-if :

fixes $j::\text{atom}$ **and** $i::x$ **and** $x::v$ **and** $t::\Delta$

assumes $j \ \# \ t \wedge j \ \# \ x$

shows $(j \ \# \ \text{subst-dv } t \ i \ x)$

using assms **proof**($\text{induct } t \text{ rule: } \Delta\text{-induct}$)

case $DNil$

then show $?case$ **using** $\text{subst-gv.simps fresh-GNil}$ **by** auto

next

case ($DCons \ u' \ t' \ D'$)

then show $?case$ **unfolding** subst-dv.simps **using** $\text{fresh-DCons fresh-subst-tw-if fresh-Pair}$ **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 ( $\lambda x. \text{Inl undefined}$ ) (case-sum ( $\lambda x. \text{Inl undefined}$ ) ( $\lambda x. \text{Inr unde-}$ 
defined)))
subst-sv ::  $s \Rightarrow x \Rightarrow v \Rightarrow s$ 
and subst-branchv ::  $\text{branch-}s \Rightarrow x \Rightarrow v \Rightarrow \text{branch-}s$ 
and subst-branchlv ::  $\text{branch-list} \Rightarrow x \Rightarrow v \Rightarrow \text{branch-list}$  where
  subst-sv ( (AS-val v') ) x v = (AS-val (subst-vv v' x v ))
  | atom y # (x,v)  $\Longrightarrow$  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)  $\Longrightarrow$  subst-sv (AS-let2 y t s1 s2) x v = (AS-let2 y (t[x::=v]tv) (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)  $\Longrightarrow$  subst-sv (AS-var u  $\tau$  v' s) x v = AS-var u (subst-tv  $\tau$  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)  $\Longrightarrow$  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:  $\bigwedge s \ x a \ v a . \text{eqvt-at subst-sv-subst-branchv-subst-branchlv-sumC (Inl (s, xa, va))} \Longrightarrow$ 
  eqvt-at ( $\lambda a. \text{projl (subst-sv-subst-branchv-subst-branchlv-sumC (Inl a))}$ ) (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.proj1 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  $\sharp$  (xa, va)  $\wedge$  atom ua  $\sharp$  (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 # A \implies subst-sv A a x = A and atom a # B \implies subst-branchv B a x = B and atom a # C \implies 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 # s \wedge j # s) \vee (j # v \wedge (j # s \vee j = atom x)) \implies j # (subst-sv s x v) **and**
 (atom x # cs \wedge j # cs) \vee (j # v \wedge (j # cs \vee j = atom x)) \implies j # (subst-branchv cs x v) **and**
 (atom x # css \wedge j # css) \vee (j # v \wedge (j # css \vee j = atom x)) \implies j # (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 # (subst-sv s x v) \implies (atom x # s \wedge j # s) \vee (j # v \wedge (j # s \vee j = atom x)) **and**

j # (subst-branchv cs x v) \implies (atom x # cs \wedge j # cs) \vee (j # v \wedge (j # cs \vee j = atom x)) **and**

j # (subst-branchlv css x v) \implies (atom x # css \wedge j # css) \vee (j # v \wedge (j # 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 $((j \# s'[x::=v]_{sv} \vee j \in \text{set } [\text{atom } y]) \wedge j \# \text{None} \wedge j \# e[x::=v]_{ev})$ **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 ($\tau[x::=v]_{\tau v}$) (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) \longleftrightarrow (atom x $\#$ s \wedge j $\#$ s) \vee (j $\#$ v \wedge (j $\#$ s \vee j = atom x))* **and**
j $\#$ (subst-branchv cs x v) \longleftrightarrow (atom x $\#$ cs \wedge j $\#$ cs) \vee (j $\#$ v \wedge (j $\#$ cs \vee 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 \implies (subst-sv (subst-sv A i t) j u) = subst-sv A i (subst-vv t j u)* **and**
atom j $\#$ B \implies (subst-branchv (subst-branchv B i t) j u) = subst-branchv B i (subst-vv t j u)
and
atom j $\#$ C \implies (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 $((\text{atom } z) \iff (\text{atom } z')) \cdot c = c'$ **and** *atom z' $\#$ c*
shows $\llbracket z : b \mid c \rrbracket = \llbracket z' : b \mid c' \rrbracket$
using $\tau.\text{eq-iff Abs1-eq-iff}(3)$
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 $\#$ (s, sa)* **and** *atom c $\#$ (v',x, xa, s, sa)* *atom x $\#$ v'* **and** *atom xa $\#$ v'* **and** $(x \leftrightarrow c)$
 $\cdot s = (xa \leftrightarrow c) \cdot sa$
shows $s[x::=v]_{sv} = sa[xa::=v]_{sv}$

proof –

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

lemma *if-type-eq*:

fixes $\Gamma::\Gamma$ **and** $v::v$ **and** $z1::x$
assumes $\text{atom } z1' \# (v, ca, (x, b, c) \#_{\Gamma} \Gamma, (CE\text{-val } v == CE\text{-val } (V\text{-lit } ll) \text{ IMP } ca[za::=[z1]^v]_{cv}))$ **and** $\text{atom } z1 \# v$
and $\text{atom } z1 \# (za, ca)$ **and** $\text{atom } z1' \# (za, ca)$
shows $(\{z1' : ba \mid CE\text{-val } v == CE\text{-val } (V\text{-lit } ll) \text{ IMP } ca[za::=[z1]^v]_{cv}\}) = \{z1 : ba \mid CE\text{-val } v == CE\text{-val } (V\text{-lit } ll) \text{ IMP } ca[za::=[z1]^v]_{cv}\}$

proof –

have $\text{atom } z1' \# (CE\text{-val } v == CE\text{-val } (V\text{-lit } ll) \text{ IMP } ca[za::=[z1]^v]_{cv})$ **using** *assms* *fresh-prod4*
by *blast*
moreover hence $(CE\text{-val } v == CE\text{-val } (V\text{-lit } ll) \text{ IMP } ca[za::=[z1]^v]_{cv}) = (z1' \leftrightarrow z1) \cdot (CE\text{-val } v == CE\text{-val } (V\text{-lit } ll) \text{ IMP } ca[za::=[z1]^v]_{cv})$

proof –

have $(z1' \leftrightarrow z1) \cdot (CE\text{-val } v == CE\text{-val } (V\text{-lit } ll) \text{ IMP } ca[za::=[z1]^v]_{cv}) = ((z1' \leftrightarrow z1) \cdot (CE\text{-val } v == CE\text{-val } (V\text{-lit } ll) \text{ IMP } ((z1' \leftrightarrow z1) \cdot ca[za::=[z1]^v]_{cv})))$

by *auto*

also have $\dots = ((CE\text{-val } v == CE\text{-val } (V\text{-lit } ll)) \text{ IMP } ((z1' \leftrightarrow z1) \cdot ca[za::=[z1]^v]_{cv}))$

using $\langle \text{atom } z1 \# v \rangle \text{ assms}$

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 $\dots = ((CE\text{-val } v == CE\text{-val } (V\text{-lit } ll)) \text{ IMP } (ca[za::=[z1]^v]_{cv}))$

using *assms* **by** *fastforce*

finally show *?thesis* **by** *auto*

qed

ultimately show *?thesis*

using $\tau.\text{eq-iff}$ *Abs1-eq-iff(3)* [*of z1' CE-val v == CE-val (V-lit ll) IMP ca[za::=[z1]^v]_{cv}*
 $z1 \text{ CE-val } v == CE\text{-val } (V\text{-lit } ll) \text{ IMP } ca[za::=[z1]^v]_{cv}$] **by** *blast*

qed

lemma *subst-sv-var-flip*:

fixes $x::x$ **and** $s::s$ **and** $z::x$

shows $\text{atom } x \# s \implies ((x \leftrightarrow z) \cdot s) = s[z::=[x]^v]_{sv}$ **and**

$\text{atom } x \# cs \implies ((x \leftrightarrow z) \cdot cs) = \text{subst-branchv } cs \ z \ [x]^v$ **and**

```

      atom x # css  $\implies ((x \leftrightarrow z) \cdot \text{css}) = \text{subst-branchlv } \text{css } z [x]^v$ 
    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- $\tau$ -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'( $\beta$ ) flip-fresh-fresh
  apply (simp add: subst-v-c-def)
using x-fresh-u Abs1-eq-iff'( $\beta$ ) flip-fresh-fresh
  by (simp add: flip-fresh-fresh)

```

instantiation $s :: \text{has-subst-v}$
begin

definition

$\text{subst-v} = \text{subst-sv}$

instance proof

```

fix j::atom and i::x and x::v and t::s
show (j # subst-v t i x) = ((atom i # t  $\wedge$  j # t)  $\vee$  (j # x  $\wedge$  (j # t  $\vee$  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  $\implies$  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  $\cdot$  subst-v t1 x1 v = subst-v (p  $\cdot$  t1) (p  $\cdot$  x1) (p  $\cdot$  v)
  using subst-sv-commute subst-v-s-def by simp

```

```

fix x::x and c::s and z::x
show atom x # c  $\implies ((x \leftrightarrow z) \cdot c) = c[z::=[x]^v]_v$ 
  using subst-sv-var-flip subst-v-s-def by simp

```

```

fix x::x and c::s and z::x
show atom x # c  $\implies c[z::=[x]^v]_v[x::=v]_v = c[z::=v]_v$ 
  using subst-sv-var-flip subst-v-s-def by simp

```

qed
end

4.10 Type Definition

nominal-function $\text{subst-ft-v} :: \text{fun-typ} \Rightarrow x \Rightarrow v \Rightarrow \text{fun-typ}$ **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]tv
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 flip-commute subst-cv-var-flip)
  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-full
  v.fresh
  using 1 subst-cv-var-flip flip-commute by metis

next
case (2 z c t s za xa va ca ta sa cb)
  hence t[z::=[ cb ]v]tv = ta[za::=[ cb ]v]tv by metis
  hence t[z::=[ cb ]v]tv[xa::=va]tv = ta[za::=[ cb ]v]tv[xa::=va]tv by auto
  then show ?case using subst-tv-commute-full 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 $(j \# \text{subst-v } t \ i \ x) = ((\text{atom } i \ \# \ t \wedge j \ \# \ t) \vee (j \ \# \ x \wedge (j \ \# \ t \vee j = \text{atom } i)))$

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 $\text{atom } a \ \# \ tm \implies \text{subst-v } tm \ a \ x = tm$

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 $\text{subst-v } tm \ a \ (V\text{-var } a) = tm$

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 $p \cdot \text{subst-v } t1 \ x1 \ v = \text{subst-v } (p \cdot t1) \ (p \cdot x1) \ (p \cdot v)$

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 $\text{atom } x \ \# \ c \implies ((x \leftrightarrow z) \cdot c) = c[z::=[x]^v]_v$

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 $\text{atom } x \ \# \ c \implies c[z::=[x]^v]_v[x::=v]_v = c[z::=v]_v$

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* ⇒ *c*[*z::=[x]^v*]_{*v*}[*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* ($\Delta[x::=v]_{\Delta v}$) = *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 $(x',b',c') \in \text{toSet } \Gamma$ **and** $\text{atom } x \notin \text{atom-dom } \Gamma$
shows $(x',b',c'[x::=v]_{cv}) \in \text{toSet } \Gamma[x::=v]_{\Gamma v}$
proof –
have $x' \neq x$ **using** *assms fresh-dom-free2* **by** *metis*
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 $((x1, b1, c1) \#_{\Gamma} \Gamma')[x::=v]_{\Gamma v} = ((x1, b1, c1[x::=v]_{cv}) \#_{\Gamma} (\Gamma'[x::=v]_{\Gamma v}))$ **using** *subst-gv.simps*
 $\langle x' \neq x \rangle$ **by** *auto*
then show *?thesis* **using** *True* **by** *auto*
next
case *False*
have $x1 \neq x$ **using** *fresh-def fresh-GCons fresh-Pair supp-at-base GCons fresh-dom-free2* **by** *auto*
hence $(x', b', c') \in \text{toSet } \Gamma'$ **using** *GCons False toSet.simps* **by** *auto*
moreover have $\text{atom } x \notin \text{atom-dom } \Gamma'$ **using** *fresh-GCons GCons dom.simps toSet.simps* **by**
simp
ultimately have $(x', b', c'[x::=v]_{cv}) \in \text{toSet } \Gamma'[x::=v]_{\Gamma v}$ **using** *GCons* **by** *auto*
hence $(x', b', c'[x::=v]_{cv}) \in \text{toSet } ((x1, b1, c1[x::=v]_{cv}) \#_{\Gamma} (\Gamma'[x::=v]_{\Gamma v}))$ **by** *auto*
then show *?thesis* **using** *subst-gv.simps* $\langle x1 \neq x \rangle$ **by** *auto*
qed
qed
qed

lemma *fresh-subst-gv-if*:

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

4.12 Lookup

lemma *set-GConsD*: $y \in \text{toSet } (x \#_{\Gamma} xs) \implies y=x \vee y \in \text{toSet } xs$
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 $((x' \leftrightarrow x) \cdot tm)[bv::=b]_b = (x' \leftrightarrow x) \cdot (tm[bv::=b]_b)$

proof –

have $(x' \leftrightarrow x) \cdot bv = bv$ **using** *pure-supp flip-fresh-fresh* **by** *force*
moreover **have** $(x' \leftrightarrow x) \cdot b' = b'$ **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* :: $b \Rightarrow bv \Rightarrow b \Rightarrow b$ **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 :: $b \Rightarrow bv \Rightarrow b \Rightarrow b$ ($[-::=]_{bb}\ [1000, 50, 50]\ 1000$)

where

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

instantiation $b :: \text{has-subst-b}$

begin

definition $\text{subst-b} = \text{subst-bb}$

instance proof

fix $j::\text{atom}$ **and** $i::\text{bv}$ **and** $x::b$ **and** $t::b$

show $j \# \text{subst-b } t \ i \ x = (\text{atom } i \# t \wedge j \# t \vee j \# x \wedge (j \# t \vee j = \text{atom } i))$

proof (*induct t rule: b.induct*)

case ($B\text{-id } x$)

then show $?case$ **using** $\text{subst-bb.simps fresh-def pure-fresh subst-b-b-def}$ **by** *auto*

next

case ($B\text{-var } x$)

then show $?case$ **using** $\text{subst-bb.simps fresh-def pure-fresh subst-b-b-def}$ **by** *auto*

next

case ($B\text{-app } x1 \ x2$)

then show $?case$ **using** $\text{subst-bb.simps fresh-def pure-fresh subst-b-b-def}$ **by** *auto*

qed(*auto simp add: subst-bb.simps fresh-def pure-fresh subst-b-b-def*)+

fix $a::\text{bv}$ **and** $tm::b$ **and** $x::b$

show $\text{atom } a \# tm \implies tm[a::=x]_b = tm$

by (*induct tm rule: b.induct, auto simp add: fresh-at-base subst-bb.simps subst-b-b-def*)

fix $a::\text{bv}$ **and** $tm::b$

show $\text{subst-b } tm \ a \ (B\text{-var } a) = tm$ **using** $\text{subst-bb.simps subst-b-b-def}$

by (*induct tm rule: b.induct, auto simp add: fresh-at-base subst-bb.simps subst-b-b-def*)

fix $p::\text{perm}$ **and** $x1::\text{bv}$ **and** $v::b$ **and** $t1::b$

show $p \cdot \text{subst-b } t1 \ x1 \ v = \text{subst-b } (p \cdot t1) \ (p \cdot x1) \ (p \cdot v)$

by (*induct tm rule: b.induct, auto simp add: fresh-at-base subst-bb.simps subst-b-b-def*)

fix $bv::\text{bv}$ **and** $c::b$ **and** $z::bv$

show $\text{atom } bv \# c \implies ((bv \leftrightarrow z) \cdot c) = c[z::=B\text{-var } bv]_b$

by (*induct c rule: b.induct, (auto simp add: fresh-at-base subst-bb.simps subst-b-b-def permute-pure pure-suppl) +*)

fix $bv::\text{bv}$ **and** $c::b$ **and** $z::bv$ **and** $v::b$

show $\text{atom } bv \# c \implies ((bv \leftrightarrow z) \cdot c)[bv::=v]_b = c[z::=v]_b$

by (*induct c rule: b.induct, (auto simp add: fresh-at-base subst-bb.simps subst-b-b-def permute-pure pure-suppl) +*)

qed

end

lemma *subst-bb-inject*:

assumes $b1 = b2[bv::=b]_{bb}$ **and** $b2 \neq B\text{-var } bv$

shows

$b1 = B\text{-int} \implies b2 = B\text{-int}$ **and**

$b1 = B\text{-bool} \implies b2 = B\text{-bool}$ **and**

```

  b1 = B-unit  $\implies$  b2 = B-unit and
  b1 = B-bitvec  $\implies$  b2 = B-bitvec and
  b1 = B-pair b11 b12  $\implies$  ( $\exists$  b11' b12' . b11 = b11'[bv::=b]bb  $\wedge$  b12 = b12'[bv::=b]bb  $\wedge$  b2 = B-pair
b11' b12') and
  b1 = B-var bv'  $\implies$  b2 = B-var bv' and
  b1 = B-id tyid  $\implies$  b2 = B-id tyid and
  b1 = B-app tyid b11  $\implies$  ( $\exists$  b11' . b11 = b11'[bv::=b]bb  $\wedge$  b2 = B-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  $\wedge$  atom c  $\#$  x using fresh-def pure-supp by auto
  hence ((bv1  $\leftrightarrow$  c)  $\cdot$  B-id x) = B-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 $\text{atom } c \# b$ and $\text{atom } c \# (bv1, bv2, b1, b2)$ and $(bv1 \leftrightarrow c) \cdot b1 = (bv2 \leftrightarrow c) \cdot b2$

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* :: $v \Rightarrow bv \Rightarrow b \Rightarrow v$ **where**

subst-vb (V-lit l) $x\ v = V\text{-lit } l$

| *subst-vb* (V-var y) $x\ v = V\text{-var } y$

| *subst-vb* (V-cons $tyid\ c\ v'$) $x\ v = V\text{-cons } tyid\ c\ (subst-vb\ v'\ x\ v)$

| *subst-vb* (V-consp $tyid\ c\ b\ v'$) $x\ v = V\text{-consp } tyid\ c\ (subst-bb\ b\ x\ v)\ (subst-vb\ v'\ x\ v)$

| *subst-vb* (V-pair $v1\ v2$) $x\ v = V\text{-pair } (subst-vb\ v1\ x\ v)\ (subst-vb\ v2\ x\ v)$

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 :: $v \Rightarrow bv \Rightarrow b \Rightarrow v$ ($[-::=]_{vb}$ [1000,50,50] 500)

where

$e[bv::=b]_{vb} \equiv subst-vb\ e\ bv\ b$

instantiation $v :: has\text{-}subst\text{-}b$

begin

definition *subst-b* = *subst-vb*

instance proof

fix $j::atom$ and $i::bv$ and $x::b$ and $t::v$

show $j \# subst-b\ t\ i\ x = (atom\ i \# t \wedge j \# t \vee j \# x \wedge (j \# t \vee j = atom\ i))$

proof (*induct t rule: v.induct*)

case (V-lit l)

have $j \# subst-b\ (V\text{-lit } l)\ i\ x = j \# (V\text{-lit } l)$ **using** *subst-vb.simps* *fresh-def* *pure-fresh*

subst-b-v-def *v.suppl* *v.fresh* *has-subst-b-class.fresh-subst-if* *subst-b-b-def* *subst-b-v-def* **by** *auto*

also have $\dots = True$ **using** *fresh-at-base* *v.fresh* *l.fresh* *suppl-empty* *fresh-def* **by** *metis*

moreover have $(atom\ i \# (V\text{-lit } l) \wedge j \# (V\text{-lit } l) \vee j \# x \wedge (j \# (V\text{-lit } l) \vee j = atom\ i)) = True$

using *fresh-at-base* *v.fresh* *l.fresh* *suppl-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  $\sharp$  tm  $\implies$  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  $\cdot$  subst-b t1 x1 v = subst-b (p  $\cdot$  t1) (p  $\cdot$  x1) (p  $\cdot$  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  $\sharp$  c  $\implies$  ((bv  $\leftrightarrow$  z)  $\cdot$  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  $\sharp$  c  $\implies$  ((bv  $\leftrightarrow$  z)  $\cdot$  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* \Rightarrow *bv* \Rightarrow *b* \Rightarrow *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* \Rightarrow *bv* \Rightarrow *b* \Rightarrow *ce* ($[-::=]_{ceb}$ [1000,50,50] 500)

where

ce[*bv*::=*b*]_{*ceb*} \equiv *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* \wedge *j* $\#$ *t* \vee *j* $\#$ *x* \wedge (*j* $\#$ *t* \vee *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*} \wedge *j* $\#$ *v2*[*i*::=*x*]_{*ceb*}) = ((*atom* *i* $\#$ *v1* \wedge *atom* *i* $\#$ *v2*) \wedge *j* $\#$ *v1* \wedge *j* $\#$ *v2* \vee *j* $\#$ *x* \wedge (*j* $\#$ *v1* \wedge *j* $\#$ *v2* \vee *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  $\sharp$  tm  $\implies$  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  $\cdot$  subst-b t1 x1 v = subst-b (p  $\cdot$  t1) (p  $\cdot$  x1) (p  $\cdot$  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  $\sharp$  c  $\implies$  ((bv  $\leftrightarrow$  z)  $\cdot$  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  $\sharp$  c  $\implies$  ((bv  $\leftrightarrow$  z)  $\cdot$  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 ⇒ bv ⇒ b ⇒ 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 ⇒ bv ⇒ b ⇒ c (-[::=]_cb [1000,50,50] 500)
where
  c[bv::=b]_cb ≡ 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 # subst-b t i x = (atom i # t ∧ j # t ∨ j # x ∧ (j # t ∨ 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  $\tau$ .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  $\tau$ .eq-iff proof
  have *:  $[[atom\ z]]lst. c' = [[atom\ z]]lst. c$  using  $\tau$ .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  $(z' \leftrightarrow ca) \cdot c' = (z \leftrightarrow ca) \cdot c$  using 1 2 * Abs1-eq-iff-all(3) by auto
    hence  $((z' \leftrightarrow ca) \cdot c')[bv1' ::= b1]_{cb} = ((z \leftrightarrow ca) \cdot c)[bv1 ::= b1]_{cb}$  by auto
    hence  $(z' \leftrightarrow ca) \cdot c'[(z' \leftrightarrow ca) \cdot bv1' ::= (z' \leftrightarrow ca) \cdot b1]_{cb} = (z \leftrightarrow ca) \cdot c[(z \leftrightarrow ca) \cdot bv1' ::= (z \leftrightarrow ca) \cdot b1]_{cb}$  by auto
    thus  $(z' \leftrightarrow ca) \cdot c'[bv1' ::= b1]_{cb} = (z \leftrightarrow ca) \cdot c[bv1 ::= b1]_{cb}$  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

$subst\text{-}tb\text{-}abbrev :: \tau \Rightarrow bv \Rightarrow b \Rightarrow \tau \ (-[::=]_{\tau b} [1000, 50, 50] 1000)$

where

$t[bv ::= b]_{\tau b} \equiv subst\text{-}tb\ t\ bv\ b'$

instantiation $\tau :: has\text{-}subst\text{-}b$

begin

definition $subst\text{-}b = subst\text{-}tb$

instance proof

fix $j::atom$ and $i::bv$ and $x::b$ and $t::\tau$

show $j \# subst\text{-}b\ t\ i\ x = (atom\ i \# t \wedge j \# t \vee j \# x \wedge (j \# t \vee j = atom\ i))$

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 $tm::\tau$ and $x::b$

```

show  $atom\ a \# tm \implies subst\text{-}b\ tm\ a\ x = tm$ 
proof (nominal-induct  $tm$  avoiding:  $a\ x$  rule:  $\tau$ .strong-induct)
  case (T-refined-type  $xx\ bb\ cc$ )
  moreover hence  $atom\ a \# bb \wedge atom\ a \# cc$  using  $\tau$ .fresh by auto
  ultimately show ?case
    unfolding subst-b- $\tau$ -def subst-tb.simps
    using forget-subst subst-b-b-def subst-b-c-def forget-subst  $\tau$ .fresh by metis
qed

fix  $a::bv$  and  $tm::\tau$ 
show  $subst\text{-}b\ tm\ a\ (B\text{-}var\ a) = tm$ 
proof (nominal-induct  $tm$  rule:  $\tau$ .strong-induct)
  case (T-refined-type  $xx\ bb\ cc$ )
  thus ?case
    unfolding subst-b- $\tau$ -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  $t1::\tau$ 
show  $p \cdot subst\text{-}b\ t1\ x1\ v = subst\text{-}b\ (p \cdot t1)\ (p \cdot x1)\ (p \cdot v)$ 
by (induct  $tm$  rule:  $\tau$ .induct, auto simp add: fresh-at-base subst-tb.simps subst-b- $\tau$ -def subst-bb.simps subst-b-b-def)

fix  $bv::bv$  and  $c::\tau$  and  $z::bv$ 
show  $atom\ bv \# c \implies ((bv \leftrightarrow z) \cdot c) = c[z::=B\text{-}var\ bv]_b$ 
apply (induct  $c$  rule:  $\tau$ .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- $\tau$ -def)

fix  $bv::bv$  and  $c::\tau$  and  $z::bv$  and  $v::b$ 
show  $atom\ bv \# c \implies ((bv \leftrightarrow z) \cdot c)[bv::=v]_b = c[z::=v]_b$ 
proof (induct  $c$  rule:  $\tau$ .induct)
  case (T-refined-type  $x1a\ x2a\ x3a$ )
  hence  $atom\ bv \# x2a \wedge atom\ bv \# x3a \wedge atom\ bv \# x1a$  using fresh-at-base  $\tau$ .fresh by simp
  then show ?case
    unfolding subst-tb.simps subst-b- $\tau$ -def  $\tau$ .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  $\{ (bv \leftrightarrow z) \cdot x1a : ((bv \leftrightarrow z) \cdot x2a)[bv::=v]_{bb} \mid ((bv \leftrightarrow z) \cdot x3a)[bv::=v]_{cb} \} = \{ x1a$ 
 $: x2a[z::=v]_{bb} \mid x3a[z::=v]_{cb} \}$ 
  by (metis  $\ll atom\ bv \# x1a; atom\ z \# x1a \rrangle \implies (bv \leftrightarrow z) \cdot x1a = x1a \gg atom\ bv \# x2a \wedge atom\ bv$ 
 $\# x3a \wedge atom\ bv \# x1a$  flip-subst-subst subst-b-b-def subst-b-c-def)
  qed
qed

qed
end

```

```

lemma subst-bb-commute [simp]:
  atom j # A  $\implies$  (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  $\implies$  (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  $\implies$  (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  $\implies$  (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  $\implies$  (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:  $\tau$ .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  $\Rightarrow$  bv  $\Rightarrow$  b  $\Rightarrow$  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  $\Rightarrow$  bv  $\Rightarrow$  b  $\Rightarrow$  e (-[::=]eb [1000,50,50] 500)
where
  e[bv::=b]eb  $\equiv$  subst-eb e bv b

instantiation e :: has-subst-b
begin

```

definition $\text{subst-b} = \text{subst-eb}$

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-subst-if
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-subst-if
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 ($\lambda x. \text{Inl undefined}$) (case-sum ($\lambda x. \text{Inl undefined}$) ($\lambda x. \text{Inr unde-}$
 fined)))

subst-sb :: *s* ⇒ *bv* ⇒ *b* ⇒ *s*

and *subst-branchb* :: *branch-s* ⇒ *bv* ⇒ *b* ⇒ *branch-s*

and *subst-branchlb* :: *branch-list* ⇒ *bv* ⇒ *b* ⇒ *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*: $\bigwedge s \ x \ va . \ eqvt-at \ subst-sb-subst-branchb-subst-branchlb-sumC \ (Inl \ (s, \ x \ a, \ va)) \implies$
 $eqvt-at \ (\lambda a. \ projl \ (subst-sb-subst-branchb-subst-branchlb-sumC \ (Inl \ a))) \ (s, \ x \ a, \ va)$

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,x,a,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.proj1 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* $\#$ (*bva, ba*) \wedge *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 (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

$\text{subst-sb-abbrev} :: s \Rightarrow bv \Rightarrow b \Rightarrow s \text{ } (-[::=]_{sb} [1000, 50, 50] 1000)$

where

$b[bv::=b]_{sb} \equiv \text{subst-sb } b \text{ } bv \text{ } b'$

lemma *fresh-subst-sb-if* [simp]:

$(j \# (\text{subst-sb } A \text{ } i \text{ } x)) = ((\text{atom } i \# A \wedge j \# A) \vee (j \# x \wedge (j \# A \vee j = \text{atom } i)))$ and
 $(j \# (\text{subst-branchb } B \text{ } i \text{ } x)) = ((\text{atom } i \# B \wedge j \# B) \vee (j \# x \wedge (j \# B \vee j = \text{atom } i)))$ and
 $(j \# (\text{subst-branchlb } C \text{ } i \text{ } x)) = ((\text{atom } i \# C \wedge j \# C) \vee (j \# x \wedge (j \# C \vee j = \text{atom } i)))$

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 $(j \# \text{subst-branchb } (\text{AS-branch } x1 \text{ } x2 \text{ } x3) \text{ } i \text{ } x) = (j \# (\text{AS-branch } x1 \text{ } x2 \text{ } (\text{subst-sb } x3 \text{ } i \text{ } x)))$ by

auto

also have $\dots = ((j \# x3[i::=x]_{sb} \vee j \in \text{set } [\text{atom } x2]) \wedge j \# x1)$ using s-branch-s-branch-list.fresh by

auto

also have $\dots = ((\text{atom } i \# \text{AS-branch } x1 \text{ } x2 \text{ } x3 \wedge j \# \text{AS-branch } x1 \text{ } x2 \text{ } x3) \vee j \# x \wedge (j \# \text{AS-branch } x1 \text{ } x2 \text{ } x3 \vee j = \text{atom } i))$

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 $f1: \forall cs \text{ } b. \text{atom } (b::bv) \# (cs::\text{char list})$ using pure-fresh by auto

then have $j \# x \wedge \text{atom } i = j \longrightarrow ((j \# x3[i::=x]_{sb} \vee j \in \text{set } [\text{atom } x2]) \wedge j \# x1) = (\text{atom } i \# \text{AS-branch } x1 \text{ } x2 \text{ } x3 \wedge j \# \text{AS-branch } x1 \text{ } x2 \text{ } x3 \vee j \# x \wedge (j \# \text{AS-branch } x1 \text{ } x2 \text{ } x3 \vee j = \text{atom } i))$

by (metis (full-types) AS-branch.hyps(3))

then have $j \# x \longrightarrow ((j \# x3[i::=x]_{sb} \vee j \in \text{set } [\text{atom } x2]) \wedge j \# x1) = (\text{atom } i \# \text{AS-branch } x1 \text{ } x2 \text{ } x3 \wedge j \# \text{AS-branch } x1 \text{ } x2 \text{ } x3 \vee j \# x \wedge (j \# \text{AS-branch } x1 \text{ } x2 \text{ } x3 \vee j = \text{atom } i))$

using AS-branch.hyps s-branch-s-branch-list.fresh by metis

moreover

```

    { assume  $\neg j \# x$ 
      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- $\tau$ -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  $\wedge$  j # s  $\vee$  j # x  $\wedge$  (j # s  $\vee$  j = atom i))  $\vee$  j  $\in$  set [atom u])  $\wedge$  j # t[i::=x] $_{\tau b}$   $\wedge$  j
# v[i::=x] $_{vb}$ ) =
    (((atom i # s  $\wedge$  j # s  $\vee$  j # x  $\wedge$  (j # s  $\vee$  j = atom i))  $\vee$  j  $\in$  set [atom u])  $\wedge$ 
      ((atom i # t  $\wedge$  j # t  $\vee$  j # x  $\wedge$  (j # t  $\vee$  j = atom i)))  $\wedge$ 
      ((atom i # v  $\wedge$  j # v  $\vee$  j # x  $\wedge$  (j # v  $\vee$  j = atom i))))
    using has-subst-b-class.fresh-subst-if subst-b-v-def subst-b- $\tau$ -def by metis
  also have ... = (((atom i # s  $\vee$  atom i  $\in$  set [atom u])  $\wedge$  atom i # t  $\wedge$  atom i # v)  $\wedge$ 
    (j # s  $\vee$  j  $\in$  set [atom u])  $\wedge$  j # t  $\wedge$  j # v  $\vee$  j # x  $\wedge$  ((j # s  $\vee$  j  $\in$  set [atom u])  $\wedge$  j # t  $\wedge$  j
# v  $\vee$  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.premis s-branch-s-branch-list.fresh by simp
    then have atom a # x4
    by (metis (no-types) Nominal-Utills.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))
  end

```

```

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  $\nVdash$  s  $\implies$  ((bv  $\leftrightarrow$  z)  $\cdot$  s) = s[z::=B-var bv]sb and
        atom bv  $\nVdash$  cs  $\implies$  ((bv  $\leftrightarrow$  z)  $\cdot$  cs) = subst-branchb cs z (B-var bv) and
        atom bv  $\nVdash$  css  $\implies$  ((bv  $\leftrightarrow$  z)  $\cdot$  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  $\leftrightarrow$  z)  $\cdot$  x1) = x1 using pure-fresh fresh-at-base flip-fresh-fresh by metis
moreover have ((bv  $\leftrightarrow$  z)  $\cdot$  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  $\leftrightarrow$  z)  $\cdot$  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  $\leftrightarrow$  z)  $\cdot$  x2) = subst-branchb 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  $\leftrightarrow$  z)  $\cdot$  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  $\leftrightarrow$  z)  $\cdot$  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- $\tau$ -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- $\tau$ -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  $\implies$   $((bv \leftrightarrow z) \cdot s)[bv::=v]_{sb} = s[z::=v]_{sb}$  and
    atom bv  $\#$  cs  $\implies$  subst-branchb  $((bv \leftrightarrow z) \cdot cs)$  bv v = subst-branchb cs z v and
    atom bv  $\#$  css  $\implies$  subst-branchlb  $((bv \leftrightarrow z) \cdot css)$  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  $((bv \leftrightarrow z) \cdot x1) = x1$  using pure-fresh fresh-at-base flip-fresh-fresh by metis
  moreover have  $((bv \leftrightarrow z) \cdot x2) = x2$  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  $((bv \leftrightarrow z) \cdot x1) = x1$  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  $((bv \leftrightarrow z) \cdot x1) = x1$  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- $\tau$ -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 $s :: \text{has-subst-b}$

begin

definition $\text{subst-b} = (\lambda s \text{ bv } b. \text{subst-sb } s \text{ bv } b)$

instance proof

fix $j::\text{atom}$ **and** $i::\text{bv}$ **and** $x::b$ **and** $t::s$

show $j \# \text{subst-b } t \ i \ x = ((\text{atom } i \# t \wedge j \# t) \vee (j \# x \wedge (j \# t \vee j = \text{atom } i)))$

using fresh-subst-sb-if subst-b-s-def **by** metis

fix $a::\text{bv}$ **and** $tm::s$ **and** $x::b$

show $\text{atom } a \# tm \implies \text{subst-b } tm \ a \ x = tm$ **using** subst-b-s-def forget-subst-sb **by** metis

fix $a::\text{bv}$ **and** $tm::s$

show $\text{subst-b } tm \ a \ (B\text{-var } a) = tm$ **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.eqv
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]_τb
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- $\tau$ -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  $\implies$  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- $\tau$ -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  $\cdot$  subst-b t x1 v = subst-b (p  $\cdot$  t) (p  $\cdot$  x1) (p  $\cdot$  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- $\tau$ -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  $\implies$  ((bv  $\leftrightarrow$  z)  $\cdot$  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- $\tau$ -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  $\implies$  ((bv  $\leftrightarrow$  z)  $\cdot$  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- $\tau$ -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* :: $\Gamma \Rightarrow bv \Rightarrow b \Rightarrow \Gamma$ **where**

```

  subst-gb GNil - - = GNil
| subst-gb ((y,b',c)# $\Gamma$ ) bv b = ((y,b'[bv::=b]b,c[bv::=b]c)# $\Gamma$  (subst-gb  $\Gamma$  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  $\Gamma.exhaust\ neq\ GNil\ conv$  by force
qed
nominal-termination (eqvt) by lexicographic-order

```

abbreviation

```

subst-gb-abbrev ::  $\Gamma \Rightarrow bv \Rightarrow b \Rightarrow \Gamma \ (-[::=])_{\Gamma b} \ [1000, 50, 50] \ 1000$ 

```

where

```

g[bv::=b] $\Gamma b$   $\equiv$  subst-gb g bv b'

```

instantiation $\Gamma :: has\ subst\ b$

begin

definition *subst-b* = *subst-gb*

instance proof

```

fix j::atom and i::bv and x::b and t:: $\Gamma$ 

```

```

show  $j \# subst\ b\ t\ i\ x = (atom\ i \# t \wedge j \# t \vee j \# x \wedge (j \# t \vee j = atom\ i))$ 

```

```

proof(induct t rule:  $\Gamma$ -induct)

```

```

  case GNil

```

```

  then show ?case using fresh-GNil subst-gb.simps fresh-def pure-fresh subst-b- $\Gamma$ -def has-subst-b-class.fresh-subst-if
fresh-GNil fresh-GCons by metis

```

```

  next

```

```

    case (GCons x' b' c'  $\Gamma'$ )

```

```

    have *:  $atom\ i \# x'$  using fresh-at-base by simp

```

```

    have  $j \# subst\ b\ ((x', b', c') \#_{\Gamma} \Gamma')\ i\ x = j \# ((x', b'[i::=x]_{bb}, c'[i::=x]_{cb}) \#_{\Gamma} (subst\ b\ \Gamma'\ i\ x))$  using
subst-gb.simps subst-b- $\Gamma$ -def by auto

```

```

    also have ... =  $(j \# ((x', b'[i::=x]_{bb}, c'[i::=x]_{cb})) \wedge (j \# (subst\ b\ \Gamma'\ i\ x)))$  using fresh-GCons by
auto

```

```

    also have ... =  $((j \# x') \wedge (j \# b'[i::=x]_{bb}) \wedge (j \# c'[i::=x]_{cb})) \wedge (j \# (subst\ b\ \Gamma'\ i\ x))$  by auto

```

```

    also have ... =  $((j \# x') \wedge ((atom\ i \# b' \wedge j \# b' \vee j \# x \wedge (j \# b' \vee j = atom\ i))) \wedge$ 
 $((atom\ i \# c' \wedge j \# c' \vee j \# x \wedge (j \# c' \vee j = atom\ i))) \wedge$ 
 $((atom\ i \# \Gamma' \wedge j \# \Gamma' \vee j \# x \wedge (j \# \Gamma' \vee j = atom\ i))))$ 

```

```

    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 ... =  $((atom\ i \# (x', b', c') \#_{\Gamma} \Gamma' \wedge j \# (x', b', c') \#_{\Gamma} \Gamma') \vee (j \# x \wedge (j \# (x', b', c') \#_{\Gamma}$ 
 $\Gamma' \vee j = atom\ i)))$  using * fresh-GCons fresh-prod3 by metis

```

```

    finally show ?case by auto

```

qed

```

fix a::bv and tm:: $\Gamma$  and x::b

```

```

show  $atom\ a \# tm \implies subst\ b\ tm\ a\ x = tm$ 

```

```

proof (induct tm rule:  $\Gamma$ -induct)

```

```

  case GNil

```

```

  then show ?case using subst-gb.simps subst-b- $\Gamma$ -def by auto

```

```

  next

```

```

    case (GCons x' b' c'  $\Gamma'$ )

```

```

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

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

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

fix  $bv::bv$  and  $c::\Gamma$  and  $z::bv$  and  $v::b$ 
show  $atom \ bv \ \# \ c \implies ((bv \leftrightarrow z) \cdot c)[bv::=v]_b = c[z::=v]_b$ 
proof (induct c rule:  $\Gamma$ -induct)
  case GNil
  then show  $?case$  using subst-b- $\Gamma$ -def subst-gb.simps by auto
next
  case  $(GCons \ x \ b \ c \ \Gamma')$ 
  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- $\Gamma$ -def permute- $\Gamma$ .simps  $*$ 

```

```

    using GCons subst-b- $\Gamma$ -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  $\Gamma$  x
  shows Some (b[bv::=b]bb, c[bv::=b]cb) = lookup  $\Gamma$ [bv::=b] $\Gamma$ b x
  using assms by(induct  $\Gamma$  rule:  $\Gamma$ -induct, auto)

```

```

lemma subst-g-b-x-fresh:
  fixes x::x and b::b and  $\Gamma$ :: $\Gamma$  and bv::bv
  assumes atom x  $\#$   $\Gamma$ 
  shows atom x  $\#$   $\Gamma$ [bv::=b] $\Gamma$ b
  using subst-b-fresh-x subst-b- $\Gamma$ -def assms by metis

```

5.10.2 Mutable Variables

```

nominal-function subst-db ::  $\Delta \Rightarrow bv \Rightarrow b \Rightarrow \Delta$  where
  subst-db [] $\Delta$  - - = [] $\Delta$ 
| subst-db ((u,t)  $\#_{\Delta}$   $\Delta$ ) bv b = ((u,t[bv::=b] $\tau$ b)  $\#_{\Delta}$  (subst-db  $\Delta$  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 ::  $\Delta \Rightarrow bv \Rightarrow b \Rightarrow \Delta$  (-[::=] $\Delta$ b [1000,50,50] 1000)
where
   $\Delta$ [bv::=b] $\Delta$ b  $\equiv$  subst-db  $\Delta$  bv b

```

```

instantiation  $\Delta$  :: has-subst-b
begin
definition subst-b = subst-db

```

```

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

```

```

    also have ... = (j # ((u, t[i::=x]τb)) ∧ (j # (subst-b Γ' i x))) using fresh-DCons by auto
    also have ... = (((j # u) ∧ (j # t[i::=x]τb)) ∧ (j # (subst-b Γ' i x))) by auto
    also have ... = ((j # u) ∧ ((atom i # t ∧ j # t) ∨ (j # x ∧ (j # t ∨ j = atom i))) ∧ (atom i # Γ'
    ∧ j # Γ' ∨ j # x ∧ (j # Γ' ∨ 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 # (u, t) #Δ Γ' ∧ j # (u, t) #Δ Γ' ∨ j # x ∧ (j # (u, t) #Δ Γ' ∨ 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 # tm ⇒ subst-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]τb = 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-b ((u,t) #Δ Γ') a x = ((u,t[a::=x]τb) #Δ (subst-b Γ' a x)) using subst-b-Δ-def
  subst-db.simps by auto
  also have ... = ((u, t) #Δ Γ') 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 · subst-b t1 x1 v = subst-b (p · t1) (p · x1) (p · 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 # c ⇒ ((bv ↔ z) · c) = c[z::=B-var bv]b

```



```

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

```

```

fix bv::bv and c:: $\Delta$  and z::bv and v::b
show atom bv  $\nmid$  c  $\implies ((bv \leftrightarrow z) \cdot c)[bv::=v]_b = c[z::=v]_b$ 
proof (induct c rule:  $\Delta$ -induct)
  case DNil
  then show ?case using subst-b- $\Delta$ -def subst-db.simps by auto
next
  case (DCons u t')
  then show ?case
    unfolding subst-db.simps subst-b- $\Delta$ -def permute- $\Delta$ .simps
    using DCons subst-b- $\Delta$ -def subst-db.simps flip-subst subst-b- $\tau$ -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,  $\tau$ )  $\in$  setD  $\Delta$ 
  shows (u,  $\tau[bv::=b]_{\tau b}$ )  $\in$  setD  $\Delta[bv::=b]_{\Delta b}$ 
  using assms by (induct  $\Delta$ , auto)

```

```

lemmas ms-fresh-all = e.fresh s-branch-s-branch-list.fresh  $\tau$ .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  $\langle \text{Ctr-Sugar.ctr-sugar-of } @\{\text{context}\} @\{\text{type-name } b\} |> \text{Option.map } \# \text{ctr}\rangle$ 

```

```

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

```

```

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

```

shows $atom\ x \# v \implies atom\ x \# v[bv::=b]_{vb}$ **and**
 $atom\ x \# ce \implies atom\ x \# ce[bv::=b]_{ceb}$ **and**
 $atom\ x \# e \implies atom\ x \# e[bv::=b]_{eb}$ **and**
 $atom\ x \# c \implies atom\ x \# c[bv::=b]_{cb}$ **and**
 $atom\ x \# t \implies atom\ x \# t[bv::=b]_{\tau b}$ **and**
 $atom\ x \# d \implies atom\ x \# d[bv::=b]_{\Delta b}$ **and**
 $atom\ x \# g \implies atom\ x \# g[bv::=b]_{\Gamma b}$ **and**
 $atom\ x \# s \implies atom\ x \# s[bv::=b]_{sb}$
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 $tm::'a::has-subst-b$ **and** $x::u$
shows $atom\ x \# tm = atom\ x \# tm[bv::=b]_b$
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 $atom\ x \# \Gamma$
shows $atom\ x \# \Gamma[bv::=b]_{\Gamma b}$
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 $atom\ x \# \Gamma$
shows $atom\ x \# \Gamma[bv::=b]_{\Delta b}$
using *subst-b-fresh-u-cls* *subst-b- Δ -def* *assms* **by** *metis*

lemma *subst-b-fresh-u*:
fixes $x::u$
shows $atom\ x \# v \implies atom\ x \# v[bv::=b]_{vb}$ **and**
 $atom\ x \# ce \implies atom\ x \# ce[bv::=b]_{ceb}$ **and**
 $atom\ x \# e \implies atom\ x \# e[bv::=b]_{eb}$ **and**
 $atom\ x \# c \implies atom\ x \# c[bv::=b]_{cb}$ **and**
 $atom\ x \# t \implies atom\ x \# t[bv::=b]_{\tau b}$ **and**
 $atom\ x \# d \implies atom\ x \# d[bv::=b]_{\Delta b}$ **and**
 $atom\ x \# g \implies atom\ x \# g[bv::=b]_{\Gamma b}$ **and**
 $atom\ x \# s \implies atom\ x \# s[bv::=b]_{sb}$
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 $atom\ u \# D$
shows $atom\ u \# D[bv::=b]_{\Delta b}$
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:: τ* **and** *bv::bv*
assumes *atom bv $\#$ 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:: τ* **and** *t2:: τ*
assumes *atom bv $\#$ b* **and** *atom bv $\#$ (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 $wfV :: \Theta \Rightarrow \mathcal{B} \Rightarrow \Gamma \Rightarrow v \Rightarrow b \Rightarrow \text{bool} \ (- ; - ; - \vdash_{wf} - : - \ [50,50,50] \ 50)$ **and**
 $wfC :: \Theta \Rightarrow \mathcal{B} \Rightarrow \Gamma \Rightarrow c \Rightarrow \text{bool} \ (- ; - ; - \vdash_{wf} - \ [50,50] \ 50)$ **and**
 $wfG :: \Theta \Rightarrow \mathcal{B} \Rightarrow \Gamma \Rightarrow \text{bool} \ (- ; - \vdash_{wf} - \ [50,50] \ 50)$ **and**
 $wfT :: \Theta \Rightarrow \mathcal{B} \Rightarrow \Gamma \Rightarrow \tau \Rightarrow \text{bool} \ (- ; - ; - \vdash_{wf} - \ [50,50] \ 50)$ **and**
 $wfTs :: \Theta \Rightarrow \mathcal{B} \Rightarrow \Gamma \Rightarrow (\text{string} * \tau) \text{ list} \Rightarrow \text{bool} \ (- ; - ; - \vdash_{wf} - \ [50,50] \ 50)$ **and**
 $wfTh :: \Theta \Rightarrow \text{bool} \ (\vdash_{wf} - \ [50] \ 50)$ **and**
 $wfB :: \Theta \Rightarrow \mathcal{B} \Rightarrow b \Rightarrow \text{bool} \ (- ; - \vdash_{wf} - \ [50,50] \ 50)$ **and**
 $wfCE :: \Theta \Rightarrow \mathcal{B} \Rightarrow \Gamma \Rightarrow ce \Rightarrow b \Rightarrow \text{bool} \ (- ; - ; - \vdash_{wf} - : - \ [50,50,50] \ 50)$ **and**
 $wfTD :: \Theta \Rightarrow \text{type-def} \Rightarrow \text{bool} \ (- \vdash_{wf} - \ [50,50] \ 50)$
where

$wfB\text{-}intI: \vdash_{wf} \Theta \Longrightarrow \Theta; \mathcal{B} \vdash_{wf} B\text{-}int$
 $wfB\text{-}boolI: \vdash_{wf} \Theta \Longrightarrow \Theta; \mathcal{B} \vdash_{wf} B\text{-}bool$
 $wfB\text{-}unitI: \vdash_{wf} \Theta \Longrightarrow \Theta; \mathcal{B} \vdash_{wf} B\text{-}unit$
 $wfB\text{-}bitvecI: \vdash_{wf} \Theta \Longrightarrow \Theta; \mathcal{B} \vdash_{wf} B\text{-}bitvec$
 $wfB\text{-}pairI: \llbracket \Theta; \mathcal{B} \vdash_{wf} b1 ; \Theta; \mathcal{B} \vdash_{wf} b2 \rrbracket \Longrightarrow \Theta; \mathcal{B} \vdash_{wf} B\text{-}pair \ b1 \ b2$
 $wfB\text{-}consI: \llbracket$
 $\quad \vdash_{wf} \Theta;$
 $\quad (AF\text{-}typedef \ s \ dclist) \in \text{set } \Theta$
 $\rrbracket \Longrightarrow$
 $\quad \Theta; \mathcal{B} \vdash_{wf} B\text{-}id \ s$
 $wfB\text{-}appI: \llbracket$
 $\quad \vdash_{wf} \Theta;$
 $\quad \Theta; \mathcal{B} \vdash_{wf} b;$

$$\begin{array}{l}
(AF\text{-typedef-poly } s \text{ bv } dclist) \in \text{set } \Theta \\
\] \Longrightarrow \\
\quad \Theta; \mathcal{B} \vdash_{wf} B\text{-app } s \text{ } b \\
\\
| \text{ wfV-varI: } \llbracket \Theta; \mathcal{B} \vdash_{wf} \Gamma; \text{Some } (b,c) = \text{lookup } \Gamma \text{ } x \rrbracket \Longrightarrow \Theta; \mathcal{B}; \Gamma \vdash_{wf} V\text{-var } x : b \\
| \text{ wfV-litI: } \Theta; \mathcal{B} \vdash_{wf} \Gamma \Longrightarrow \Theta; \mathcal{B}; \Gamma \vdash_{wf} V\text{-lit } l : \text{base-for-lit } l \\
\\
| \text{ wfV-pairI: } \llbracket \\
\quad \Theta; \mathcal{B}; \Gamma \vdash_{wf} v1 : b1 ; \\
\quad \Theta; \mathcal{B}; \Gamma \vdash_{wf} v2 : b2 \\
\] \Longrightarrow \\
\quad \Theta; \mathcal{B}; \Gamma \vdash_{wf} (V\text{-pair } v1 \text{ } v2) : B\text{-pair } b1 \text{ } b2 \\
\\
| \text{ wfV-consI: } \llbracket \\
\quad AF\text{-typedef } s \text{ } dclist \in \text{set } \Theta; \\
\quad (dc, \llbracket x : b' \mid c \rrbracket) \in \text{set } dclist ; \\
\quad \Theta; \mathcal{B}; \Gamma \vdash_{wf} v : b' \\
\] \Longrightarrow \\
\quad \Theta; \mathcal{B}; \Gamma \vdash_{wf} V\text{-cons } s \text{ } dc \text{ } v : B\text{-id } s \\
\\
| \text{ wfV-conspI: } \llbracket \\
\quad AF\text{-typedef-poly } s \text{ bv } dclist \in \text{set } \Theta; \\
\quad (dc, \llbracket x : b' \mid c \rrbracket) \in \text{set } dclist ; \\
\quad \Theta ; \mathcal{B} \vdash_{wf} b; \\
\quad \text{atom } bv \nmid (\Theta, \mathcal{B}, \Gamma, b, v); \\
\quad \Theta; \mathcal{B}; \Gamma \vdash_{wf} v : b'[bv::=b]_{bb} \\
\] \Longrightarrow \\
\quad \Theta; \mathcal{B}; \Gamma \vdash_{wf} V\text{-consp } s \text{ } dc \text{ } b \text{ } v : B\text{-app } s \text{ } b \\
\\
| \text{ wfCE-valI : } \llbracket \\
\quad \Theta; \mathcal{B}; \Gamma \vdash_{wf} v : b \\
\] \Longrightarrow \\
\quad \Theta; \mathcal{B}; \Gamma \vdash_{wf} CE\text{-val } v : b \\
\\
| \text{ wfCE-plusI: } \llbracket \\
\quad \Theta; \mathcal{B}; \Gamma \vdash_{wf} v1 : B\text{-int}; \\
\quad \Theta; \mathcal{B}; \Gamma \vdash_{wf} v2 : B\text{-int} \\
\] \Longrightarrow \\
\quad \Theta; \mathcal{B}; \Gamma \vdash_{wf} CE\text{-op Plus } v1 \text{ } v2 : B\text{-int} \\
\\
| \text{ wfCE-leqI: } \llbracket \\
\quad \Theta; \mathcal{B}; \Gamma \vdash_{wf} v1 : B\text{-int}; \\
\quad \Theta; \mathcal{B}; \Gamma \vdash_{wf} v2 : B\text{-int} \\
\] \Longrightarrow \\
\quad \Theta; \mathcal{B}; \Gamma \vdash_{wf} CE\text{-op LEq } v1 \text{ } v2 : B\text{-bool} \\
\\
| \text{ wfCE-fstI: } \llbracket \\
\quad \Theta; \mathcal{B}; \Gamma \vdash_{wf} v1 : B\text{-pair } b1 \text{ } b2 \\
\] \Longrightarrow \\
\quad \Theta; \mathcal{B}; \Gamma \vdash_{wf} CE\text{-fst } v1 : b1
\end{array}$$

$| \text{wfCE-sndI}: \llbracket$
 $\quad \Theta; \mathcal{B}; \Gamma \vdash_{wf} v1 : B\text{-pair } b1 \ b2$
 $\rrbracket \implies$
 $\quad \Theta; \mathcal{B}; \Gamma \vdash_{wf} CE\text{-snd } v1 : b2$

$| \text{wfCE-concatI}: \llbracket$
 $\quad \Theta; \mathcal{B}; \Gamma \vdash_{wf} v1 : B\text{-bitvec} ;$
 $\quad \Theta; \mathcal{B}; \Gamma \vdash_{wf} v2 : B\text{-bitvec}$
 $\rrbracket \implies$
 $\quad \Theta; \mathcal{B}; \Gamma \vdash_{wf} CE\text{-concat } v1 \ v2 : B\text{-bitvec}$

$| \text{wfCE-lenI}: \llbracket$
 $\quad \Theta; \mathcal{B}; \Gamma \vdash_{wf} v1 : B\text{-bitvec}$
 $\rrbracket \implies$
 $\quad \Theta; \mathcal{B}; \Gamma \vdash_{wf} CE\text{-len } v1 : B\text{-int}$

$| \text{wfTI} : \llbracket$
 $\quad atom \ z \ \# \ (\Theta, \mathcal{B}, \Gamma) ;$
 $\quad \Theta; \mathcal{B} \vdash_{wf} b ;$
 $\quad \Theta; \mathcal{B} ; (z, b, C\text{-true}) \#_{\Gamma} \Gamma \vdash_{wf} c$
 $\rrbracket \implies$
 $\quad \Theta; \mathcal{B}; \Gamma \vdash_{wf} \{ z : b \mid c \}$

$| \text{wfC-eqI}: \llbracket$
 $\quad \Theta; \mathcal{B}; \Gamma \vdash_{wf} e1 : b ;$
 $\quad \Theta; \mathcal{B}; \Gamma \vdash_{wf} e2 : b \rrbracket \implies$
 $\quad \Theta; \mathcal{B}; \Gamma \vdash_{wf} C\text{-eq } e1 \ e2$

$| \text{wfC-trueI}: \Theta; \mathcal{B} \vdash_{wf} \Gamma \implies \Theta; \mathcal{B}; \Gamma \vdash_{wf} C\text{-true}$

$| \text{wfC-falseI}: \Theta; \mathcal{B} \vdash_{wf} \Gamma \implies \Theta; \mathcal{B}; \Gamma \vdash_{wf} C\text{-false}$

$| \text{wfC-conjI}: \llbracket \Theta; \mathcal{B}; \Gamma \vdash_{wf} c1 ; \Theta; \mathcal{B}; \Gamma \vdash_{wf} c2 \rrbracket \implies \Theta; \mathcal{B}; \Gamma \vdash_{wf} C\text{-conj } c1 \ c2$

$| \text{wfC-disjI}: \llbracket \Theta; \mathcal{B}; \Gamma \vdash_{wf} c1 ; \Theta; \mathcal{B}; \Gamma \vdash_{wf} c2 \rrbracket \implies \Theta; \mathcal{B}; \Gamma \vdash_{wf} C\text{-disj } c1 \ c2$

$| \text{wfC-notI}: \llbracket \Theta; \mathcal{B}; \Gamma \vdash_{wf} c1 \rrbracket \implies \Theta; \mathcal{B}; \Gamma \vdash_{wf} C\text{-not } c1$

$| \text{wfC-impI}: \llbracket \Theta; \mathcal{B}; \Gamma \vdash_{wf} c1 ;$
 $\quad \Theta; \mathcal{B}; \Gamma \vdash_{wf} c2 \rrbracket \implies \Theta; \mathcal{B}; \Gamma \vdash_{wf} C\text{-imp } c1 \ c2$

$| \text{wfG-nilI}: \vdash_{wf} \Theta \implies \Theta; \mathcal{B} \vdash_{wf} GNil$

$| \text{wfG-cons1I}: \llbracket c \notin \{ TRUE, FALSE \} ;$
 $\quad \Theta; \mathcal{B} \vdash_{wf} \Gamma ;$
 $\quad atom \ x \ \# \ \Gamma ;$
 $\quad \Theta ; \mathcal{B} ; (x, b, C\text{-true}) \#_{\Gamma} \Gamma \vdash_{wf} c ; wfB \ \Theta \ \mathcal{B} \ b$
 $\rrbracket \implies \Theta; \mathcal{B} \vdash_{wf} ((x, b, c) \#_{\Gamma} \Gamma)$

$| \text{wfG-cons2I}: \llbracket c \in \{ TRUE, FALSE \} ;$
 $\quad \Theta; \mathcal{B} \vdash_{wf} \Gamma ;$
 $\quad atom \ x \ \# \ \Gamma ;$
 $\quad wfB \ \Theta \ \mathcal{B} \ b$
 $\rrbracket \implies \Theta; \mathcal{B} \vdash_{wf} ((x, b, c) \#_{\Gamma} \Gamma)$

$| \text{wfTh-emptyI}: \vdash_{wf} \llbracket$

$| \text{wfTh-consI: } \llbracket$
 $(\text{name-of-type } tdef) \notin \text{name-of-type 'set } \Theta ;$
 $\vdash_{wf} \Theta ;$
 $\Theta \vdash_{wf} tdef \rrbracket \implies \vdash_{wf} tdef \# \Theta$

$| \text{wfTD-simpleI: } \llbracket$
 $\Theta ; \{||\} ; GNil \vdash_{wf} lst$
 $\rrbracket \implies$
 $\Theta \vdash_{wf} (AF\text{-typedef } s \text{ } lst)$

$| \text{wfTD-poly: } \llbracket$
 $\Theta ; \{|bv|\} ; GNil \vdash_{wf} lst$
 $\rrbracket \implies$
 $\Theta \vdash_{wf} (AF\text{-typedef-poly } s \text{ } bv \text{ } lst)$

$| \text{wfTs-nil: } \Theta ; \mathcal{B} \vdash_{wf} \Gamma \implies \Theta ; \mathcal{B} ; \Gamma \vdash_{wf} [] :: (\text{string} * \tau) \text{ list}$
 $| \text{wfTs-cons: } \llbracket \Theta ; \mathcal{B} ; \Gamma \vdash_{wf} \tau ;$
 $dc \notin \text{fst 'set } ts ;$
 $\Theta ; \mathcal{B} ; \Gamma \vdash_{wf} ts :: (\text{string} * \tau) \text{ list } \rrbracket \implies \Theta ; \mathcal{B} ; \Gamma \vdash_{wf} ((dc, \tau) \# ts)$

inductive-cases *wfC-elim*:

$\Theta ; \mathcal{B} ; \Gamma \vdash_{wf} C\text{-true}$
 $\Theta ; \mathcal{B} ; \Gamma \vdash_{wf} C\text{-false}$
 $\Theta ; \mathcal{B} ; \Gamma \vdash_{wf} C\text{-eq } e1 \text{ } e2$
 $\Theta ; \mathcal{B} ; \Gamma \vdash_{wf} C\text{-conj } c1 \text{ } c2$
 $\Theta ; \mathcal{B} ; \Gamma \vdash_{wf} C\text{-disj } c1 \text{ } c2$
 $\Theta ; \mathcal{B} ; \Gamma \vdash_{wf} C\text{-not } c1$
 $\Theta ; \mathcal{B} ; \Gamma \vdash_{wf} C\text{-imp } c1 \text{ } c2$

inductive-cases *wfV-elim*:

$\Theta ; \mathcal{B} ; \Gamma \vdash_{wf} V\text{-var } x : b$
 $\Theta ; \mathcal{B} ; \Gamma \vdash_{wf} V\text{-lit } l : b$
 $\Theta ; \mathcal{B} ; \Gamma \vdash_{wf} V\text{-pair } v1 \text{ } v2 : b$
 $\Theta ; \mathcal{B} ; \Gamma \vdash_{wf} V\text{-cons } tyid \text{ } dc \text{ } v : b$
 $\Theta ; \mathcal{B} ; \Gamma \vdash_{wf} V\text{-consp } tyid \text{ } dc \text{ } b \text{ } v : b'$

inductive-cases *wfCE-elim*:

$\Theta ; \mathcal{B} ; \Gamma \vdash_{wf} CE\text{-val } v : b$
 $\Theta ; \mathcal{B} ; \Gamma \vdash_{wf} CE\text{-op Plus } v1 \text{ } v2 : b$
 $\Theta ; \mathcal{B} ; \Gamma \vdash_{wf} CE\text{-op LEq } v1 \text{ } v2 : b$
 $\Theta ; \mathcal{B} ; \Gamma \vdash_{wf} CE\text{-fst } v1 : b$
 $\Theta ; \mathcal{B} ; \Gamma \vdash_{wf} CE\text{-snd } v1 : b$
 $\Theta ; \mathcal{B} ; \Gamma \vdash_{wf} CE\text{-concat } v1 \text{ } v2 : b$
 $\Theta ; \mathcal{B} ; \Gamma \vdash_{wf} CE\text{-len } v1 : b$
 $\Theta ; \mathcal{B} ; \Gamma \vdash_{wf} CE\text{-op opp } v1 \text{ } v2 : b$

inductive-cases *wfT-elim*:

$\Theta ; \mathcal{B} ; \Gamma \vdash_{wf} \tau :: \tau$
 $\Theta ; \mathcal{B} ; \Gamma \vdash_{wf} \llbracket z : b \mid c \rrbracket$

inductive-cases *wfG-elim*:

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

inductive-cases *wfTh-elim*:

$\vdash_{wf} []$
 $\vdash_{wf} td \# \Pi$

inductive-cases *wfTD-elim*:

$\Theta \vdash_{wf} (AF\text{-typedef } s \text{ } lst)$
 $\Theta \vdash_{wf} (AF\text{-typedef-poly } s \text{ } bv \text{ } lst)$

inductive-cases *wfTs-elim*:

$\Theta; \mathcal{B} ; GNil \vdash_{wf} ([::((string*\tau) \text{ } list))$
 $\Theta; \mathcal{B} ; GNil \vdash_{wf} ((t\#ts)::((string*\tau) \text{ } list))$

inductive-cases *wfB-elim*:

$\Theta; \mathcal{B} \vdash_{wf} B\text{-pair } b1 \text{ } b2$
 $\Theta; \mathcal{B} \vdash_{wf} B\text{-id } s$
 $\Theta; \mathcal{B} \vdash_{wf} B\text{-app } s \text{ } b$

equivariance *wfV*

This is by no means complete as we have for some of lemmas like weakening done it the hard way

nominal-inductive *wfV*

avoids *wfV-conspI*: *bv* | *wfTI*: *z*

proof(*goal-cases*)

case (*1 s bv dclist* $\Theta \text{ } dc \text{ } x \text{ } b' \text{ } c \text{ } \mathcal{B} \text{ } b \text{ } \Gamma \text{ } v$)

moreover hence *atom bv* $\nVdash V\text{-consp } s \text{ } dc \text{ } b \text{ } v$ **using** *v.fresh fresh-prodN pure-fresh* **by** *metis*

moreover have *atom bv* $\nVdash B\text{-app } s \text{ } b$ **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* $\Theta \text{ } dc \text{ } x \text{ } b' \text{ } c \text{ } \mathcal{B} \text{ } b \text{ } \Gamma \text{ } v$)

then show *?case* **by** *auto*

next

case (*3 z* $\Gamma \text{ } \Theta \text{ } \mathcal{B} \text{ } b \text{ } c$)

then show *?case* **using** *τ .fresh fresh-star-def fresh-prodN* **by** *fastforce*

next

case (*4 z* $\Gamma \text{ } \Theta \text{ } \mathcal{B} \text{ } b \text{ } c$)

then show *?case* **by** *auto*

qed

inductive

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

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

$wfE\text{-valI} : \llbracket ($
 $\Theta \vdash_{wf} \Phi) ;$
 $\Theta ; \mathcal{B} ; \Gamma \vdash_{wf} \Delta ;$
 $\Theta ; \mathcal{B} ; \Gamma \vdash_{wf} v : b$
 $\rrbracket \Rightarrow$
 $\Theta ; \Phi ; \mathcal{B} ; \Gamma ; \Delta \vdash_{wf} AE\text{-val } v : b$

$| \text{wfE-plusI} : \llbracket$
 $\Theta \vdash_{wf} \Phi ;$
 $\Theta ; \mathcal{B} ; \Gamma \vdash_{wf} \Delta ;$
 $\Theta ; \mathcal{B} ; \Gamma \vdash_{wf} v1 : B\text{-int} ;$
 $\Theta ; \mathcal{B} ; \Gamma \vdash_{wf} v2 : B\text{-int}$
 $\rrbracket \Rightarrow$
 $\Theta ; \Phi ; \mathcal{B} ; \Gamma ; \Delta \vdash_{wf} AE\text{-op Plus } v1 \ v2 : B\text{-int}$

$| \text{wfE-leqI} : \llbracket$
 $\Theta \vdash_{wf} \Phi ;$
 $\Theta ; \mathcal{B} ; \Gamma \vdash_{wf} \Delta ;$
 $\Theta ; \mathcal{B} ; \Gamma \vdash_{wf} v1 : B\text{-int} ;$
 $\Theta ; \mathcal{B} ; \Gamma \vdash_{wf} v2 : B\text{-int}$
 $\rrbracket \Rightarrow$
 $\Theta ; \Phi ; \mathcal{B} ; \Gamma ; \Delta \vdash_{wf} AE\text{-op LEq } v1 \ v2 : B\text{-bool}$

$| \text{wfE-fstI} : \llbracket$
 $\Theta \vdash_{wf} \Phi ;$
 $\Theta ; \mathcal{B} ; \Gamma \vdash_{wf} \Delta ;$
 $\Theta ; \mathcal{B} ; \Gamma \vdash_{wf} v1 : B\text{-pair } b1 \ b2$
 $\rrbracket \Rightarrow$
 $\Theta ; \Phi ; \mathcal{B} ; \Gamma ; \Delta \vdash_{wf} AE\text{-fst } v1 : b1$

$| \text{wfE-sndI} : \llbracket$
 $\Theta \vdash_{wf} \Phi ;$
 $\Theta ; \mathcal{B} ; \Gamma \vdash_{wf} \Delta ;$
 $\Theta ; \mathcal{B} ; \Gamma \vdash_{wf} v1 : B\text{-pair } b1 \ b2$
 $\rrbracket \Rightarrow$
 $\Theta ; \Phi ; \mathcal{B} ; \Gamma ; \Delta \vdash_{wf} AE\text{-snd } v1 : b2$

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

$\mathbb{I} \Rightarrow$
 $\Theta ; \Phi ; \mathcal{B} ; \Gamma ; \Delta \vdash_{wf} AE\text{-concat } v1 \ v2 : B\text{-bitvec}$

$| \text{ } wfE\text{-splitI}: \mathbb{I}$
 $\Theta \vdash_{wf} \Phi ;$
 $\Theta ; \mathcal{B} ; \Gamma \vdash_{wf} \Delta ;$
 $\Theta ; \mathcal{B} ; \Gamma \vdash_{wf} v1 : B\text{-bitvec};$
 $\Theta ; \mathcal{B} ; \Gamma \vdash_{wf} v2 : B\text{-int}$
 $\mathbb{I} \Rightarrow$
 $\Theta ; \Phi ; \mathcal{B} ; \Gamma ; \Delta \vdash_{wf} AE\text{-split } v1 \ v2 : B\text{-pair } B\text{-bitvec } B\text{-bitvec}$

$| \text{ } wfE\text{-lenI}: \mathbb{I}$
 $\Theta \vdash_{wf} \Phi ;$
 $\Theta ; \mathcal{B} ; \Gamma \vdash_{wf} \Delta ;$
 $\Theta ; \mathcal{B} ; \Gamma \vdash_{wf} v1 : B\text{-bitvec}$
 $\mathbb{I} \Rightarrow$
 $\Theta ; \Phi ; \mathcal{B} ; \Gamma ; \Delta \vdash_{wf} AE\text{-len } v1 : B\text{-int}$

$| \text{ } wfE\text{-appI}: \mathbb{I}$
 $\Theta \vdash_{wf} \Phi ;$
 $\Theta ; \mathcal{B} ; \Gamma \vdash_{wf} \Delta ;$
 $Some (AF\text{-fundef } f (AF\text{-fun-typ-none } (AF\text{-fun-typ } x \ b \ c \ \tau \ s))) = lookup\text{-fun } \Phi \ f ;$
 $\Theta ; \mathcal{B} ; \Gamma \vdash_{wf} v : b$
 $\mathbb{I} \Rightarrow$
 $\Theta ; \Phi ; \mathcal{B} ; \Gamma ; \Delta \vdash_{wf} AE\text{-app } f \ v : b\text{-of } \tau$

$| \text{ } wfE\text{-appPI}: \mathbb{I}$
 $\Theta \vdash_{wf} \Phi ;$
 $\Theta ; \mathcal{B} ; \Gamma \vdash_{wf} \Delta ;$
 $\Theta ; \mathcal{B} \vdash_{wf} b' ;$
 $atom \ bv \ \sharp (\Phi, \Theta, \mathcal{B}, \Gamma, \Delta, b', v, (b\text{-of } \tau)[bv ::= b]_b);$
 $Some (AF\text{-fundef } f (AF\text{-fun-typ-some } bv (AF\text{-fun-typ } x \ b \ c \ \tau \ s))) = lookup\text{-fun } \Phi \ f ;$
 $\Theta ; \mathcal{B} ; \Gamma \vdash_{wf} v : (b[bv ::= b]_b)$
 $\mathbb{I} \Rightarrow$
 $\Theta ; \Phi ; \mathcal{B} ; \Gamma ; \Delta \vdash_{wf} (AE\text{-appP } f \ b' \ v) : ((b\text{-of } \tau)[bv ::= b]_b)$

$| \text{ } wfE\text{-mvarI}: \mathbb{I}$
 $\Theta \vdash_{wf} \Phi ;$
 $\Theta ; \mathcal{B} ; \Gamma \vdash_{wf} \Delta ;$
 $(u, \tau) \in setD \ \Delta$
 $\mathbb{I} \Rightarrow$
 $\Theta ; \Phi ; \mathcal{B} ; \Gamma ; \Delta \vdash_{wf} AE\text{-mvar } u : b\text{-of } \tau$

$| \text{ } wfS\text{-valI}: \mathbb{I}$
 $\Theta \vdash_{wf} \Phi ;$
 $\Theta ; \mathcal{B} ; \Gamma \vdash_{wf} v : b ;$
 $\Theta ; \mathcal{B} ; \Gamma \vdash_{wf} \Delta$
 $\mathbb{I} \Rightarrow$
 $\Theta ; \Phi ; \mathcal{B} ; \Gamma ; \Delta \vdash_{wf} (AS\text{-val } v) : b$

$| \text{ } wfS\text{-letI}: \mathbb{I}$
 $wfE \ \Theta \ \Phi \ \mathcal{B} \ \Gamma \ \Delta \ e \ b' ;$

$$\begin{array}{l}
\Theta ; \Phi ; \mathcal{B} ; (x, b', C\text{-true}) \#_{\Gamma} \Gamma ; \Delta \vdash_{wf} s : b ; \\
\Theta ; \mathcal{B} ; \Gamma \vdash_{wf} \Delta ; \\
atom\ x \# (\Phi, \Theta, \mathcal{B}, \Gamma, \Delta, e, b) \\
\parallel \implies \\
\Theta ; \Phi ; \mathcal{B} ; \Gamma ; \Delta \vdash_{wf} LET\ x = e\ IN\ s : b \\
\\
| \text{ wfS-assertI: } \parallel \\
\Theta ; \Phi ; \mathcal{B} ; (x, B\text{-bool}, c) \#_{\Gamma} \Gamma ; \Delta \vdash_{wf} s : b ; \\
\Theta ; \mathcal{B} ; \Gamma \vdash_{wf} c ; \\
\Theta ; \mathcal{B} ; \Gamma \vdash_{wf} \Delta ; \\
atom\ x \# (\Phi, \Theta, \mathcal{B}, \Gamma, \Delta, c, b, s) \\
\parallel \implies \\
\Theta ; \Phi ; \mathcal{B} ; \Gamma ; \Delta \vdash_{wf} ASSERT\ c\ IN\ s : b \\
\\
| \text{ wfS-let2I: } \parallel \Theta ; \Phi ; \mathcal{B} ; \Gamma ; \Delta \vdash_{wf} s1 : b\text{-of } \tau ; \\
\Theta ; \mathcal{B} ; \Gamma \vdash_{wf} \tau ; \\
\Theta ; \Phi ; \mathcal{B} ; (x, b\text{-of } \tau, C\text{-true}) \#_{\Gamma} \Gamma ; \Delta \vdash_{wf} s2 : b ; \\
atom\ x \# (\Phi, \Theta, \mathcal{B}, \Gamma, \Delta, s1, b, \tau) \\
\parallel \implies \\
\Theta ; \Phi ; \mathcal{B} ; \Gamma ; \Delta \vdash_{wf} LET\ x : \tau = s1\ IN\ s2 : b \\
| \text{ wfS-ifI: } \parallel \Theta ; \mathcal{B} ; \Gamma \vdash_{wf} v : B\text{-bool} ; \\
\Theta ; \Phi ; \mathcal{B} ; \Gamma ; \Delta \vdash_{wf} s1 : b ; \\
\Theta ; \Phi ; \mathcal{B} ; \Gamma ; \Delta \vdash_{wf} s2 : b \parallel \implies \\
\Theta ; \Phi ; \mathcal{B} ; \Gamma ; \Delta \vdash_{wf} IF\ v\ THEN\ s1\ ELSE\ s2 : b \\
\\
| \text{ wfS-varI : } \parallel \text{ wfT } \Theta\ \mathcal{B}\ \Gamma\ \tau ; \\
\Theta ; \mathcal{B} ; \Gamma \vdash_{wf} v : b\text{-of } \tau ; \\
atom\ u \# (\Phi, \Theta, \mathcal{B}, \Gamma, \Delta, \tau, v, b) ; \\
\Theta ; \Phi ; \mathcal{B} ; \Gamma ; (u, \tau) \#_{\Delta} \Delta \vdash_{wf} s : b \parallel \implies \\
\Theta ; \Phi ; \mathcal{B} ; \Gamma ; \Delta \vdash_{wf} VAR\ u : \tau = v\ IN\ s : b \\
\\
| \text{ wfS-assignI: } \parallel (u, \tau) \in setD\ \Delta ; \\
\Theta ; \mathcal{B} ; \Gamma \vdash_{wf} \Delta ; \\
\Theta \vdash_{wf} \Phi ; \\
\Theta ; \mathcal{B} ; \Gamma \vdash_{wf} v : b\text{-of } \tau \parallel \implies \\
\Theta ; \Phi ; \mathcal{B} ; \Gamma ; \Delta \vdash_{wf} u ::= v : B\text{-unit} \\
\\
| \text{ wfS-whileI: } \parallel \Theta ; \Phi ; \mathcal{B} ; \Gamma ; \Delta \vdash_{wf} s1 : B\text{-bool} ; \\
\Theta ; \Phi ; \mathcal{B} ; \Gamma ; \Delta \vdash_{wf} s2 : b \parallel \implies \\
\Theta ; \Phi ; \mathcal{B} ; \Gamma ; \Delta \vdash_{wf} WHILE\ s1\ DO\ \{ s2 \} : b \\
\\
| \text{ wfS-seqI: } \parallel \Theta ; \Phi ; \mathcal{B} ; \Gamma ; \Delta \vdash_{wf} s1 : B\text{-unit} ; \\
\Theta ; \Phi ; \mathcal{B} ; \Gamma ; \Delta \vdash_{wf} s2 : b \parallel \implies \\
\Theta ; \Phi ; \mathcal{B} ; \Gamma ; \Delta \vdash_{wf} s1 ;; s2 : b \\
\\
| \text{ wfS-matchI: } \parallel \text{ wfV } \Theta\ \mathcal{B}\ \Gamma\ v\ (B\text{-id } tid) ; \\
(AF\text{-typedef } tid\ dclist) \in set\ \Theta ; \\
\text{ wfD } \Theta\ \mathcal{B}\ \Gamma\ \Delta ; \\
\Theta \vdash_{wf} \Phi ; \\
\Theta ; \Phi ; \mathcal{B} ; \Gamma ; \Delta ; tid ; dclist \vdash_{wf} cs : b \parallel \implies \Theta ; \Phi ; \mathcal{B} ; \Gamma ; \Delta \vdash_{wf} AS\text{-match } v\ cs : b \\
\\
| \text{ wfS-branchI: } \parallel \Theta ; \Phi ; \mathcal{B} ; (x, b\text{-of } \tau, C\text{-true}) \#_{\Gamma} \Gamma ; \Delta \vdash_{wf} s : b ;
\end{array}$$

$$\begin{array}{l}
\text{atom } x \# (\Phi, \Theta, \mathcal{B}, \Gamma, \Delta, \Gamma, \tau); \\
\Theta; \mathcal{B}; \Gamma \vdash_{wf} \Delta \\
\boxed{\phantom{\Theta; \Phi; \mathcal{B}; \Gamma; \Delta; tid; dc; \tau \vdash_{wf} dc \ x \Rightarrow s : b}} \Rightarrow \\
\Theta; \Phi; \mathcal{B}; \Gamma; \Delta; tid; dc; \tau \vdash_{wf} dc \ x \Rightarrow s : b \\
\\
| \text{ wfS-finalI: } \boxed{\phantom{\Theta; \Phi; \mathcal{B}; \Gamma; \Delta; tid; dc; t \vdash_{wf} cs : b}} \\
\boxed{\phantom{\Theta; \Phi; \mathcal{B}; \Gamma; \Delta; tid; [(dc, t)] \vdash_{wf} AS-final \ cs : b}} \Rightarrow \\
\Theta; \Phi; \mathcal{B}; \Gamma; \Delta; tid; [(dc, t)] \vdash_{wf} AS-final \ cs : b \\
\\
| \text{ wfS-cons: } \boxed{\phantom{\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}} \\
\boxed{\phantom{\Theta; \Phi; \mathcal{B}; \Gamma; \Delta; tid; (dc, t) \# dclist \vdash_{wf} AS-cons \ cs \ css : b}} \Rightarrow \\
\Theta; \Phi; \mathcal{B}; \Gamma; \Delta; tid; (dc, t) \# dclist \vdash_{wf} AS-cons \ cs \ css : b \\
\\
| \text{ wfD-emptyI: } \Theta; \mathcal{B} \vdash_{wf} \Gamma \Rightarrow \Theta; \mathcal{B}; \Gamma \vdash_{wf} \boxed{}_{\Delta} \\
| \text{ wfD-cons: } \boxed{\phantom{\Theta; \mathcal{B}; \Gamma \vdash_{wf} \Delta :: \Delta; \\ \Theta; \mathcal{B}; \Gamma \vdash_{wf} \tau; \\ u \notin fst \text{ ' setD } \Delta}} \Rightarrow \Theta; \mathcal{B}; \Gamma \vdash_{wf} ((u, \tau) \#_{\Delta} \Delta) \\
\\
| \text{ wfPhi-emptyI: } \vdash_{wf} \Theta \Rightarrow \Theta \vdash_{wf} \boxed{} \\
| \text{ wfPhi-consI: } \boxed{\phantom{f \notin name-of-fun \text{ ' set } \Phi; \\ \Theta; \Phi \vdash_{wf} ft; \\ \Theta \vdash_{wf} \Phi}} \\
\boxed{\phantom{\Theta \vdash_{wf} ((AF-fundef \ f \ ft) \# \Phi)}} \Rightarrow \\
\Theta \vdash_{wf} ((AF-fundef \ f \ ft) \# \Phi) \\
| \text{ wfFTNone: } \Theta; \Phi; \{|\} \vdash_{wf} ft \Rightarrow \Theta; \Phi \vdash_{wf} AF-fun-typ-none \ ft \\
| \text{ wfFTSome: } \Theta; \Phi; \{|\ bv |\} \vdash_{wf} ft \Rightarrow \Theta; \Phi \vdash_{wf} AF-fun-typ-some \ bv \ ft \\
\\
| \text{ wfFTI: } \boxed{\phantom{\Theta; B \vdash_{wf} b; \\ supp \ s \subseteq \{atom \ x\} \cup supp \ B; \\ supp \ c \subseteq \{atom \ x\}; \\ \Theta; B; (x, b, c) \#_{\Gamma} GNil \vdash_{wf} \tau; \\ \Theta \vdash_{wf} \Phi}} \\
\boxed{\phantom{\Theta; \Phi; B \vdash_{wf} (AF-fun-typ \ x \ b \ c \ \tau \ s)}} \Rightarrow \\
\Theta; \Phi; B \vdash_{wf} (AF-fun-typ \ x \ b \ c \ \tau \ s)
\end{array}$$

inductive-cases *wfE-elim*:

$$\begin{array}{l}
\Theta; \Phi; \mathcal{B}; \Gamma; \Delta \vdash_{wf} AE-val \ v : b \\
\Theta; \Phi; \mathcal{B}; \Gamma; \Delta \vdash_{wf} AE-op \ Plus \ v1 \ v2 : b \\
\Theta; \Phi; \mathcal{B}; \Gamma; \Delta \vdash_{wf} AE-op \ LEq \ v1 \ v2 : b \\
\Theta; \Phi; \mathcal{B}; \Gamma; \Delta \vdash_{wf} AE-fst \ v1 : b \\
\Theta; \Phi; \mathcal{B}; \Gamma; \Delta \vdash_{wf} AE-snd \ v1 : b \\
\Theta; \Phi; \mathcal{B}; \Gamma; \Delta \vdash_{wf} AE-concat \ v1 \ v2 : b \\
\Theta; \Phi; \mathcal{B}; \Gamma; \Delta \vdash_{wf} AE-len \ v1 : b \\
\Theta; \Phi; \mathcal{B}; \Gamma; \Delta \vdash_{wf} AE-op \ opp \ v1 \ v2 : b \\
\Theta; \Phi; \mathcal{B}; \Gamma; \Delta \vdash_{wf} AE-app \ f \ v : b \\
\Theta; \Phi; \mathcal{B}; \Gamma; \Delta \vdash_{wf} AE-appP \ f \ b' \ v : b
\end{array}$$

$\Theta; \Phi; \mathcal{B}; \Gamma; \Delta \vdash_{wf} AE\text{-mvar } u : b$

inductive-cases *wfCS-elim*:

$\Theta; \Phi; \mathcal{B}; \Gamma; \Delta; tid; dc; t \vdash_{wf} (cs::branch\text{-}s) : b$

$\Theta; \Phi; \mathcal{B}; \Gamma; \Delta; tid; dc \vdash_{wf} (cs::branch\text{-}list) : b$

inductive-cases *wfPhi-elim*:

$\Theta \vdash_{wf} []$

$\Theta \vdash_{wf} ((AF\text{-fundef } f\ ft)\#\Pi)$

$\Theta \vdash_{wf} (fd\#\Phi::\Phi)$

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

inductive-cases *wfFTQ-elim*:

$\Theta; \Phi \vdash_{wf} AF\text{-fun-typr-none } ft$

$\Theta; \Phi \vdash_{wf} AF\text{-fun-typr-some } bv\ ft$

$\Theta; \Phi \vdash_{wf} AF\text{-fun-typr-some } bv\ (AF\text{-fun-typr } x\ b\ c\ \tau\ s)$

inductive-cases *wfFT-elim*:

$\Theta; \Phi; \mathcal{B} \vdash_{wf} AF\text{-fun-typr } x\ b\ c\ \tau\ s$

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

inductive-cases *wfD-elim*:

$\Pi; \mathcal{B}; (\Gamma::\Gamma) \vdash_{wf} []_{\Delta}$

$\Pi; \mathcal{B}; (\Gamma::\Gamma) \vdash_{wf} (u, \tau) \#_{\Delta} \Delta::\Delta$

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 $\Theta\ \Phi\ \mathcal{B}\ \Gamma\ \Delta\ b'\ bv\ v\ \tau\ f\ x\ b\ c\ s$)

moreover hence *atom bv # 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 $\Theta\ \Phi\ \mathcal{B}\ \Gamma\ \Delta\ b'\ bv\ v\ \tau\ f\ x\ b\ c\ s$)

then show *?case by auto*

next

case (3 $\Phi\ \Theta\ \mathcal{B}\ \Gamma\ \Delta\ e\ b'\ x\ s\ b$)

moreover hence *atom x # LET x = e IN s using fresh-prodN by auto*

ultimately show *?case using fresh-prodN fresh-star-def by fastforce*

next

case (4 $\Phi\ \Theta\ \mathcal{B}\ \Gamma\ \Delta\ e\ b'\ x\ s\ b$)

then show *?case by auto*

next

case (5 $\Theta\ \Phi\ \mathcal{B}\ x\ c\ \Gamma\ \Delta\ s\ b$)

hence $atom\ x \# ASSERT\ c\ IN\ s$ **using** $s\text{-branch-}s\text{-branch-list.fresh}$ **by** $auto$
 then show $?case$ **using** $fresh\text{-prodN}\ fresh\text{-star-def}\ 5$ **by** $fastforce$
 next
 case $(6\ \Theta\ \Phi\ \mathcal{B}\ x\ c\ \Gamma\ \Delta\ s\ b)$
 then show $?case$ **by** $auto$
 next
 case $(7\ \Phi\ \Theta\ \mathcal{B}\ \Gamma\ \Delta\ s1\ \tau\ x\ s2\ b)$
 hence $atom\ x \# \tau \wedge atom\ x \# s1$ **using** $fresh\text{-prodN}$ **by** $metis$
 moreover hence $atom\ x \# LET\ x : \tau = s1\ IN\ s2$
 using $s\text{-branch-}s\text{-branch-list.fresh}(\beta)[of\ atom\ x\ x\ \tau\ s1\ s2]$ $fresh\text{-prodN}$ **by** $simp$
 ultimately show $?case$ **using** $fresh\text{-prodN}\ fresh\text{-star-def}\ 7$ **by** $fastforce$
 next
 case $(8\ \Phi\ \Theta\ \mathcal{B}\ \Gamma\ \Delta\ s1\ \tau\ x\ s2\ b)$
 then show $?case$ **by** $auto$
 next
 case $(9\ \Theta\ \mathcal{B}\ \Gamma\ \tau\ v\ u\ \Phi\ \Delta\ b\ s)$
 moreover hence $atom\ u \# AS\text{-var}\ u\ \tau\ v\ s$ **using** $fresh\text{-prodN}\ s\text{-branch-}s\text{-branch-list.fresh}$ **by** $simp$
 ultimately show $?case$ **using** $fresh\text{-star-def}\ fresh\text{-prodN}\ s\text{-branch-}s\text{-branch-list.fresh}$ **by** $fastforce$
 next
 case $(10\ \Theta\ \mathcal{B}\ \Gamma\ \tau\ v\ u\ \Phi\ \Delta\ b\ s)$
 then show $?case$ **by** $auto$
 next
 case $(11\ \Phi\ \Theta\ \mathcal{B}\ x\ \tau\ \Gamma\ \Delta\ s\ b\ tid\ dc)$
 moreover have $atom\ x \# (dc\ x \Rightarrow s)$ **using** $pure\text{-fresh}\ s\text{-branch-}s\text{-branch-list.fresh}$ **by** $auto$
 ultimately show $?case$ **using** $fresh\text{-prodN}\ fresh\text{-star-def}\ pure\text{-fresh}$ **by** $fastforce$
 next
 case $(12\ \Phi\ \Theta\ \mathcal{B}\ x\ \tau\ \Gamma\ \Delta\ s\ b\ tid\ dc)$
 then show $?case$ **by** $auto$
 qed

inductive $wfVDs :: var\text{-def}\ list \Rightarrow bool$ **where**

$wfVDs\text{-nilI} : wfVDs\ []$

$| wfVDs\text{-consI} : [$
 $\quad atom\ u \# ts;$
 $\quad wfV\ ([::\Theta)\ \{\|\}\ GNil\ v\ (b\text{-of}\ \tau));$
 $\quad wfT\ ([::\Theta)\ \{\|\}\ GNil\ \tau;$
 $\quad wfVDs\ ts$
 $] \Rightarrow wfVDs\ ((AV\text{-def}\ u\ \tau\ v)\#ts)$

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. S_{Ut} 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 $rcl\text{-}val = S_{\text{Bitvec}} \text{ bit list} \mid S_{\text{Num}} \text{ int} \mid S_{\text{Bool}} \text{ bool} \mid S_{\text{Pair}} \text{ rcl-val rcl-val} \mid$
 $S_{\text{Cons}} \text{ tyid string rcl-val} \mid S_{\text{Consp}} \text{ tyid string b rcl-val} \mid$
 $S_{\text{Unit}} \mid S_{\text{Ut}} \text{ rcl-val}$

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 $rcl\text{-}sort = S_{\text{bool}} \mid S_{\text{int}} \mid S_{\text{unit}} \mid S_{\text{pair}} \text{ rcl-sort rcl-sort} \mid S_{\text{id}} \text{ tyid} \mid S_{\text{app}} \text{ tyid}$
 $rcl\text{-}sort \mid S_{\text{bitvec}} \mid S_{\text{ut}}$

type-synonym $valuation = (x, rcl\text{-}val) \text{ map}$

type-synonym $type\text{-}valuation = (bv, rcl\text{-}sort) \text{ map}$

inductive $wfRCV :: \Theta \Rightarrow rcl\text{-}val \Rightarrow b \Rightarrow bool \text{ (} - \vdash - : - [50, 50] \text{) where}$
 $wfRCV\text{-}B_{\text{Bitvec}}I: P \vdash (S_{\text{Bitvec}} bv) : B\text{-}bitvec$
 $\mid wfRCV\text{-}B_{\text{Int}}I: P \vdash (S_{\text{Num}} n) : B\text{-}int$
 $\mid wfRCV\text{-}B_{\text{Bool}}I: P \vdash (S_{\text{Bool}} b) : B\text{-}bool$
 $\mid wfRCV\text{-}B_{\text{Pair}}I: \llbracket P \vdash s1 : b1 ; P \vdash s2 : b2 \rrbracket \Longrightarrow P \vdash (S_{\text{Pair}} s1 s2) : (B\text{-}pair b1 b2)$
 $\mid wfRCV\text{-}B_{\text{Cons}}I: \llbracket AF\text{-}typedef \text{ s dclist } \in \text{ set } \Theta;$
 $\quad (dc, \llbracket x : b \mid c \rrbracket) \in \text{ set dclist} ;$
 $\quad \Theta \vdash s1 : b \rrbracket \Longrightarrow \Theta \vdash (S_{\text{Cons}} s dc s1) : (B\text{-}id s)$
 $\mid wfRCV\text{-}B_{\text{ConsPI}}I: \llbracket AF\text{-}typedef\text{-}poly \text{ s bv dclist } \in \text{ set } \Theta;$

```

      (dc, { x : b | c }) ∈ set dclist ;
      atom bv # (Θ, SConsp s dc b' s1, B-app s b');
      Θ ⊢ s1 : b[bv::=b']bb ] ⇒ Θ ⊢ (SConsp s dc b' s1) : (B-app s b')
| wfRCV-BUnitI: P ⊢ SUnit : B-unit
| wfRCV-BVarI: P ⊢ (SUnit n) : (B-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-elim :

```

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 (SUnit s1) b

```

thm wfRCV-elim(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 substituted in explicit.

inductive wfRCV-subst:: $\Theta \Rightarrow rcl\text{-}val \Rightarrow b \Rightarrow (bv*b) \text{ option} \Rightarrow bool$ **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 ] ⇒ wfRCV-subst P
(SPair s1 s2) (B-pair b1 b2) sub
| wfRCV-subst-BConsI: [ AF-typedef s dclist ∈ set Θ;
  (dc, { x : b | c }) ∈ set dclist ;
  wfRCV-subst Θ s1 b None ] ⇒ wfRCV-subst Θ (SCons s dc s1) (B-id s) sub
| wfRCV-subst-BConspI: [ AF-typedef-poly s bv dclist ∈ set Θ;
  (dc, { x : b | c }) ∈ set dclist ;
  wfRCV-subst Θ s1 (b[bv::=b']bb) sub ] ⇒ wfRCV-subst Θ (SConsp s dc b' s1) (B-app s b') sub
| wfRCV-subst-BUnitI: wfRCV-subst P SUnit B-unit sub

```


$| \text{wfRCV-subst-BVar1I}: \text{bvar} \neq \text{bv} \implies \text{wfRCV-subst } P \text{ (S} \text{U}t \text{ } n) \text{ (B-var } \text{bv}) \text{ (Some (bvar, bin))}$
 $| \text{wfRCV-subst-BVar2I}: \llbracket \text{bvar} = \text{bv}; \text{wfRCV-subst } P \text{ } s \text{ bin None} \rrbracket \implies \text{wfRCV-subst } P \text{ } s \text{ (B-var } \text{bv})$
 $(\text{Some (bvar, bin)})$
 $| \text{wfRCV-subst-BVar3I}: \text{wfRCV-subst } P \text{ (S} \text{U}t \text{ } n) \text{ (B-var } \text{bv}) \text{ None}$
equivariance wfRCV-subst
nominal-inductive wfRCV-subst .

7.1.2 Evaluation base-types

inductive $\text{eval-b} :: \text{type-valuation} \Rightarrow b \Rightarrow \text{rcl-sort} \Rightarrow \text{bool} \text{ (- } \llbracket \text{ - } \rrbracket \sim \text{ -)}$ **where**
 $v \llbracket \text{B-bool} \rrbracket \sim \text{S-bool}$
 $| v \llbracket \text{B-int} \rrbracket \sim \text{S-int}$
 $| \text{Some } s = v \text{ bv} \implies v \llbracket \text{B-var } \text{bv} \rrbracket \sim s$
equivariance eval-b
nominal-inductive eval-b .

7.1.3 Wellformed Evaluation

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

7.1.4 Evaluating Terms

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

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

inductive-cases eval-v-elim :

$i \llbracket \text{V-lit } l \rrbracket \sim s$
 $i \llbracket \text{V-var } x \rrbracket \sim s$
 $i \llbracket \text{V-pair } v1 \text{ } v2 \rrbracket \sim s$
 $i \llbracket \text{V-cons } \text{tyid } \text{dc } v \rrbracket \sim s$
 $i \llbracket \text{V-consp } \text{tyid } \text{dc } b \text{ } v \rrbracket \sim s$

inductive $\text{eval-e} :: \text{valuation} \Rightarrow \text{ce} \Rightarrow \text{rcl-val} \Rightarrow \text{bool} \text{ (- } \llbracket \text{ - } \rrbracket \sim \text{ -)}$ **where**
 $\text{eval-e-valI}: i \llbracket v \rrbracket \sim sv \implies i \llbracket \text{CE-val } v \rrbracket \sim sv$

$| \text{eval-e-plusI}: \llbracket i \llbracket v1 \rrbracket \sim \text{SNum } n1; i \llbracket v2 \rrbracket \sim \text{SNum } n2 \rrbracket \implies i \llbracket (\text{CE-op Plus } v1 \ v2) \rrbracket \sim (\text{SNum } (n1+n2))$
 $| \text{eval-e-leqI}: \llbracket i \llbracket v1 \rrbracket \sim (\text{SNum } n1); i \llbracket v2 \rrbracket \sim (\text{SNum } n2) \rrbracket \implies i \llbracket (\text{CE-op LEq } v1 \ v2) \rrbracket \sim (\text{SBool } (n1 \leq n2))$
 $| \text{eval-e-fstI}: \llbracket i \llbracket v \rrbracket \sim \text{SPair } v1 \ v2 \rrbracket \implies i \llbracket (\text{CE-fst } v) \rrbracket \sim v1$
 $| \text{eval-e-sndI}: \llbracket i \llbracket v \rrbracket \sim \text{SPair } v1 \ v2 \rrbracket \implies i \llbracket (\text{CE-snd } v) \rrbracket \sim v2$
 $| \text{eval-e-concatI}: \llbracket i \llbracket v1 \rrbracket \sim (\text{SBitvec } bv1); i \llbracket v2 \rrbracket \sim (\text{SBitvec } bv2) \rrbracket \implies i \llbracket (\text{CE-concat } v1 \ v2) \rrbracket \sim (\text{SBitvec } (bv1 @ bv2))$
 $| \text{eval-e-lenI}: \llbracket i \llbracket v \rrbracket \sim (\text{SBitvec } bv) \rrbracket \implies i \llbracket (\text{CE-len } v) \rrbracket \sim (\text{SNum } (\text{int } (\text{List.length } bv)))$

equivariance *eval-e*

nominal-inductive *eval-e* .

thm *eval-e.induct*

inductive-cases *eval-e-elim*:

$i \llbracket (\text{CE-val } v) \rrbracket \sim s$
 $i \llbracket (\text{CE-op Plus } v1 \ v2) \rrbracket \sim s$
 $i \llbracket (\text{CE-op LEq } v1 \ v2) \rrbracket \sim s$
 $i \llbracket (\text{CE-fst } v) \rrbracket \sim s$
 $i \llbracket (\text{CE-snd } v) \rrbracket \sim s$
 $i \llbracket (\text{CE-concat } v1 \ v2) \rrbracket \sim s$
 $i \llbracket (\text{CE-len } v) \rrbracket \sim s$

inductive *eval-c* :: *valuation* \Rightarrow *c* \Rightarrow *bool* \Rightarrow *bool* (*-* \llbracket *-* $\rrbracket \sim$ *-*) **where**

$\text{eval-c-trueI}: i \llbracket \text{C-true} \rrbracket \sim \text{True}$
 $\text{eval-c-falseI}: i \llbracket \text{C-false} \rrbracket \sim \text{False}$
 $\text{eval-c-conjI}: \llbracket i \llbracket c1 \rrbracket \sim b1 ; i \llbracket c2 \rrbracket \sim b2 \rrbracket \implies i \llbracket (\text{C-conj } c1 \ c2) \rrbracket \sim (b1 \wedge b2)$
 $\text{eval-c-disjI}: \llbracket i \llbracket c1 \rrbracket \sim b1 ; i \llbracket c2 \rrbracket \sim b2 \rrbracket \implies i \llbracket (\text{C-disj } c1 \ c2) \rrbracket \sim (b1 \vee b2)$
 $\text{eval-c-impI}: \llbracket i \llbracket c1 \rrbracket \sim b1 ; i \llbracket c2 \rrbracket \sim b2 \rrbracket \implies i \llbracket (\text{C-imp } c1 \ c2) \rrbracket \sim (b1 \longrightarrow b2)$
 $\text{eval-c-notI}: \llbracket i \llbracket c \rrbracket \sim b \rrbracket \implies i \llbracket (\text{C-not } c) \rrbracket \sim (\neg b)$
 $\text{eval-c-eqI}: \llbracket i \llbracket e1 \rrbracket \sim sv1 ; i \llbracket e2 \rrbracket \sim sv2 \rrbracket \implies i \llbracket (\text{C-eq } e1 \ e2) \rrbracket \sim (sv1 = sv2)$

equivariance *eval-c*

nominal-inductive *eval-c* .

inductive-cases *eval-c-elim*:

$i \llbracket \text{C-true} \rrbracket \sim \text{True}$
 $i \llbracket \text{C-false} \rrbracket \sim \text{False}$
 $i \llbracket (\text{C-conj } c1 \ c2) \rrbracket \sim s$
 $i \llbracket (\text{C-disj } c1 \ c2) \rrbracket \sim s$
 $i \llbracket (\text{C-imp } c1 \ c2) \rrbracket \sim s$
 $i \llbracket (\text{C-not } c) \rrbracket \sim s$
 $i \llbracket (\text{C-eq } e1 \ e2) \rrbracket \sim s$
 $i \llbracket \text{C-true} \rrbracket \sim s$
 $i \llbracket \text{C-false} \rrbracket \sim s$

7.1.5 Satisfiability

inductive *is-satis* :: *valuation* \Rightarrow *c* \Rightarrow *bool* (*-* \models *-*) **where**

$i \llbracket c \rrbracket \sim \text{True} \implies i \models c$

equivariance *is-satis*

nominal-inductive *is-satis* .

nominal-function *is-satis-g* :: *valuation* $\Rightarrow \Gamma \Rightarrow \text{bool}$ (*-* \models *-*) **where**
i \models *GNil* = *True*
| *i* \models ((*x*,*b*,*c*) $\#_{\Gamma}$ *G*) = (*i* \models *c* \wedge *i* \models *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* :: $\Theta \Rightarrow \mathcal{B} \Rightarrow \Gamma \Rightarrow c \Rightarrow \text{bool}$ (*-* ; *-* ; *-* \models *-* [*50*, *50*] *50*) **where**
P ; *B* ; *G* $\models c$ = ((*P* ; *B* ; *G* \vdash_{wf} *c*) \wedge ($\forall i. (P ; G \vdash i) \wedge i \models G \longrightarrow i \models c$))
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* \vdash_{wf} *G*
shows *P* ; *B* ; *G* \models *C-true*
proof –
have $\forall i. i \models C\text{-true}$ **using** *is-satis.simps eval-c-trueI* **by** *simp*
moreover have *P* ; *B* ; *G* \vdash_{wf} *C-true* **using** *wfC-trueI assms* **by** *simp*
ultimately show *?thesis* **using** *valid.simps* **by** *simp*
qed

inductive *split* :: *int* \Rightarrow *bit list* \Rightarrow *bit list* * *bit list* \Rightarrow *bool* **where**
split 0 *xs* ([], *xs*)
| *split* *m* *xs* (*ys*,*zs*) \Longrightarrow *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 \leq *n* \wedge *n* \leq *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  $\text{supp } (\llbracket z : b \mid \text{CE-val } (V\text{-var } z) \rrbracket == \text{CE-val } (V\text{-var } x) \rrbracket) = \{ \text{atom } x \} \cup \text{supp } b$ 
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  $\exists y::x. \text{atom } y \# (s, c, \tau, t) \wedge (\text{AF-fundef } f (\text{AF-fun-typ-none } (\text{AF-fun-typ } x \ b \ c \ \tau \ s)) = \text{AF-fundef } f (\text{AF-fun-typ-none } (\text{AF-fun-typ } y \ b \ ((y \leftrightarrow x) \cdot c) ((y \leftrightarrow x) \cdot \tau) ((y \leftrightarrow x) \cdot s))))$ 
proof -
  obtain  $y::x$  where  $y: \text{atom } y \# (s, c, \tau, t)$  using obtain-fresh by blast
  moreover have  $\text{AF-fundef } f (\text{AF-fun-typ-none } (\text{AF-fun-typ } y \ b \ ((y \leftrightarrow x) \cdot c) ((y \leftrightarrow x) \cdot \tau) ((y \leftrightarrow x) \cdot s))) = (\text{AF-fundef } f (\text{AF-fun-typ-none } (\text{AF-fun-typ } x \ b \ c \ \tau \ s)))$ 
  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  $(\text{AF-fun-typ } y \ b \ ((y \leftrightarrow x) \cdot c) ((y \leftrightarrow x) \cdot \tau) ((y \leftrightarrow x) \cdot s)) = (\text{AF-fun-typ } x \ b \ c \ \tau \ s)$  proof(subst fun-typ.eq-iff, subst Abs1-eq-iff(3))
      show  $((y = x \wedge (((y \leftrightarrow x) \cdot c, (y \leftrightarrow x) \cdot \tau), (y \leftrightarrow x) \cdot s) = ((c, \tau), s) \vee y \neq x \wedge (((y \leftrightarrow x) \cdot c, (y \leftrightarrow x) \cdot \tau), (y \leftrightarrow x) \cdot s) = (y \leftrightarrow x) \cdot ((c, \tau), s) \wedge \text{atom } y \# ((c, \tau), s))) \wedge b = b)$ 
    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  $\text{Some } (\text{AF-fundef } f \ ft) = \text{lookup-fun } \Phi \ f$ 
  shows  $\text{AF-fundef } f \ ft \in \text{set } \Phi$ 
using assms proof (induct  $\Phi$ )
  case Nil
  then show ?case by auto
next
  case (Cons a  $\Phi$ )
  then show ?case using lookup-fun.simps
    by (metis fun-def.exhaust insert-iff list.simps(15) option.inject)
qed

```

```

lemma rig-dom-eq:
   $\text{dom } (G[x \mapsto c]) = \text{dom } G$ 
proof(induct G rule:  $\Gamma$ .induct)
  case GNil

```

```

  then show ?case using replace-in-g.simps by presburger
next
case (GCons xbc  $\Gamma'$ )
  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  $\Gamma$  x
  shows Some (b,c') = lookup ( $\Gamma[x \mapsto c']$ ) x
using assms proof(induct  $\Gamma$  rule:  $\Gamma$ -induct)
  case GNil
  then show ?case by auto
next
case (GCons x b c  $\Gamma'$ )
  then show ?case using replace-in-g.simps lookup.simps by auto
qed

```

```

lemma lookup-in-rig-neq:
  assumes Some (b,c) = lookup  $\Gamma$  y and  $x \neq y$ 
  shows Some (b,c) = lookup ( $\Gamma[x \mapsto c']$ ) y
using assms proof(induct  $\Gamma$  rule:  $\Gamma$ -induct)
  case GNil
  then show ?case by auto
next
case (GCons x' b' c'  $\Gamma'$ )
  then show ?case using replace-in-g.simps lookup.simps by auto
qed

```

```

lemma lookup-in-rig:
  assumes Some (b,c) = lookup  $\Gamma$  y
  shows  $\exists c''. \text{Some } (b,c'') = \text{lookup } (\Gamma[x \mapsto c']) y$ 
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  $x \notin \text{fst } \text{toSet } \Gamma'$ 
  shows Some (b1,c1) = lookup ( $\Gamma'@(x,b1,c1)\#_{\Gamma}\Gamma$ ) x
  using assms by(induct  $\Gamma'$ ,auto)

```

```

lemma lookup-inside2:
  assumes Some (b1,c1) = lookup ( $\Gamma'@((x,b0,c0)\#_{\Gamma}\Gamma)$ ) y and  $x \neq y$ 
  shows Some (b1,c1) = lookup ( $\Gamma'@((x,b0,c0')\#_{\Gamma}\Gamma)$ ) y
  using assms by(induct  $\Gamma'$  rule:  $\Gamma$ .induct,auto+)

```

```

fun tail:: 'a list  $\Rightarrow$  'a list where
  tail [] = []

```

| $\text{tail } (x \# xs) = xs$

lemma *lookup-options*:

assumes $\text{Some } (b,c) = \text{lookup } (xt \#_{\Gamma} G) x$

shows $((x,b,c) = xt) \vee (\text{Some } (b,c) = \text{lookup } G x)$

by (*metis* *assms* *lookup.simps(2)* *option.inject surj-pair*)

lemma *lookup-x*:

assumes $\text{Some } (b,c) = \text{lookup } G x$

shows $x \in \text{fst } \text{toSet } G$

using *assms*

by(*induct* *G* *rule*: $\Gamma.\text{induct}$,*auto*+))

lemma *GCons-eq-appendI*:

fixes $xs1::\Gamma$

shows $[[x \#_{\Gamma} xs1 = ys; xs = xs1 @ zs]] ==> x \#_{\Gamma} xs = ys @ zs$

by (*drule sym*) *simp*

lemma *split-G*: $x : \text{toSet } xs \implies \exists ys zs. xs = ys @ x \#_{\Gamma} zs$

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 toSet.simps(2)*)

qed

lemma *lookup-not-empty*:

assumes $\text{Some } \tau = \text{lookup } G x$

shows $G \neq \text{GNil}$

using *assms* **by** *auto*

lemma *lookup-in-g*:

assumes $\text{Some } (b,c) = \text{lookup } \Gamma x$

shows $(x,b,c) \in \text{toSet } \Gamma$

using *assms* **apply**(*induct* Γ , *simp*)

using *lookup-options* **by** *fastforce*

lemma *lookup-split*:

fixes $\Gamma::\Gamma$

assumes $\text{Some } (b,c) = \text{lookup } \Gamma x$

shows $\exists G G' . \Gamma = G' @ (x,b,c) \#_{\Gamma} G$

by (*meson* *assms(1)* *lookup-in-g split-G*)

lemma *toSet-splitU*[*simp*]:

$(x',b',c') \in \text{toSet } (\Gamma' @ (x, b, c) \#_{\Gamma} \Gamma) \longleftrightarrow (x',b',c') \in (\text{toSet } \Gamma' \cup \{(x, b, c)\} \cup \text{toSet } \Gamma)$

using *append-g-toSetU toSet.simps* **by** *auto*

lemma *toSet-splitP*[*simp*]:

$(\forall (x', b', c') \in \text{toSet } (\Gamma' @ (x, b, c) \#_{\Gamma} \Gamma). P x' b' c') \longleftrightarrow (\forall (x', b', c') \in \text{toSet } \Gamma'. P x' b' c') \wedge P x b c \wedge (\forall (x', b', c') \in \text{toSet } \Gamma. P x' b' c') \text{ (is } ?A \longleftrightarrow ?B)$

using *toSet-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  $\Gamma'$  rule: $\Gamma$ -induct)
  case GNil
  then show ?case by auto
next
  case (GCons x1 b1 c1  $\Gamma'$ )
  then show ?case by auto
qed

lemma supp-list-member:
  fixes  $x::'a::fs$  and  $l::'a$  list
  assumes  $x \in \text{set } l$ 
  shows supp  $x \subseteq \text{supp } l$ 
  using assms apply(induct  $l$ , auto)
  using supp-Cons by auto

lemma GNil-append:
  assumes  $GNil = G1@G2$ 
  shows  $G1 = GNil \wedge G2 = GNil$ 
proof(rule ccontr)
  assume  $\neg (G1 = GNil \wedge G2 = GNil)$ 
  hence  $G1@G2 \neq GNil$  using append-g.simps by (metis  $\Gamma$ .distinct(1)  $\Gamma$ .exhaust)
  thus False using assms by auto
qed

lemma GCons-eq-append-conv:
  fixes  $xs::\Gamma$ 
  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

lemma dclist-distinct-unique:
  assumes  $(dc, const) \in \text{set } dclist2$  and  $(cons, const1) \in \text{set } dclist2$  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  $u \nVdash \delta$ 
  shows  $u \notin \text{fst } \delta$ 
using assms proof(induct  $\delta$ )
  case Nil
  then show ?case by auto

```


next
 case (*Cons ut δ'*)
 obtain *u'* and *t'* where $\ast:ut = (u', t')$ by *fastforce*
 hence $\text{atom } u \# ut \wedge \text{atom } u \# \delta'$ using *fresh-Cons Cons* by *auto*
 moreover hence $\text{atom } u \# \text{fst } ut$ using \ast *fresh-Pair*[of $\text{atom } u \ u' \ t'$] *Cons* by *auto*
 ultimately show *?case* using *Cons* by *auto*
qed

lemma *bv-not-in-bset-supp*:
 fixes *bv::bv*
 assumes $bv \notin B$
 shows $\text{atom } bv \notin \text{supp } B$
proof –
 have $\ast:\text{supp } B = \text{fset } (\text{fimage } \text{atom } B)$
 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 *u-fresh-d*:
 assumes $\text{atom } u \# D$
 shows $u \notin \text{fst } \text{'setD } D$
 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

7.5 Type Definitions

lemma *exist-fresh-bv*:
 fixes *tm::'a::fs*
 shows $\exists bva2 \ dclist2. AF\text{-typedef-poly } tyid \ bva \ dclist = AF\text{-typedef-poly } tyid \ bva2 \ dclist2 \wedge$
 $\text{atom } bva2 \# tm$
proof –
 obtain *bva2::bv* where $\ast:\text{atom } bva2 \# (bva, dclist, tyid, tm)$ using *obtain-fresh* by *metis*
 moreover hence $bva2 \neq bva$ using *fresh-at-base* by *auto*
 moreover have $dclist = (bva \leftrightarrow bva2) \cdot (bva2 \leftrightarrow bva) \cdot dclist$ by *simp*
 moreover have $\text{atom } bva \# (bva2 \leftrightarrow bva) \cdot dclist$ **proof** –
 have $\text{atom } bva2 \# dclist$ using \ast *fresh-prodN* by *auto*
 hence $\text{atom } ((bva2 \leftrightarrow bva) \cdot bva2) \# (bva2 \leftrightarrow bva) \cdot dclist$ using *fresh-eqvt True-eqvt*
proof –
 have $(bva2 \leftrightarrow bva) \cdot \text{atom } bva2 \# (bva2 \leftrightarrow bva) \cdot dclist$
 by (*metis True-eqvt (atom bva2 # dclist) fresh-eqvt*)
 then show *?thesis*
 by *simp*
qed
 thus *?thesis* by *auto*

qed
ultimately have $AF\text{-typedef-poly } tyid \ bva \ dclist = AF\text{-typedef-poly } tyid \ bva2 \ ((bva2 \leftrightarrow bva) \cdot dclist)$
unfolding $type\text{-def.eq-iff}$ $Abs1\text{-eq-iff}$ by metis
thus ?thesis using * fresh-prodN by metis
qed
lemma obtain-fresh-bv:
fixes $tm::'a::fs$
obtains $bva2::bv$ and $dclist2$ where $AF\text{-typedef-poly } tyid \ bva \ dclist = AF\text{-typedef-poly } tyid \ bva2 \ dclist2 \wedge$
 $atom \ bva2 \not\# tm$
using exist-fresh-bv by metis

7.6 Function Definitions

lemma fun-typ-flip:
fixes $bv1::bv$ and $c::bv$
shows $(bv1 \leftrightarrow c) \cdot AF\text{-fun-typ } x1 \ b1 \ c1 \ \tau1 \ s1 = AF\text{-fun-typ } x1 \ ((bv1 \leftrightarrow c) \cdot b1) \ ((bv1 \leftrightarrow c) \cdot c1) \ ((bv1 \leftrightarrow c) \cdot \tau1) \ ((bv1 \leftrightarrow c) \cdot s1)$
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 $AF\text{-fundeff } a \ (AF\text{-fun-typ-none } (AF\text{-fun-typ } xa \ ba \ ca \ \tau a \ sa)) = AF\text{-fundeff } (AF\text{-fun-typ-none } (AF\text{-fun-typ } x \ b \ c \ \tau \ s))$
shows $f = fa$ and $b = ba$ and $[[atom \ xa]]lst. \ sa = [[atom \ x]]lst. \ s$ and $[[atom \ xa]]lst. \ \tau a = [[atom \ x]]lst. \ \tau$ 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 $([[atom \ xa]]lst. \ ((ca, \tau a), sa) = [[atom \ x]]lst. \ ((c, \tau), s))$ 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 $[[atom \ xa]]lst. \ \tau a = [[atom \ x]]lst. \ \tau$ and
 $[[atom \ xa]]lst. \ ca = [[atom \ x]]lst. \ c$ using lst-snd lst-fst by metis+
qed

lemma fun-arg-unique-aux:
assumes $AF\text{-fun-typ } x1 \ b1 \ c1 \ \tau1' \ s1' = AF\text{-fun-typ } x2 \ b2 \ c2 \ \tau2' \ s2'$
shows $\{ x1 : b1 \mid c1 \} = \{ x2 : b2 \mid c2 \}$
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 $\tau.eq-iff$ by fast
qed

lemma fresh-x-neq:
fixes $x::x$ and $y::x$

shows $\text{atom } x \# y = (x \neq y)$
using *fresh-at-base fresh-def* **by** *auto*

lemma *obtain-fresh-z3*:
fixes $tm::'b::fs$
obtains $z::x$ **where** $\{ x : b \mid c \} = \{ z : b \mid c[x::=V\text{-var } z]_{cv} \} \wedge \text{atom } z \# tm \wedge \text{atom } z \# (x, c)$
proof –
obtain $z::x$ **and** $c':c$ **where** $z::\{ x : b \mid c \} = \{ z : b \mid c' \} \wedge \text{atom } z \# (tm, x, c)$ **using** *obtain-fresh-z2*
b-of.simps **by** *metis*
hence $c' = c[x::=V\text{-var } z]_{cv}$ **proof** –
have $([\text{atom } z]]\text{lst. } c' = [[\text{atom } x]]\text{lst. } c)$ **using** $z \tau.\text{eq-iff}$ **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 $\dots = c[x::=V\text{-var } z]_{cv}$
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 $\text{atom } u \# t$
by(*nominal-induct t rule:v.strong-induct,auto*)

lemma *u-fresh-ce*:
fixes $u::u$ **and** $t::ce$
shows $\text{atom } u \# t$
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 $\text{atom } u \# t$
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 $\text{atom } u \# t$
by(*induct t rule:\Gamma-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 $\text{atom } u \# t$
by(*nominal-induct t rule:\tau.strong-induct,auto simp add: \tau.fresh u-fresh-c u-fresh-b*)

lemma *b-of-c-of-eq*:
assumes $\text{atom } z \# \tau$

```

shows  $\llbracket z : b\text{-of } \tau \mid c\text{-of } \tau z \rrbracket = \tau$ 
using assms proof(nominal-induct  $\tau$  avoiding: z rule:  $\tau$ .strong-induct)
case (T-refined-type  $x1a$   $x2a$   $x3a$ )

hence  $\llbracket z : b\text{-of } \llbracket x1a : x2a \mid x3a \rrbracket \mid c\text{-of } \llbracket x1a : x2a \mid x3a \rrbracket z \rrbracket = \llbracket z : x2a \mid x3a[x1a::=V\text{-var}$ 
 $z]_{cv} \rrbracket$ 
using b-of.simps c-of.simps c-of-eq by auto
moreover have  $\llbracket z : x2a \mid x3a[x1a::=V\text{-var } z]_{cv} \rrbracket = \llbracket x1a : x2a \mid x3a \rrbracket$  using T-refined-type  $\tau$ .fresh
by auto
ultimately show ?case by auto
qed

```

```

lemma fresh-d-not-in:
  assumes atom  $u2 \# \Delta'$ 
  shows  $u2 \notin \text{fst } \text{'setD } \Delta'$ 
using assms proof(induct  $\Delta'$  rule:  $\Delta$ -induct)
  case DNil
  then show ?case by simp
next
  case (DCons  $u$   $t$   $\Delta'$ )
  hence  $*$ : atom  $u2 \# \Delta' \wedge \text{atom } u2 \# (u, t)$ 
  by (simp add: fresh-def supp-DCons)
  hence  $u2 \notin \text{fst } \text{'setD } \Delta'$  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*:

$(b\text{-of } (\tau[bv::=b]_{\tau b})) = (b\text{-of } \tau)[bv::=b]_{bb}$

proof –

obtain z' **and** b' **and** c' **where** $\tau = \{ \{ z' : b' \mid c' \} \}$ **using** *obtain-fresh-z* **by** *metis*

moreover **hence** $(b\text{-of } (\tau[bv::=b]_{\tau b})) = b\text{-of } \{ \{ z' : b'[bv::=b]_{bb} \mid c' \} \}$ **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 $\Gamma::\Gamma$ **and** $\Gamma'::\Gamma$ **and** $v::v$ **and** $e::e$ **and** $c::c$ **and** $\tau::\tau$ **and** $ts::(\text{string}*\tau)$ *list*

and $\Delta::\Delta$ **and** $b::b$ **and** $ftq::\text{fun-ty}p\text{-}q$ **and** $ft::\text{fun-ty}p$ **and** $ce::ce$ **and** $td::\text{type-def}$ **and** $s::s$

and $tm::'a::fs$

and $cs::\text{branch-}s$ **and** $css::\text{branch-list}$ **and** $\Theta::\Theta$

shows $\Theta; \mathcal{B}; \Gamma \vdash_{wf} (V\text{-consp } tyid \ dc \ b \ v) : b'' \implies (\exists \ bv \ dclist \ x \ b' \ c. b'' = B\text{-app } tyid \ b \wedge$

$AF\text{-typedef-poly } tyid \ bv \ dclist \in \text{set } \Theta \wedge$

$(dc, \{ \{ x : b' \mid c \} \}) \in \text{set } dclist \wedge$

$\Theta; \mathcal{B} \vdash_{wf} b \wedge \text{atom } bv \ \# \ (\Theta, \mathcal{B}, \Gamma, b, v) \wedge \Theta; \mathcal{B}; \Gamma \vdash_{wf} v : b'[bv::=b]_{bb} \wedge \text{atom } bv \ \# \ tm)$

and

$\Theta; \mathcal{B}; \Gamma \vdash_{wf} c \implies \text{True}$ **and**

$\Theta; \mathcal{B} \vdash_{wf} \Gamma \implies \text{True}$ **and**

$\Theta; \mathcal{B}; \Gamma \vdash_{wf} \tau \implies$

$\exists \ z \ b \ c. \tau = \{ \{ z : b \mid c \} \} \wedge \text{atom } z \ \# \ (\Theta, \mathcal{B}, \Gamma) \wedge \text{atom } z \ \# \ tm \wedge$

$\Theta; \mathcal{B} \vdash_{wf} b \wedge \Theta; \mathcal{B}; (z, b, \text{TRUE}) \ \#_{\Gamma} \Gamma \vdash_{wf} c$ **and**

$\Theta; \mathcal{B}; \Gamma \vdash_{wf} ts \implies \text{True}$ **and**

$\vdash_{wf} \Theta \implies \text{True}$ **and**

$\Theta; \mathcal{B} \vdash_{wf} b \implies \text{True}$ **and**

$\Theta; \mathcal{B}; \Gamma \vdash_{wf} ce : b' \implies \text{True}$ **and**

$\Theta \vdash_{wf} td \implies \text{True}$

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-conspl 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* :: $\Theta \Rightarrow \mathcal{B} \Rightarrow \Gamma \Rightarrow \Gamma \Rightarrow \text{bool}$ (- ; - \vdash_{wf} - < - [50,50,50] 50) **where**
wfExt T B G1 G2 = (*wfG T B G2* \wedge *wfG T B G1* \wedge *toSet G1* \subseteq *toSet G2*)

8.4 Context

lemma *wfG-cons[ms-wb]*:
fixes $\Gamma::\Gamma$
assumes *P; $\mathcal{B} \vdash_{wf} (z,b,c) \#_{\Gamma} \Gamma$*
shows *P; $\mathcal{B} \vdash_{wf} \Gamma \wedge \text{atom } z \# \Gamma \wedge \text{wfB } P \mathcal{B} b$*
using *wfG-elim(2)[OF assms]* **by** *metis*

lemma *wfG-cons2[ms-wb]*:
fixes $\Gamma::\Gamma$
assumes *P; $\mathcal{B} \vdash_{wf} zbc \#_{\Gamma} \Gamma$*
shows *P; $\mathcal{B} \vdash_{wf} \Gamma$*

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 $\Gamma::\Gamma$
assumes $\Theta; \mathcal{B} \vdash_{wf} \Gamma$ **and** $(x,b,c) \in \text{toSet } \Gamma$ **and** $(x,b',c') \in \text{toSet } \Gamma$
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 $(x,b,c)=a \wedge (x,b',c')=a \mid (x,b,c)=a \wedge (x,b',c') \neq a \mid (x,b,c) \neq a \wedge (x,b',c')=a \mid (x,b,c) \neq a \wedge (x,b',c') \neq a$ **by** *blast*
then show *?case proof(cases)*
case *1*
then show *?thesis by auto*
next
case *2*
hence *atom x $\# \Gamma$ using wfG-elim(2) GCons by blast*

moreover have $(x, b', c') \in \text{toSet } \Gamma$ **using** $G\text{Cons } 2$ **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 # Γ* **using** *wfG-elim3(2) GCons by blast*
 moreover have $(x, b, c) \in \text{toSet } \Gamma$ **using** $G\text{Cons } 3$ **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** $G\text{Cons wfG-elim3 by blast}$
 hence $\Theta; \mathcal{B} \vdash_{wf} \Gamma \wedge (x, b, c) \in \text{toSet } \Gamma \wedge (x, b', c') \in \text{toSet } \Gamma$ **using** $G\text{Cons wfG-elim3 4 xbc prod-cases3 set-GConsD using forget-subst-gv fresh-GCons fresh-GNil fresh-gamma-elem Γ .distinct subst-gv.simps 4 GCons by meson$
 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 Γ x*
 shows $(x, b, c) \in \text{toSet } \Gamma \wedge (\forall b' c'. (x, b', c') \in \text{toSet } \Gamma \longrightarrow b'=b \wedge c'=c)$
using *assms proof(induct Γ rule: Γ .induct)*
 case $G\text{Nil}$
 then show *?case by auto*
 next
 case ($G\text{Cons } xbc \Gamma$)
 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 toSet.simps wfG-elim3 by metis*
 qed

lemma *lookup-if2*:

assumes $\text{wfG } P \mathcal{B} \Gamma$ **and** $(x, b, c) \in \text{toSet } \Gamma \wedge (\forall b' c'. (x, b', c') \in \text{toSet } \Gamma \longrightarrow b'=b \wedge c'=c)$
 shows *Some (b,c) = lookup Γ x*
using *assms proof(induct Γ rule: Γ .induct)*
 case $G\text{Nil}$
 then show *?case by auto*
 next
 case ($G\text{Cons } xbc \Gamma$)
 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 toSet.simps Un-iff set-GConsD wfG-cons2
    by (metis (full-types) Un-iff set-GConsD toSet.simps(2) wfG-cons2)
qed
qed

```

```

lemma lookup-iff:
  fixes  $\Theta::\Theta$  and  $\Gamma::\Gamma$ 
  assumes  $\Theta; \mathcal{B} \vdash_{wf} \Gamma$ 
  shows  $Some (b,c) = lookup \ \Gamma \ x \longleftrightarrow (x,b,c) \in toSet \ \Gamma \wedge (\forall b' \ c'. (x,b',c') \in toSet \ \Gamma \longrightarrow b'=b \wedge c'=c)$ 
  using assms lookup-if1 lookup-if2 by meson

```

```

lemma wfG-lookup-wf:
  fixes  $\Theta::\Theta$  and  $\Gamma::\Gamma$  and  $b::b$  and  $\mathcal{B}::\mathcal{B}$ 
  assumes  $\Theta; \mathcal{B} \vdash_{wf} \Gamma$  and  $Some (b,c) = lookup \ \Gamma \ x$ 
  shows  $\Theta; \mathcal{B} \vdash_{wf} b$ 
using assms proof(induct  $\Gamma$  rule:  $\Gamma$ -induct)
  case GNil
  then show ?case by auto
next
  case ( $GCons \ x' \ b' \ c' \ \Gamma'$ )
  then show ?case proof(cases  $x=x'$ )
    case True
    then show ?thesis using lookup.simps wfG-elim(2) GCons by fastforce
  next
    case False
    then show ?thesis using lookup.simps wfG-elim(2) GCons by fastforce
  qed
qed

```

```

lemma wfG-unique:
  fixes  $\Gamma::\Gamma$ 
  assumes  $wfG \ B \ \Theta \ ((x, b, c) \#_{\Gamma} \Gamma)$  and  $(x1,b1,c1) \in toSet \ ((x, b, c) \#_{\Gamma} \Gamma)$  and  $x1=x$ 
  shows  $b1 = b \wedge c1 = c$ 
proof -
  have  $(x, b, c) \in toSet \ ((x, b, c) \#_{\Gamma} \Gamma)$  by simp
  thus ?thesis using wf-g-unique assms by blast
qed

```

```

lemma wfG-unique-full:
  fixes  $\Gamma::\Gamma$ 
  assumes  $wfG \ \Theta \ B \ (\Gamma'@ (x, b, c) \#_{\Gamma} \Gamma)$  and  $(x1,b1,c1) \in toSet \ (\Gamma'@ (x, b, c) \#_{\Gamma} \Gamma)$  and  $x1=x$ 
  shows  $b1 = b \wedge c1 = c$ 
proof -
  have  $(x, b, c) \in toSet \ (\Gamma'@ (x, b, c) \#_{\Gamma} \Gamma)$  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 $\Gamma::\Gamma$ **and** $\Gamma':\Gamma$ **and** $v::v$ **and** $e::e$ **and** $c::c$ **and** $\tau::\tau$ **and** $ts::(\text{string}*\tau)$ *list* **and** $\Delta::\Delta$ **and** $s::s$ **and** $b::b$ **and** $ftq::\text{fun-ty}p-q$ **and** $ft::\text{fun-ty}p$ **and** $ce::ce$ **and** $td::\text{type-def}$ **and** $cs::\text{branch-s}$ **and** $css::\text{branch-list}$

shows $wfV\text{-}wf: \Theta; \mathcal{B}; \Gamma \vdash_{wf} v : b \implies \Theta; \mathcal{B} \vdash_{wf} \Gamma \wedge \vdash_{wf} \Theta$ **and**
 $wfC\text{-}wf: \Theta; \mathcal{B}; \Gamma \vdash_{wf} c \implies \Theta; \mathcal{B} \vdash_{wf} \Gamma \wedge \vdash_{wf} \Theta$ **and**
 $wfG\text{-}wf: \Theta; \mathcal{B} \vdash_{wf} \Gamma \implies \vdash_{wf} \Theta$ **and**
 $wfT\text{-}wf: \Theta; \mathcal{B}; \Gamma \vdash_{wf} \tau \implies \Theta; \mathcal{B} \vdash_{wf} \Gamma \wedge \vdash_{wf} \Theta \wedge \Theta; \mathcal{B} \vdash_{wf} b\text{-of } \tau$ **and**
 $wfTs\text{-}wf: \Theta; \mathcal{B}; \Gamma \vdash_{wf} ts \implies \Theta; \mathcal{B} \vdash_{wf} \Gamma \wedge \vdash_{wf} \Theta$ **and**
 $\vdash_{wf} \Theta \implies \text{True}$ **and**
 $wfB\text{-}wf: \Theta; \mathcal{B} \vdash_{wf} b \implies \vdash_{wf} \Theta$ **and**
 $wfCE\text{-}wf: \Theta; \mathcal{B}; \Gamma \vdash_{wf} ce : b \implies \Theta; \mathcal{B} \vdash_{wf} \Gamma \wedge \vdash_{wf} \Theta$ **and**
 $wfTD\text{-}wf: \Theta \vdash_{wf} td \implies \vdash_{wf} \Theta$

proof(*induct rule:wfV-wfC-wfG-wfT-wfTs-wfTh-wfB-wfCE-wfTD.inducts*)

case ($wfV\text{-}varI \ \Theta \ \mathcal{B} \ \Gamma \ b \ c \ x$)
hence $(x, b, c) \in \text{toSet } \Gamma$ **using** *lookup-iff lookup-in-g by presburger*
hence $b \in \text{fst'snd'toSet } \Gamma$ **by force**
hence $wfB \ \Theta \ \mathcal{B} \ b$ **using** *wfV-varI using wfG-lookup-wf by auto*
then show $?case$ **using** *wfV-varI wfV-elim wf-intros by metis*
next
case ($wfV\text{-}litI \ \Theta \ \mathcal{B} \ \Gamma \ l$)
moreover have $wfTh \ \Theta$ **using** *wfV-litI by metis*
ultimately show $?case$ **using** *wf-intros base-for-lit.simps l.exhaust by metis*
next
case ($wfV\text{-}pairI \ \Theta \ \mathcal{B} \ \Gamma \ v1 \ b1 \ v2 \ b2$)
then show $?case$ **using** *wfB-pairI by simp*
next
case ($wfV\text{-}consI \ s \ dclist \ \Theta \ dc \ x \ b \ c \ \mathcal{B} \ \Gamma \ v$)
then show $?case$ **using** *wf-intros by metis*
next
case ($wfTI \ z \ \Gamma \ \Theta \ \mathcal{B} \ b \ c$)
then show $?case$ **using** *wf-intros b-of.simps wfG-cons2 by metis*
qed(*auto*)

lemma *wfX-wfY2*:

fixes $\Gamma::\Gamma$ **and** $\Gamma':\Gamma$ **and** $v::v$ **and** $e::e$ **and** $c::c$ **and** $\tau::\tau$ **and** $ts::(\text{string}*\tau)$ *list* **and** $\Delta::\Delta$ **and** $s::s$ **and** $b::b$ **and** $ftq::\text{fun-ty}p-q$ **and** $ft::\text{fun-ty}p$ **and** $ce::ce$ **and** $td::\text{type-def}$ **and** $cs::\text{branch-s}$ **and** $css::\text{branch-list}$

shows

$wfE\text{-}wf: \Theta; \Phi; \mathcal{B}; \Gamma; \Delta \vdash_{wf} e : b \implies \Theta; \mathcal{B} \vdash_{wf} \Gamma \wedge \Theta; \mathcal{B}; \Gamma \vdash_{wf} \Delta \wedge \vdash_{wf} \Theta \wedge \Theta \vdash_{wf} \Phi$ **and**
 $wfS\text{-}wf: \Theta; \Phi; \mathcal{B}; \Gamma; \Delta \vdash_{wf} s : b \implies \Theta; \mathcal{B} \vdash_{wf} \Gamma \wedge \Theta; \mathcal{B}; \Gamma \vdash_{wf} \Delta \wedge \vdash_{wf} \Theta \wedge \Theta \vdash_{wf} \Phi$ **and**
 $\Theta; \Phi; \mathcal{B}; \Gamma; \Delta; tid; dc; t \vdash_{wf} cs : b \implies \Theta; \mathcal{B} \vdash_{wf} \Gamma \wedge \Theta; \mathcal{B}; \Gamma \vdash_{wf} \Delta \wedge \vdash_{wf} \Theta \wedge \Theta \vdash_{wf} \Phi$ **and**
 $\Theta; \Phi; \mathcal{B}; \Gamma; \Delta; tid; dclist \vdash_{wf} css : b \implies \Theta; \mathcal{B} \vdash_{wf} \Gamma \wedge \Theta; \mathcal{B}; \Gamma \vdash_{wf} \Delta \wedge \vdash_{wf} \Theta \wedge \Theta \vdash_{wf} \Phi$ **and**
 $wfPhi\text{-}wf: \Theta \vdash_{wf} (\Phi::\Phi) \implies \vdash_{wf} \Theta$ **and**

$wfD\text{-}wf: \Theta; \mathcal{B}; \Gamma \vdash_{wf} \Delta \implies \Theta; \mathcal{B} \vdash_{wf} \Gamma \wedge \vdash_{wf} \Theta$ **and**
 $wfFTQ\text{-}wf: \Theta; \Phi \vdash_{wf} ftq \implies \Theta \vdash_{wf} \Phi \wedge \vdash_{wf} \Theta$ **and**
 $wfFT\text{-}wf: \Theta; \Phi; \mathcal{B} \vdash_{wf} ft \implies \Theta \vdash_{wf} \Phi \wedge \vdash_{wf} \Theta$

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-elim* **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-elim DCons setD-ConsD*
 by (*metis case-prodI2 set-ConsD*)
qed

lemma *subst-b-lookup-d*:
 assumes $u \notin fst \text{ ' } setD \Delta$
 shows $u \notin fst \text{ ' } setD \Delta[bv::=b]_{\Delta b}$
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 $\Theta; \mathcal{B} \vdash_{wf} \Gamma$ and $atom\ x \# \Gamma$ and $wfB\ \Theta\ \mathcal{B}\ b$ 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 $\Theta; \mathcal{B} \vdash_{wf} ((x, b, c) \#_{\Gamma} \Gamma)$
 using wfG-cons1I wfG-cons2I assms by metis

lemma wfG-consI:
 fixes $\Phi::\Phi$ and $\Gamma::\Gamma$ and $c::c$
 assumes $\Theta; \mathcal{B} \vdash_{wf} \Gamma$ and $atom\ x \# \Gamma$ and $wfB\ \Theta\ \mathcal{B}\ b$ and
 $\Theta ; \mathcal{B} ; (x, b, C\text{-true}) \#_{\Gamma} \Gamma \vdash_{wf} c$
 shows $\Theta ; \mathcal{B} \vdash_{wf} ((x, b, c) \#_{\Gamma} \Gamma)$
 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 $P; \mathcal{B} ; (x, b, TRUE) \#_{\Gamma} \Gamma \vdash_{wf} c \wedge wfB\ P\ \mathcal{B}\ b$
proof(cases $c \in \{ TRUE, FALSE \}$)
 case True
 have $P; \mathcal{B} \vdash_{wf} \Gamma \wedge atom\ x \# \Gamma \wedge wfB\ P\ \mathcal{B}\ b$ using wfG-elim(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-elim(2)[OF assms] by auto
 qed

lemma wfG-cons-wfC:
 fixes $\Gamma::\Gamma$ and $c::c$
 assumes $\Theta ; B \vdash_{wf} (x, b, c) \#_{\Gamma} \Gamma$
 shows $\Theta ; B ; ((x, b, TRUE) \#_{\Gamma} \Gamma) \vdash_{wf} c$
 using assms wfG-elim2 by auto

lemma wfG-wfB:
 assumes $wfG\ P\ \mathcal{B}\ \Gamma$ and $b \in fst'snd'toSet\ \Gamma$
 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  $b \in \text{fst}'\text{snd}'\text{toSet } \Gamma'$  using GCons by auto
  moreover have  $\text{wfG } P \ \mathcal{B} \ \Gamma'$  using wfG-cons GCons by auto
  ultimately show ?thesis using GCons by auto
qed
qed

```

```

lemma wfG-cons-TRUE:
  fixes  $\Gamma::\Gamma$  and  $b::b$ 
  assumes  $P; \mathcal{B} \vdash_{wf} \Gamma$  and  $\text{atom } z \# \Gamma$  and  $P; \mathcal{B} \vdash_{wf} b$ 
  shows  $P; \mathcal{B} \vdash_{wf} (z, b, \text{TRUE}) \#_{\Gamma} \Gamma$ 
  using wfG-cons2I wfG-wfB assms by simp

```

```

lemma wfG-cons-TRUE2:
  assumes  $P; \mathcal{B} \vdash_{wf} (z, b, c) \#_{\Gamma} \Gamma$  and  $\text{atom } z \# \Gamma$ 
  shows  $P; \mathcal{B} \vdash_{wf} (z, b, \text{TRUE}) \#_{\Gamma} \Gamma$ 
  using wfG-cons wfG-cons2I assms by simp

```

```

lemma wfG-suffix:
  fixes  $\Gamma::\Gamma$ 
  assumes  $\text{wfG } P \ \mathcal{B} \ (\Gamma' @ \Gamma)$ 
  shows  $\text{wfG } P \ \mathcal{B} \ \Gamma$ 
using assms proof(induct  $\Gamma'$  rule:  $\Gamma$ -induct)
  case GNil
  then show ?case by auto
next
  case ( $GCons \ x \ b \ c \ \Gamma'$ )
  hence  $P; \mathcal{B} \vdash_{wf} \Gamma' @ \Gamma$  using wfG-elim by auto
  then show ?case using GCons wfG-elim by auto
qed

```

```

lemma wfV-wfCE:
  fixes  $v::v$ 
  assumes  $\Theta; \mathcal{B}; \Gamma \vdash_{wf} v : b$ 
  shows  $\Theta; \mathcal{B}; \Gamma \vdash_{wf} \text{CE-val } v : b$ 
proof -
  have  $\Theta \vdash_{wf} ([::\Phi])$  using wfPhi-emptyI wfV-wf wfG-wf assms by metis
  moreover have  $\Theta; \mathcal{B}; \Gamma \vdash_{wf} []_{\Delta}$  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-suppl1:
  fixes  $\Gamma::\Gamma$  and  $\Gamma':\Gamma$  and  $v::v$  and  $e::e$  and  $c::c$  and  $\tau::\tau$  and  $ts::(\text{string}*\tau)$  list and  $\Delta::\Delta$  and
 $s::s$  and  $b::b$  and  $ftq::\text{fun-ty-p-q}$  and  $ft::\text{fun-ty-p}$  and  $ce::ce$  and  $td::\text{type-def}$  and  $cs::\text{branch-s}$  and  $css$ 
 $::\text{branch-list}$ 

```

```

  shows  $\text{wfV-suppl: } \Theta; \mathcal{B}; \Gamma \vdash_{wf} v : b \implies \text{supp } v \subseteq \text{atom-dom } \Gamma \cup \text{supp } \mathcal{B}$  and
 $\text{wfC-suppl: } \Theta; \mathcal{B}; \Gamma \vdash_{wf} c \implies \text{supp } c \subseteq \text{atom-dom } \Gamma \cup \text{supp } \mathcal{B}$  and

```

$wfG\text{-supp}: \Theta; \mathcal{B} \vdash_{wf} \Gamma \implies \text{atom-dom } \Gamma \subseteq \text{supp } \Gamma$ **and**
 $wfT\text{-supp}: \Theta; \mathcal{B}; \Gamma \vdash_{wf} \tau \implies \text{supp } \tau \subseteq \text{atom-dom } \Gamma \cup \text{supp } \mathcal{B}$ **and**
 $wfTs\text{-supp}: \Theta; \mathcal{B}; \Gamma \vdash_{wf} ts \implies \text{supp } ts \subseteq \text{atom-dom } \Gamma \cup \text{supp } \mathcal{B}$ **and**
 $wfTh\text{-supp}: \vdash_{wf} \Theta \implies \text{supp } \Theta = \{\}$ **and**
 $wfB\text{-supp}: \Theta; \mathcal{B} \vdash_{wf} b \implies \text{supp } b \subseteq \text{supp } \mathcal{B}$ **and**
 $wfCE\text{-supp}: \Theta; \mathcal{B}; \Gamma \vdash_{wf} ce : b \implies \text{supp } ce \subseteq \text{atom-dom } \Gamma \cup \text{supp } \mathcal{B}$ **and**
 $wfTD\text{-supp}: \Theta \vdash_{wf} td \implies \text{supp } td \subseteq \{\}$

proof(*induct rule:wfV-wfC-wfG-wfT-wfTs-wfTh-wfB-wfCE-wfTD.inducts*)

case ($wfB\text{-consI } \Theta \ s \ \text{dclist } \mathcal{B}$)

then show $?case$ **by**(*auto simp add: b.supp pure-supp*)

next

case ($wfB\text{-appI } \Theta \ \mathcal{B} \ b \ s \ \text{bv dclist}$)

then show $?case$ **by**(*auto simp add: b.supp pure-supp*)

next

case ($wfV\text{-varI } \Theta \ \mathcal{B} \ \Gamma \ b \ c \ x$)

then show $?case$ **using** *v.supp wfV-elim*
 empty-subsetI insert-subset supp-at-base
 fresh-dom-free2 lookup-if1
 by (*metis sup.coboundedI1*)

next

case ($wfV\text{-litI } \Theta \ \mathcal{B} \ \Gamma \ l$)

then show $?case$ **using** *supp-l-empty v.supp* **by** *simp*

next

case ($wfV\text{-pairI } \Theta \ \mathcal{B} \ \Gamma \ v1 \ b1 \ v2 \ b2$)

then show $?case$ **using** *v.supp wfV-elim* **by** (*metis Un-subset-iff*)

next

case ($wfV\text{-consI } s \ \text{dclist } \Theta \ dc \ x \ b \ c \ \mathcal{B} \ \Gamma \ v$)

then show $?case$ **using** *v.supp wfV-elim*
 Un-commute b.supp sup-bot.right-neutral supp-b-empty pure-supp **by** *metis*

next

case ($wfV\text{-consPI typid bv dclist } \Theta \ dc \ x \ b' \ c \ \mathcal{B} \ \Gamma \ v \ b$)

then show $?case$ **unfolding** *v.supp*
 using *wfV-elim*
 Un-commute b.supp sup-bot.right-neutral supp-b-empty pure-supp
 by (*simp add: Un-commute pure-supp sup.coboundedI1*)

next

case ($wfC\text{-eqI } \Theta \ \mathcal{B} \ \Gamma \ e1 \ b \ e2$)

hence $\text{supp } e1 \subseteq \text{atom-dom } \Gamma \cup \text{supp } \mathcal{B}$ **using** *c.supp wfC-elim*
 image-empty list.set(1) sup-bot.right-neutral **by** (*metis IntI UnE empty-iff subsetCE subsetI*)

moreover have $\text{supp } e2 \subseteq \text{atom-dom } \Gamma \cup \text{supp } \mathcal{B}$ **using** *c.supp wfC-elim*
 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 ($wfG\text{-cons1I } c \ \Theta \ \mathcal{B} \ \Gamma \ x \ b$)

then show $?case$ **using** *atom-dom.simps dom-supp-g supp-GCons* **by** *metis*

next

case ($wfG\text{-cons2I } c \ \Theta \ \mathcal{B} \ \Gamma \ x \ b$)

then show $?case$ **using** *atom-dom.simps dom-supp-g supp-GCons* **by** *metis*

next

case $wfTh\text{-emptyI}$

```

    then show ?case by (simp add: supp-Nil)
next
  case (wfTh-consI  $\Theta$  lst)
  then show ?case using supp-Cons by fast
next
  case (wfTD-simpleI  $\Theta$  lst s)
  then have supp (AF-typedef s lst) = supp lst  $\cup$  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  $\Theta$  bv lst s)
  then have supp (AF-typedef-poly s bv lst) = supp lst - { atom bv }  $\cup$  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  $\Theta$   $\mathcal{B}$   $\Gamma$ )
  then show ?case using supp-Nil by auto
next
  case (wfTs-cons  $\Theta$   $\mathcal{B}$   $\Gamma$   $\tau$  dc ts)
  then show ?case using supp-Cons supp-Pair pure-supp[of dc] by blast
next
  case (wfCE-valI  $\Theta$   $\mathcal{B}$   $\Gamma$  v b)
  thus ?case using ce.supp wfCE-elim by simp
next
  case (wfCE-plusI  $\Theta$   $\mathcal{B}$   $\Gamma$  v1 v2)
  hence supp (CE-op Plus v1 v2)  $\subseteq$  atom-dom  $\Gamma \cup$  supp  $\mathcal{B}$  using ce.supp pure-supp
    by (simp add: wfCE-plusI opp.supp)
  then show ?case using ce.supp wfCE-elim UnCI subsetCE subsetI x-not-in-b-set by auto
next
  case (wfCE-leqI  $\Theta$   $\mathcal{B}$   $\Gamma$  v1 v2)
  hence supp (CE-op LEq v1 v2)  $\subseteq$  atom-dom  $\Gamma \cup$  supp  $\mathcal{B}$  using ce.supp pure-supp
    by (simp add: wfCE-plusI opp.supp)
  then show ?case using ce.supp wfE-elim UnCI subsetCE subsetI x-not-in-b-set by auto
next
  case (wfCE-fstI  $\Theta$   $\mathcal{B}$   $\Gamma$  v1 b1 b2)
  thus ?case using ce.supp wfCE-elim by simp
next
  case (wfCE-sndI  $\Theta$   $\mathcal{B}$   $\Gamma$  v1 b1 b2)
  thus ?case using ce.supp wfCE-elim by simp
next
  case (wfCE-concatI  $\Theta$   $\mathcal{B}$   $\Gamma$  v1 v2)
  thus ?case using ce.supp wfCE-elim by simp
next
  case (wfCE-lenI  $\Theta$   $\mathcal{B}$   $\Gamma$  v1)
  thus ?case using ce.supp wfCE-elim by simp
next
  case (wfTI z  $\Theta$   $\mathcal{B}$   $\Gamma$  b c)
  hence supp c  $\subseteq$  supp z  $\cup$  atom-dom  $\Gamma \cup$  supp  $\mathcal{B}$  using supp-at-base dom-cons by metis
  moreover have supp b  $\subseteq$  supp  $\mathcal{B}$  using wfTI by auto
  ultimately have supp { z : b | c }  $\subseteq$  atom-dom  $\Gamma \cup$  supp  $\mathcal{B}$  using  $\tau$ .supp supp-at-base by force
  thus ?case by auto

```

qed(auto)

lemma wf-suppl2:

fixes $\Gamma::\Gamma$ and $\Gamma':\Gamma$ and $v::v$ and $e::e$ and $c::c$ and $\tau::\tau$ and
 $ts::(\text{string}*\tau)$ list and $\Delta::\Delta$ and $s::s$ and $b::b$ and $ftq::\text{fun-ty-p-q}$ and
 $ft::\text{fun-ty-p}$ and $ce::ce$ and $td::\text{type-def}$ and $cs::\text{branch-s}$ and $css::\text{branch-list}$

shows

$wfE\text{-suppl}: \Theta; \Phi; \mathcal{B}; \Gamma; \Delta \vdash_{wf} e : b \implies (\text{suppl } e \subseteq \text{atom-dom } \Gamma \cup \text{suppl } \mathcal{B} \cup \text{atom } 'fst' \text{ setD } \Delta)$ and

$wfS\text{-suppl}: \Theta; \Phi; \mathcal{B}; \Gamma; \Delta \vdash_{wf} s : b \implies \text{suppl } s \subseteq \text{atom-dom } \Gamma \cup \text{atom } 'fst' \text{ setD } \Delta \cup \text{suppl } \mathcal{B}$
and

$\Theta; \Phi; \mathcal{B}; \Gamma; \Delta; tid; dc; t \vdash_{wf} cs : b \implies \text{suppl } cs \subseteq \text{atom-dom } \Gamma \cup \text{atom } 'fst' \text{ setD } \Delta \cup \text{suppl } \mathcal{B}$ and

$\Theta; \Phi; \mathcal{B}; \Gamma; \Delta; tid; dclist \vdash_{wf} css : b \implies \text{suppl } css \subseteq \text{atom-dom } \Gamma \cup \text{atom } 'fst' \text{ setD } \Delta \cup \text{suppl } \mathcal{B}$ and

$wfPhi\text{-suppl}: \Theta \vdash_{wf} (\Phi::\Phi) \implies \text{suppl } \Phi = \{\}$ and

$wfD\text{-suppl}: \Theta; \mathcal{B}; \Gamma \vdash_{wf} \Delta \implies \text{suppl } \Delta \subseteq \text{atom } 'fst' (\text{setD } \Delta) \cup \text{atom-dom } \Gamma \cup \text{suppl } \mathcal{B}$ and

$\Theta; \Phi \vdash_{wf} ftq \implies \text{suppl } ftq = \{\}$ and

$\Theta; \Phi; \mathcal{B} \vdash_{wf} ft \implies \text{suppl } ft \subseteq \text{suppl } \mathcal{B}$

proof(induct rule:wfE-wfS-wfCS-wfCSS-wfPhi-wfD-wfFTQ-wfFT.inducts)

case (wfE-valI $\Theta \Phi \mathcal{B} \Gamma \Delta v b$)

hence $\text{suppl } (AE\text{-val } v) \subseteq \text{atom-dom } \Gamma \cup \text{suppl } \mathcal{B}$ using $e.\text{suppl } wf\text{-suppl1}$ by simp

then show ?case using $e.\text{suppl } wfE\text{-elims } UnCI \text{ subsetCE } \text{subsetI } x\text{-not-in-b-set}$ by metis

next

case (wfE-plusI $\Theta \Phi \mathcal{B} \Gamma \Delta v1 v2$)

hence $\text{suppl } (AE\text{-op } Plus \ v1 \ v2) \subseteq \text{atom-dom } \Gamma \cup \text{suppl } \mathcal{B}$

using $wfE\text{-plusI } opp.\text{suppl } wf\text{-suppl1 } e.\text{suppl } pure\text{-suppl } Un\text{-least}$

by (metis sup-bot.left-neutral)

then show ?case using $e.\text{suppl } wfE\text{-elims } UnCI \text{ subsetCE } \text{subsetI } x\text{-not-in-b-set}$ by auto

next

case (wfE-leqI $\Theta \Phi \mathcal{B} \Gamma \Delta v1 v2$)

hence $\text{suppl } (AE\text{-op } LEq \ v1 \ v2) \subseteq \text{atom-dom } \Gamma \cup \text{suppl } \mathcal{B}$ using $e.\text{suppl } pure\text{-suppl } Un\text{-least}$
 $sup\text{-bot.left-neutral}$ using $opp.\text{suppl } wf\text{-suppl1}$ by auto

then show ?case using $e.\text{suppl } wfE\text{-elims } UnCI \text{ subsetCE } \text{subsetI } x\text{-not-in-b-set}$ by auto

next

case (wfE-fstI $\Theta \Phi \mathcal{B} \Gamma \Delta v1 b1 b2$)

hence $\text{suppl } (AE\text{-fst } v1) \subseteq \text{atom-dom } \Gamma \cup \text{suppl } \mathcal{B}$ using $e.\text{suppl } pure\text{-suppl } sup\text{-bot.left-neutral}$
using $opp.\text{suppl } wf\text{-suppl1}$ by auto

then show ?case using $e.\text{suppl } wfE\text{-elims } UnCI \text{ subsetCE } \text{subsetI } x\text{-not-in-b-set}$ by auto

next

case (wfE-sndI $\Theta \Phi \mathcal{B} \Gamma \Delta v1 b1 b2$)

hence $\text{suppl } (AE\text{-snd } v1) \subseteq \text{atom-dom } \Gamma \cup \text{suppl } \mathcal{B}$ using $e.\text{suppl } pure\text{-suppl } wfE\text{-plusI } opp.\text{suppl } wf\text{-suppl1}$ by (metis Un-least)

then show ?case using $e.\text{suppl } wfE\text{-elims } UnCI \text{ subsetCE } \text{subsetI } x\text{-not-in-b-set}$ by auto

next

case (wfE-concatI $\Theta \Phi \mathcal{B} \Gamma \Delta v1 v2$)

hence $\text{suppl } (AE\text{-concat } v1 \ v2) \subseteq \text{atom-dom } \Gamma \cup \text{suppl } \mathcal{B}$ using $e.\text{suppl } pure\text{-suppl } wfE\text{-plusI } opp.\text{suppl } wf\text{-suppl1}$ by (metis Un-least)

then show ?case using $e.\text{suppl } wfE\text{-elims } UnCI \text{ subsetCE } \text{subsetI } x\text{-not-in-b-set}$ by auto

next

case (wfE-splitI $\Theta \Phi \mathcal{B} \Gamma \Delta v1 v2$)

hence $\text{supp } (AE\text{-split } v1 \ v2) \subseteq \text{atom-dom } \Gamma \cup \text{supp } \mathcal{B}$ **using** $e.\text{supp pure-supp}$
 $\text{wfE-plusI opp.supp wf-suppl1}$ **by** (metis Un-least)
 then show $?case$ **using** $e.\text{supp wfE-elim UnCI subsetCE subsetI x-not-in-b-set}$ **by** *auto*
 next
 case $(\text{wfE-lenI } \Theta \ \Phi \ \mathcal{B} \ \Gamma \ \Delta \ v1)$
 hence $\text{supp } (AE\text{-len } v1) \subseteq \text{atom-dom } \Gamma \cup \text{supp } \mathcal{B}$ **using** $e.\text{supp pure-supp}$
using $e.\text{supp pure-supp sup-bot.left-neutral}$ **using** $\text{opp.supp wf-suppl1}$ **by** *auto*
 then show $?case$ **using** $e.\text{supp wfE-elim UnCI subsetCE subsetI x-not-in-b-set}$ **by** *auto*
 next
 case $(\text{wfE-appI } \Theta \ \Phi \ \mathcal{B} \ \Gamma \ \Delta \ f \ x \ b \ c \ \tau \ s \ v)$
 then obtain b where $\Theta; \mathcal{B}; \Gamma \vdash_{wf} v : b$ **using** wfE-elim **by** *metis*
 hence $\text{supp } v \subseteq \text{atom-dom } \Gamma \cup \text{supp } \mathcal{B}$ **using** $\text{wfE-appI wf-suppl1}$ **by** *metis*
 hence $\text{supp } (AE\text{-app } f \ v) \subseteq \text{atom-dom } \Gamma \cup \text{supp } \mathcal{B}$ **using** $e.\text{supp pure-supp}$ **by** *fast*
 then show $?case$ **using** $e.\text{supp}(2) \ \text{UnCI subsetCE subsetI wfE-appI}$ **using** $b.\text{supp}(3) \ \text{pure-supp}$
 $x\text{-not-in-b-set}$ **by** *auto*
 next
 case $(\text{wfE-appPI } \Theta \ \Phi \ \mathcal{B} \ \Gamma \ \Delta \ b' \ bv \ v \ \tau \ f \ xa \ ba \ ca \ s)$
 then obtain b where $\Theta; \mathcal{B}; \Gamma \vdash_{wf} v : (b[bv::=b])_b$ **using** wfE-elim **by** *metis*
 hence $\text{supp } v \subseteq \text{atom-dom } \Gamma \cup \text{supp } \mathcal{B}$ **using** $\text{wfE-appPI wf-suppl1}$ **by** *auto*
 moreover have $\text{supp } b' \subseteq \text{supp } \mathcal{B}$ **using** $\text{wf-suppl1}(7) \ \text{wfE-appPI}$ **by** *simp*
 ultimately show $?case$ **unfolding** $e.\text{supp}$ **using** $\text{wfE-appPI pure-supp}$ **by** *fast*
 next
 case $(\text{wfE-mvarI } \Theta \ \Phi \ \mathcal{B} \ \Gamma \ \Delta \ u \ \tau)$
 then obtain τ where $(u, \tau) \in \text{setD } \Delta$ **using** $\text{wfE-elim}(10)$ **by** *metis*
 hence $\text{atom } u \in \text{atom}'\text{fst}'\text{setD } \Delta$ **by** *force*
 hence $\text{supp } (AE\text{-mvar } u) \subseteq \text{atom}'\text{fst}'\text{setD } \Delta$ **using** $e.\text{supp}$
by $(\text{simp add: supp-at-base})$
 thus $?case$ **using** $\text{UnCI subsetCE subsetI e.supp wfE-mvarI supp-at-base subsetCE supp-at-base u-not-in-b-set}$
by $(\text{simp add: supp-at-base})$
 next
 case $(\text{wfS-valI } \Theta \ \Phi \ \mathcal{B} \ \Gamma \ v \ b \ \Delta)$
 then show $?case$ **using** wf-suppl1
by $(\text{metis s-branch-s-branch-list.supp}(1) \ \text{sup.coboundedI2 sup-assoc sup-commute})$
 next
 case $(\text{wfS-letI } \Theta \ \Phi \ \mathcal{B} \ \Gamma \ \Delta \ e \ b' \ x \ s \ b)$
 then show $?case$ **by** *auto*
 next
 case $(\text{wfS-let2I } \Theta \ \Phi \ \mathcal{B} \ \Gamma \ \Delta \ s1 \ \tau \ x \ s2 \ b)$
 then show $?case$ **unfolding** $\text{s-branch-s-branch-list.supp } (3)$ **using** $\text{wf-suppl1}(4)[OF \ \text{wfS-let2I}(3)]$ **by** *auto*
 next
 case $(\text{wfS-ifI } \Theta \ \Phi \ \mathcal{B} \ \Gamma \ v \ \Phi \ \Delta \ s1 \ b \ s2)$
 then show $?case$ **using** $\text{wf-suppl1}(1)[OF \ \text{wfS-ifI}(1)]$ **by** *auto*
 next
 case $(\text{wfS-varI } \Theta \ \mathcal{B} \ \Gamma \ \tau \ v \ u \ \Delta \ \Phi \ s \ b)$
 then show $?case$ **using** $\text{wf-suppl1}(1)[OF \ \text{wfS-varI}(2)] \ \text{wf-suppl1}(4)[OF \ \text{wfS-varI}(1)]$ **by** *auto*
 next
 next
 case $(\text{wfS-assignI } u \ \tau \ \Delta \ \Theta \ \mathcal{B} \ \Gamma \ \Phi \ v)$
 hence $\text{supp } u \subseteq \text{atom}' \ \text{fst}' \ \text{setD } \Delta$ **proof** $(\text{induct } \Delta \ \text{rule:}\Delta\text{-induct})$
 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 toSet.simps DCons supp-at-base by fastforce
next
case False
then show ?thesis using toSet.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-suppl(1)[OF wfS-assignI(6)]
by auto
next
case (wfS-matchI Θ B Γ v tid dclist Δ Φ cs b)
then show ?case using wf-suppl(1)[OF wfS-matchI(1)] by auto
next
case (wfS-branchI Θ Φ B x τ Γ Δ s b tid dc)
moreover have supp s ⊆ supp x ∪ atom-dom Γ ∪ atom 'fst 'setD Δ ∪ supp 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 B using
supp-at-base by force
ultimately have
  (supp s - set [atom x]) ∪ (supp dc) ⊆ atom-dom Γ ∪ atom 'fst 'setD Δ ∪ supp B
  by (simp add: pure-suppl)
thus ?case using s-branch-s-branch-list.supp(2) by auto
next
case (wfD-emptyI Θ B Γ)
then show ?case using supp-DNil by auto
next
case (wfD-cons Θ B Γ Δ τ u)
have supp ((u, τ) #Δ 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 B
  using wfD-cons wf-suppl(4)[OF wfD-cons(3)] by auto
also have ... ⊆ atom 'fst 'setD ((u, τ) #Δ Δ) ∪ atom-dom Γ ∪ supp 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-suppl supp-Cons)
next
case (wfFTI Θ B' b s x c τ Φ)
thm fun-typ.supp
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-suppl
proof -
  have f1: supp τ ⊆ {atom x} ∪ atom-dom GNil ∪ supp B'

```

```

    using dom-cons wfFTI.hyps wf-suppl(4) by blast
  have  $\text{supp } b \subseteq \text{supp } B'$ 
    using wfFTI.hyps(1) wf-suppl(7) by blast
  then show ?thesis
    using f1  $\langle \text{supp } (AF\text{-fun-ty}p \ x \ b \ c \ \tau \ s) = \text{supp } c \cup (\text{supp } \tau \cup \text{supp } s) - \text{set } [\text{atom } x] \cup \text{supp } b \rangle$ 
      wfFTI.hyps(4) wfFTI.hyps by auto
qed
next
case (wfFTNone  $\Theta \ \Phi \ ft$ )
then show ?case by (simp add: fun-tyq-suppl(2))
next
case (wfFTSome  $\Theta \ \Phi \ bv \ ft$ )
then show ?case using fun-tyq-suppl
  by (simp add: suppl-at-base)
next
case (wfS-assertI  $\Theta \ \Phi \ \mathcal{B} \ x \ c \ \Gamma \ \Delta \ s \ b$ )
then have  $\text{supp } c \subseteq \text{atom-dom } \Gamma \cup \text{atom } 'fst' \ 'setD' \ \Delta \cup \text{supp } \mathcal{B}$  using wf-suppl1
  by (metis Un-assoc Un-commute le-supI2)
moreover have  $\text{supp } s \subseteq \text{atom-dom } \Gamma \cup \text{atom } 'fst' \ 'setD' \ \Delta \cup \text{supp } \mathcal{B}$  proof
  fix z
  assume *:  $z \in \text{supp } s$ 
  have *:  $\text{atom } x \notin \text{supp } s$  using wfS-assertI fresh-prodN fresh-def by metis
  have  $z \in \text{atom-dom } ((x, B\text{-bool}, c) \#_{\Gamma} \Gamma) \cup \text{atom } 'fst' \ 'setD' \ \Delta \cup \text{supp } \mathcal{B}$  using wfS-assertI * by
blast
  have  $z \in \text{atom-dom } ((x, B\text{-bool}, c) \#_{\Gamma} \Gamma) \implies z \in \text{atom-dom } \Gamma$  using * by auto
  thus  $z \in \text{atom-dom } \Gamma \cup \text{atom } 'fst' \ 'setD' \ \Delta \cup \text{supp } \mathcal{B}$  using * by
    using  $\langle z \in \text{atom-dom } ((x, B\text{-bool}, c) \#_{\Gamma} \Gamma) \cup \text{atom } 'fst' \ 'setD' \ \Delta \cup \text{supp } \mathcal{B} \rangle$  by blast
qed
ultimately show ?case by auto
qed(auto)

lemmas wf-suppl = wf-suppl1 wf-suppl2

lemma wfV-suppl-nil:
  fixes  $v::v$ 
  assumes  $P ; \{|\}\} ; GNil \vdash_{wf} v : b$ 
  shows  $\text{supp } v = \{\}$ 
  using wfV-suppl[of  $P \ \{|\}\} \ GNil \ v \ b$ ] dom.simps toSet.simps
  using assms by auto

lemma wfT-TRUE-aux:
  assumes  $wfG \ P \ \mathcal{B} \ \Gamma$  and  $\text{atom } z \nmid (P, \mathcal{B}, \Gamma)$  and  $wfB \ P \ \mathcal{B} \ b$ 
  shows  $wfT \ P \ \mathcal{B} \ \Gamma \ (\llbracket z : b \mid TRUE \rrbracket)$ 
proof (rule)
  show  $\langle \text{atom } z \nmid (P, \mathcal{B}, \Gamma) \rangle$  using assms by auto
  show  $\langle P ; \mathcal{B} \vdash_{wf} b \rangle$  using assms by auto
  show  $\langle P ; \mathcal{B} ; (z, b, TRUE) \#_{\Gamma} \Gamma \vdash_{wf} TRUE \rangle$  using wfG-cons2I wfC-trueI assms by auto
qed

lemma wfT-TRUE:
  assumes  $wfG \ P \ \mathcal{B} \ \Gamma$  and  $wfB \ P \ \mathcal{B} \ b$ 
  shows  $wfT \ P \ \mathcal{B} \ \Gamma \ (\llbracket z : b \mid TRUE \rrbracket)$ 

```

proof –

obtain $z'::x$ **where** $*:atom\ z' \# (P, \mathcal{B}, \Gamma)$ **using** *obtain-fresh* **by** *metis*

hence $\{z : b \mid TRUE\} = \{z' : b \mid TRUE\}$ **by** *auto*

thus *?thesis* **using** *wfT-TRUE-aux* *assms* $*$ **by** *metis*

qed

lemma *phi-flip-eq*:

assumes *wfPhi* $T\ P$

shows $(x \leftrightarrow xa) \cdot P = P$

using *wfPhi-supp* [*OF* *assms*] *flip-fresh-fresh* *fresh-def* **by** *blast*

lemma *wfC-supp-cons*:

fixes $c'::c$ **and** $G::\Gamma$

assumes $P; \mathcal{B}; (x', b', TRUE) \#_{\Gamma} G \vdash_{wf} c'$

shows $supp\ c' \subseteq atom-dom\ G \cup supp\ x' \cup supp\ \mathcal{B}$ **and** $supp\ c' \subseteq supp\ G \cup supp\ x' \cup supp\ \mathcal{B}$

proof –

show $supp\ c' \subseteq atom-dom\ G \cup supp\ x' \cup supp\ \mathcal{B}$

using *wfC-supp* [*OF* *assms*] *dom-cons* *supp-at-base* **by** *blast*

moreover **have** $atom-dom\ G \subseteq supp\ G$

by (*meson* *assms* *wfC-wf* *wfG-cons* *wfG-supp*)

ultimately **show** $supp\ c' \subseteq supp\ G \cup supp\ x' \cup supp\ \mathcal{B}$ **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 $atom\ x \in atom-dom\ G \longleftrightarrow atom\ x \in supp\ G$

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 $(atom\ x \in atom-dom\ ((x', b', c') \#_{\Gamma} G)) = (atom\ x \in atom-dom\ G)$ **using** *atom-dom.simps*
False **by** *simp*

also **have** $\dots = (atom\ x \in supp\ G)$ **using** *GCons* *wfG-elim* **by** *metis*

also **have** $\dots = (atom\ x \in (supp\ (x', b', c') \cup supp\ G))$ **proof**

show $atom\ x \in supp\ G \implies atom\ x \in supp\ (x', b', c') \cup supp\ G$ **by** *auto*

assume $atom\ x \in supp\ (x', b', c') \cup supp\ G$

then **consider** $atom\ x \in supp\ (x', b', c') \mid atom\ x \in supp\ G$ **by** *auto*

then **show** $atom\ x \in supp\ G$ **proof**(*cases*)

case *1*

assume $atom\ x \in supp\ (x', b', c')$

hence $\text{atom } x \in \text{supp } c'$ **using** *supp-triple False supp-b-empty supp-at-base* **by** *force*
 moreover have $P; \mathcal{B}; (x', b', \text{TRUE}) \#_{\Gamma} G \vdash_{wf} c'$ **using** *wfG-elim2 GCons* **by** *simp*
 moreover hence $\text{supp } c' \subseteq \text{supp } G \cup \text{supp } x' \cup \text{supp } \mathcal{B}$ **using** *wfC-supp-cons* **by** *auto*
 ultimately have $\text{atom } x \in \text{supp } G \cup \text{supp } x'$ **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 $\dots = (\text{atom } x \in \text{supp } ((x', b', c') \#_{\Gamma} G))$ **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 $wfG \ P \ \mathcal{B} \ G$
 shows $\text{atom } x \in \text{atom-dom } G \longleftrightarrow \text{atom } x \in \text{supp } G$
 using *wfG-dom-supp assms* **by** *auto*
 lemma *beta-flip-eq*:
 fixes $x::x$ and $xa::x$ and $\mathcal{B}::\mathcal{B}$
 shows $(x \leftrightarrow xa) \cdot \mathcal{B} = \mathcal{B}$
 proof –
 thm *x-not-in-b-set*
 have $\text{atom } x \notin \mathcal{B} \wedge \text{atom } xa \notin \mathcal{B}$ **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 $\vdash_{wf} \Theta$
 shows $(z \leftrightarrow za) \cdot \Theta = \Theta$
 proof –
 have $\text{supp } \Theta = \{\}$ **using** *wfTh-supp assms* **by** *simp*
 thus *?thesis*
 by (*simp add: flip-fresh-fresh fresh-def*)
 qed
 lemma *theta-flip-eq*:
 assumes $wfTh \ \Theta$
 shows $(x \leftrightarrow xa) \cdot \Theta = \Theta$
 using *wfTh-supp flip-fresh-fresh fresh-def*
 by (*simp add: assms theta-flip-eq2*)
 lemma *wfT-wfC*:
 fixes $c::c$
 assumes $\Theta; \mathcal{B}; \Gamma \vdash_{wf} \{ z : b \mid c \}$ and $\text{atom } z \notin \Gamma$
 shows $\Theta; \mathcal{B}; (z, b, \text{TRUE}) \#_{\Gamma} \Gamma \vdash_{wf} c$

proof –

```

obtain  $za\ ba\ ca$  where  $\ast: \llbracket z : b \mid c \rrbracket = \llbracket za : ba \mid ca \rrbracket \wedge atom\ za \# (\Theta, \mathcal{B}, \Gamma) \wedge \Theta; \mathcal{B}; (za, ba,$ 
 $TRUE) \#_{\Gamma} \Gamma \vdash_{wf} ca$ 
using  $wfT\text{-}elims[OF\ assms(1)]$  by metis
hence  $c1: \llbracket atom\ z \rrbracket lst.\ c = \llbracket atom\ za \rrbracket lst.\ ca$  using  $\tau.eq\text{-}iff$  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  $\ast\ True$  by simp
next
  case False
  have  $\vdash_{wf} \Theta$  using  $wfT\text{-}wf\ wfG\text{-}wf\ assms$  by metis
  moreover have  $atom\ za \# \Gamma$  using  $\ast\ fresh\text{-}prodN$  by auto
  ultimately have  $\Theta; \mathcal{B}; (z \leftrightarrow za) \cdot (za, ba, TRUE) \#_{\Gamma} \Gamma \vdash_{wf} (z \leftrightarrow za) \cdot ca$ 
    using  $wfC.eqvt\ theta\text{-}flip\text{-}eq2\ beta\text{-}flip\text{-}eq\ \ast\ GCons\text{-}eqvt\ assms\ flip\text{-}fresh\text{-}fresh$  by metis
  moreover have  $atom\ z \# ca$ 
  proof –
    have  $supp\ ca \subseteq atom\text{-}dom\ \Gamma \cup \{ atom\ za \} \cup supp\ \mathcal{B}$  using  $\ast\ wfC\text{-}supp\ atom\text{-}dom.\text{simps}$ 
 $toSet.\text{simps}$  by fastforce
    moreover have  $atom\ z \notin atom\text{-}dom\ \Gamma$  using  $assms\ fresh\text{-}def\ wfT\text{-}wf\ wfG\text{-}dom\text{-}supp\ wfC\text{-}supp$ 
by metis
    moreover hence  $atom\ z \notin atom\text{-}dom\ \Gamma \cup \{ atom\ za \}$  using False by simp
    moreover have  $atom\ z \notin supp\ \mathcal{B}$  using  $x\text{-}not\text{-}in\text{-}b\text{-}set$  by simp
    ultimately show  $?thesis$  using  $fresh\text{-}def\ False$  by fast
  qed
  moreover hence  $(z \leftrightarrow za) \cdot ca = c$  using  $type\text{-}eq\text{-}subst\text{-}eq1(3)$   $\ast$  by metis
  ultimately show  $?thesis$  using  $assms\ G\text{-}cons\text{-}flip\text{-}fresh\ \ast$  by auto
qed
qed

```

```

lemma  $u\text{-}not\text{-}in\text{-}dom\text{-}g$ :
  fixes  $u::u$ 
  shows  $atom\ u \notin atom\text{-}dom\ G$ 
  using  $toSet.\text{simps}\ atom\text{-}dom.\text{simps}\ u\text{-}not\text{-}in\text{-}x\text{-}atoms$  by auto

```

```

lemma  $bv\text{-}not\text{-}in\text{-}dom\text{-}g$ :
  fixes  $bv::bv$ 
  shows  $atom\ bv \notin atom\text{-}dom\ G$ 
  using  $toSet.\text{simps}\ atom\text{-}dom.\text{simps}\ u\text{-}not\text{-}in\text{-}x\text{-}atoms$  by auto

```

An important lemma that confirms that Γ does not rely on mutable variables

```

lemma  $u\text{-}not\text{-}in\text{-}g$ :
  fixes  $u::u$ 
  assumes  $wfG\ \Theta\ B\ G$ 
  shows  $atom\ u \notin supp\ G$ 
using  $assms$  proof(induct  $G$  rule:  $\Gamma$ -induct)
case  $GNil$ 
  then show  $?case$  using  $supp\text{-}GNil\ fresh\text{-}def$ 
    using  $fresh\text{-}set\text{-}empty$  by fastforce
next
  case ( $GCons\ x\ b\ c\ \Gamma'$ )

```

moreover hence $\text{atom } u \notin \text{supp } b$ **using**
 $\text{wfB-supp wfC-supp u-not-in-x-atoms wfG-elim wfX-wfY}$ **by** *auto*
moreover hence $\text{atom } u \notin \text{supp } x$ **using** $\text{u-not-in-x-atoms supp-at-base}$ **by** *blast*
moreover hence $\text{atom } u \notin \text{supp } c$ **proof** –
 have $\Theta ; B ; (x, b, \text{TRUE}) \#_{\Gamma} \Gamma' \vdash_{\text{wf}} c$ **using** $\text{wfG-cons-wfC GCons}$ **by** *simp*
 hence $\text{supp } c \subseteq \text{atom-dom } ((x, b, \text{TRUE}) \#_{\Gamma} \Gamma') \cup \text{supp } B$ **using** wfC-supp **by** *blast*
 thus $?thesis$ **using** $\text{u-not-in-dom-g u-not-in-b-atoms}$
 using u-not-in-b-set **by** *auto*
qed
ultimately **have** $\text{atom } u \notin \text{supp } (x, b, c)$ **using** supp-Pair **by** *simp*
thus $?case$ **using** $\text{supp-GCons GCons wfG-elim}$ **by** *blast*
qed

lemma u-not-in-t :

fixes $u::u$
assumes $\text{wfT } \Theta \ B \ G \ \tau$
shows $\text{atom } u \notin \text{supp } \tau$
proof –
 have $\text{supp } \tau \subseteq \text{atom-dom } G \cup \text{supp } B$ **using** wfT-supp assms **by** *auto*
 thus $?thesis$ **using** $\text{u-not-in-dom-g u-not-in-b-set}$ **by** *blast*
qed

lemma wfT-supp-c :

fixes $\mathcal{B}::\mathcal{B}$ **and** $z::x$
assumes $\text{wfT } P \ \mathcal{B} \ \Gamma \ (\{ z : b \mid c \})$
shows $\text{supp } c - \{ \text{atom } z \} \subseteq \text{atom-dom } \Gamma \cup \text{supp } \mathcal{B}$
using $\text{wf-supp } \tau.\text{supp assms}$
by $(\text{metis Un-subset-iff empty-set list.simps}(15))$

lemma $\text{wfG-wfC}[ms-wb]$:

assumes $\text{wfG } P \ \mathcal{B} \ ((x, b, c) \#_{\Gamma} \Gamma)$
shows $\text{wfC } P \ \mathcal{B} \ ((x, b, \text{TRUE}) \#_{\Gamma} \Gamma) \ c$
using assms **proof** $(\text{cases } c \in \{ \text{TRUE}, \text{FALSE} \})$
 case *True*
 have $\text{atom } x \# \Gamma \wedge \text{wfG } P \ \mathcal{B} \ \Gamma \wedge \text{wfB } P \ \mathcal{B} \ b$ **using** wfG-cons assms **by** *auto*
 hence $\text{wfG } P \ \mathcal{B} \ ((x, b, \text{TRUE}) \#_{\Gamma} \Gamma)$ **using** wfG-cons2I **by** *auto*
 then show $?thesis$ **using** $\text{wfC-trueI wfC-falseI True}$ **by** *auto*
 next
 case *False*
 then show $?thesis$ **using** wfG-elim assms **by** *blast*
qed

lemma wfT-wf-cons :

assumes $\text{wfT } P \ \mathcal{B} \ \Gamma \ (\{ z : b \mid c \})$ **and** $\text{atom } z \# \Gamma$
shows $\text{wfG } P \ \mathcal{B} \ ((z, b, c) \#_{\Gamma} \Gamma)$
using assms **proof** $(\text{cases } c \in \{ \text{TRUE}, \text{FALSE} \})$
 case *True*
 then show $?thesis$ **using** $\text{wfT-wfC wfC-wf wfG-wfB wfG-cons2I assms wfT-wf}$ **by** *fastforce*
 next
 case *False*
 then show $?thesis$ **using** $\text{wfT-wfC wfC-wf wfG-wfB wfG-cons1I wfT-wf wfT-wfC assms}$ **by** *fastforce*

qed

lemma *wfV-b-fresh*:

fixes $b::b$ **and** $v::v$ **and** $bv::bv$

assumes $\Theta; \mathcal{B}; \Gamma \vdash_{wf} v: b$ **and** $bv \notin \mathcal{B}$

shows $atom\ bv \# v$

using *wfV-suppl bv-not-in-dom-g fresh-def assms bv-not-in-bset-suppl* **by** *blast*

lemma *wfCE-b-fresh*:

fixes $b::b$ **and** $ce::ce$ **and** $bv::bv$

assumes $\Theta; \mathcal{B}; \Gamma \vdash_{wf} ce: b$ **and** $bv \notin \mathcal{B}$

shows $atom\ bv \# ce$

using *bv-not-in-dom-g fresh-def assms bv-not-in-bset-suppl wf-suppl1(8)* **by** *fast*

8.7 Freshness

lemma *wfG-fresh-x*:

fixes $\Gamma::\Gamma$ **and** $z::x$

assumes $\Theta; \mathcal{B}; \Gamma \vdash_{wf} \Gamma$ **and** $atom\ z \# \Gamma$

shows $atom\ z \# (\Theta, \mathcal{B}, \Gamma)$

unfolding *fresh-prodN* **apply**(*intro conjI*)

using *wf-suppl1 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** $atom\ x \# c$

shows $P; \mathcal{B}; G \vdash_{wf} \{ z : b \mid c \}$

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** $atom\ x \# (P, \mathcal{B}, G)$ **using** *wfG-elim3 assms wfG-fresh-x* **by** *metis*

ultimately **have** $wfT\ P\ \mathcal{B}\ G\ \{ x : b \mid c[z::=V-var\ x]_{cv} \}$ **using** *wfTI assms* **by** *metis*

moreover **have** $\{ x : b \mid c[z::=V-var\ x]_{cv} \} = \{ z : b \mid c \}$ **using** *type-eq-subst (atom x # c)* **by** *auto*

ultimately **show** *?thesis* **by** *auto*

qed

lemma *wfT-wfT-if*:

assumes $wfT\ \Theta\ \mathcal{B}\ \Gamma\ (\{ z2 : b \mid CE-val\ v == CE-val\ (V-lit\ L-false)\ IMP\ c[z::=V-var\ z2]_{cv} \})$

and $atom\ z2 \# (c, \Gamma)$

shows $wfT\ \Theta\ \mathcal{B}\ \Gamma\ \{ z : b \mid c \}$

proof –

have $*$: $atom\ z2 \# (\Theta, \mathcal{B}, \Gamma)$ **using** *wfG-fresh-x wfX-wfY assms fresh-Pair* **by** *metis*

have $wfB\ \Theta\ \mathcal{B}\ b$ **using** *assms wfT-elim3* **by** *metis*

have $\Theta; \mathcal{B}; (GCons\ (z2, b, TRUE)\ \Gamma) \vdash_{wf} (CE-val\ v == CE-val\ (V-lit\ L-false)\ IMP\ c[z::=V-var\ z2]_{cv})$ **using** *wfT-wfC assms fresh-Pair* **by** *auto*

hence $\Theta; \mathcal{B}; ((z2, b, TRUE) \#_{\Gamma} \Gamma) \vdash_{wf} c[z::=V-var\ z2]_{cv}$ **using** *wfC-elim3* **by** *metis*

hence $wfT\ \Theta\ \mathcal{B}\ \Gamma\ (\{ z2 : b \mid c[z::=V-var\ z2]_{cv} \})$ **using** *assms fresh-Pair wfTI (wfB Θ B b) ** **by** *auto*

moreover **have** $\{ z : b \mid c \} = \{ z2 : b \mid c[z::=V-var\ z2]_{cv} \}$ **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 $wfT\ P\ \mathcal{B}\ \Gamma\ \{\!| z : b \mid c \!\}$ **and** $atom\ x\ \# \Gamma$ **and** $x \neq z$

shows $atom\ x\ \# c$

proof(*rule ccontr*)

assume $\neg atom\ x\ \# c$

hence $atom\ x \in supp\ c$ **using** *fresh-def* **by** *auto*

moreover have $supp\ c - set\ [atom\ z] \cup supp\ b \subseteq atom-dom\ \Gamma \cup supp\ \mathcal{B}$

using *assms wfT-supply* $\tau.supply$ **by** *blast*

moreover hence $atom\ x \in supp\ c - set\ [atom\ z]$ **using** *assms ** **by** *auto*

ultimately have $atom\ x \in atom-dom\ \Gamma$ **using** *x-not-in-b-set* **by** *auto*

thus *False* **using** *assms wfG-atoms-supply-eq wfT-wf fresh-def* **by** *metis*

qed

lemma *wfG-x-fresh [simp]*:

fixes $x::x$

assumes $wfG\ P\ \mathcal{B}\ G$

shows $atom\ x \notin atom-dom\ G \longleftrightarrow atom\ x\ \# G$

using *wfG-atoms-supply-eq assms fresh-def* **by** *metis*

lemma *wfD-x-fresh*:

fixes $x::x$

assumes $atom\ x\ \# \Gamma$ **and** $wfD\ P\ B\ \Gamma\ \Delta$

shows $atom\ x\ \# \Delta$

using *assms proof*(*induct* Δ *rule*: Δ -*induct*)

case *DNil*

then show *?case* **using** *supp-DNil fresh-def* **by** *auto*

next

case (*DCons* $u'\ t'\ \Delta'$)

have $wfg: wfG\ P\ B\ \Gamma$ **using** *wfD-wf DCons* **by** *blast*

hence $wfd: wfD\ P\ B\ \Gamma\ \Delta'$ **using** *wfD-elim DCons* **by** *blast*

have $supp\ t' \subseteq atom-dom\ \Gamma \cup supp\ B$ **using** *wfT-supply DCons wfD-elim* **by** *metis*

moreover have $atom\ x \notin atom-dom\ \Gamma$ **using** *DCons(2) fresh-def wfG-supply wfg* **by** *blast*

ultimately have $atom\ x\ \# t'$ **using** *fresh-def DCons wfG-supply wfg x-not-in-b-set* **by** *blast*

moreover have $atom\ x\ \# u'$ **using** *supp-at-base fresh-def* **by** *fastforce*

ultimately have $atom\ x\ \# (u',t')$ **using** *supp-Pair* **by** *fastforce*

thus *?case* **using** *DCons fresh-DCons wfd* **by** *fast*

qed

thm *wf-supply2*

lemma *wfG-fresh-x2*:

fixes $\Gamma::\Gamma$ **and** $z::x$ **and** $\Delta::\Delta$ **and** $\Phi::\Phi$

assumes $\Theta; \mathcal{B}; \Gamma \vdash_{wf} \Delta$ **and** $\Theta \vdash_{wf} \Phi$ **and** $atom\ z\ \# \Gamma$

shows $atom\ z\ \# (\Theta, \Phi, \mathcal{B}, \Gamma, \Delta)$

unfolding *fresh-prodN* **apply**(*intro conjI*)

using *wfG-fresh-x assms fresh-prod3 wfX-wfY* **apply** *metis*

using *wf-supply2(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-sup2(6)* *assms fresh-def wfD-x-fresh* **by** *metis*

lemma *wfV-x-fresh:*

fixes *v::v* **and** *b::b* **and** $\Gamma::\Gamma$ **and** *x::x*
assumes $\Theta; \mathcal{B}; \Gamma \vdash_{wf} v : b$ **and** $\text{atom } x \# \Gamma$
shows $\text{atom } x \# v$

proof –

have $\text{supp } v \subseteq \text{atom-dom } \Gamma \cup \text{supp } \mathcal{B}$ **using** *assms wfV-sup* **by** *auto*
moreover **have** $\text{atom } x \notin \text{atom-dom } \Gamma$ **using** *fresh-def assms*
dom.simps subsetCE wfE-elim wfG-sup **by** (*metis dom-sup-g*)
moreover **have** $\text{atom } x \notin \text{supp } \mathcal{B}$ **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** $\Gamma::\Gamma$ **and** $\Delta::\Delta$ **and** $\Phi::\Phi$ **and** *x::x*
assumes $\Theta; \Phi; \mathcal{B}; \Gamma; \Delta \vdash_{wf} e : b$ **and** $\text{atom } x \# \Gamma$
shows $\text{atom } x \# e$

proof –

have $\text{wfG } \Theta \mathcal{B} \Gamma$ **using** *assms wfE-wf* **by** *auto*
hence $\text{supp } e \subseteq \text{atom-dom } \Gamma \cup \text{supp } \mathcal{B} \cup \text{atom}'\text{fst}'\text{setD } \Delta$ **using** *wfE-sup dom.simps assms* **by** *auto*
moreover **have** $\text{atom } x \notin \text{atom-dom } \Gamma$ **using** *fresh-def assms*
dom.simps subsetCE <wfG Θ B Γ> wfG-sup **by** (*metis dom-sup-g*)
moreover **have** $\text{atom } x \notin \text{atom}'\text{fst}'\text{setD } \Delta$ **by** *auto*
ultimately show *?thesis* **using** *fresh-def x-not-in-b-set* **by** *fast*

qed

lemma *wfT-x-fresh:*

fixes $\tau::\tau$ **and** $\Gamma::\Gamma$ **and** *x::x*
assumes $\Theta; \mathcal{B}; \Gamma \vdash_{wf} \tau$ **and** $\text{atom } x \# \Gamma$
shows $\text{atom } x \# \tau$

proof –

have $\text{wfG } \Theta \mathcal{B} \Gamma$ **using** *assms wfX-wfY* **by** *auto*
hence $\text{supp } \tau \subseteq \text{atom-dom } \Gamma \cup \text{supp } \mathcal{B}$ **using** *wfT-sup dom.simps assms* **by** *auto*
moreover **have** $\text{atom } x \notin \text{atom-dom } \Gamma$ **using** *fresh-def assms*
dom.simps subsetCE <wfG Θ B Γ> wfG-sup **by** (*metis dom-sup-g*)
moreover **have** $\text{atom } x \notin \text{supp } \mathcal{B}$ **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** $\Delta::\Delta$ **and** *x::x*
assumes $\Theta; \Phi; \mathcal{B}; \Gamma; \Delta \vdash_{wf} s : b$ **and** $\text{atom } x \# \Gamma$
shows $\text{atom } x \# s$

proof –

have $\text{supp } s \subseteq \text{atom-dom } \Gamma \cup \text{atom}'\text{fst}'\text{setD } \Delta \cup \text{supp } \mathcal{B}$ **using** *wf-sup assms* **by** *metis*
moreover **have** $\text{atom } x \notin \text{atom}'\text{fst}'\text{setD } \Delta$ **by** *auto*
moreover **have** $\text{atom } x \notin \text{atom-dom } \Gamma$ **using** *assms fresh-def wfG-dom-sup wfX-wfY* **by** *metis*
moreover **have** $\text{atom } x \notin \text{supp } \mathcal{B}$ **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 $\#$ T
using wf-suppl1 assms fresh-def by fastforce

lemmas wfTh-x-fresh = wfTh-fresh

lemma wfPhi-fresh:
fixes x
assumes wfPhi T P
shows atom x $\#$ P
using wf-suppl 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 $\Gamma::\Gamma$ and $x::x$
assumes wfG P \mathcal{B} ($\Gamma' @ ((x, b, c) \#_{\Gamma} \Gamma)$)
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; $\mathcal{B} \vdash_{wf} (\Gamma 1 @ (x, b, c) \#_{\Gamma} \Gamma)$ using wfG-elim append-g.simps GCons by metis
have atom x1 $\#$ ($\Gamma 1 @ (x, b, c) \#_{\Gamma} \Gamma$) using GCons wfG-elim append-g.simps by metis
hence atom x1 \notin atom-dom ($\Gamma 1 @ (x, b, c) \#_{\Gamma} \Gamma$) using wfG-dom-suppl fresh-def * by metis
thus ?thesis by auto
qed
ultimately show ?case using append-g.simps atom-dom.simps toSet.simps wfG-elim dom.simps
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 ($\Gamma' @ (x, b, c[z::=V-var x]_{cv}) \#_{\Gamma} \Gamma$)
by(induct Γ' rule: Γ -induct, (simp add: toSet.simps atom-dom.simps)+)

lemma wfG-inside-x-neg:
fixes $c::c$ and $x::x$ and $\Gamma::\Gamma$ and $G::\Gamma$ and $xa::x$
assumes $G=(\Gamma' @ (x, b, c[z::=V-var x]_{cv}) \#_{\Gamma} \Gamma)$ and atom xa $\#$ 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-suppl-eq assms by metis

moreover have $\text{atom } x \in \text{atom-dom } G$ 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 $G = (\Gamma' @ (x, b, c[z::=V\text{-var } x]_{cv}) \#_{\Gamma} \Gamma)$ and $\text{atom } xa \# G$ and $\Theta; \mathcal{B} \vdash_{wf} G$
shows $\text{atom } xa \# x$
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 $\text{supp } t = \{\}$
using *wfT-supp atom-dom.simps assms toSet.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 $\Theta; \mathcal{B} \vdash_{wf} (\Gamma' @ (x, b, c) \#_{\Gamma} \Gamma) \wedge \text{atom } x' \# (\Gamma' @ (x, b, c) \#_{\Gamma} \Gamma) \wedge \Theta; \mathcal{B} \vdash_{wf} b' \wedge x' \neq x$
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 $*\Theta; \mathcal{B} \vdash_{wf} (\Gamma' @ (x, b, c) \#_{\Gamma} \Gamma) \wedge \text{atom } x' \# (\Gamma' @ (x, b, c) \#_{\Gamma} \Gamma) \wedge \Theta; \mathcal{B} \vdash_{wf} b'$ using
assms wfG-cons by *metis*
moreover have $\text{atom } x' \# x$ **proof**(*rule wfG-inside-x-fresh[of (\Gamma' @ (x, b, c) \#_{\Gamma} \Gamma)]*)
show $\Gamma' @ (x, b, c) \#_{\Gamma} \Gamma = \Gamma' @ (x, b, c[z::=V\text{-var } x]_{cv}) \#_{\Gamma} \Gamma$ by *simp*
show $\text{atom } x' \# \Gamma' @ (x, b, c) \#_{\Gamma} \Gamma$ 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 $\Theta; \mathcal{B}; \Gamma \vdash_{wf} \tau$
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* $P \mathcal{B} \Gamma \{z : b \mid c\}$ and $\text{atom } x \# (c, \Gamma)$
shows *wfG* $P \mathcal{B} ((x, b, c[z::=V\text{-var } x]_{cv}) \#_{\Gamma} \Gamma)$

```

have  $\{ x : b \mid c[z::=V\text{-var } x]_{cv} \} = \{ z : b \mid c \}$  using assms freshers type-eq-subst by metis
hence  $*:wfT\ P\ \mathcal{B}\ \Gamma\ \{ x : b \mid c[z::=V\text{-var } x]_{cv} \}$  using assms by metis
show ?thesis proof(rule wfG-consI)
  show  $\langle P; \mathcal{B} \vdash_{wf} \Gamma \rangle$  using assms wfT-wf by auto
  show  $\langle atom\ x\ \sharp\ \Gamma \rangle$  using assms by auto
  show  $\langle P; \mathcal{B} \vdash_{wf} b \rangle$  using assms wfX-wfY b-of.simps by metis
  show  $\langle P; \mathcal{B}; (x, b, TRUE) \#_{\Gamma} \Gamma \vdash_{wf} c[z::=V\text{-var } x]_{cv} \rangle$  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  $y :: 'a :: \text{at-base}$  and  $\Gamma :: \Gamma$ 
assumes  $\text{atom } y \# (\Gamma' @ (x, b, c)) \#_{\Gamma} \Gamma$ 
shows  $\text{atom } y \# (\Gamma' @ \Gamma)$ 
using assms proof(induct  $\Gamma'$  rule:  $\Gamma$ -induct)
  case GNil
  then show ?case using fresh-GCons fresh-GNil by auto
next
  case (GCons  $x' b' c' \Gamma''$ )
  then show ?case using fresh-GCons fresh-GNil by auto
qed

```

lemma *wf-restrict1*:

fixes $\Gamma::\Gamma$ and $\Gamma':\Gamma$ and $v::v$ and $e::e$ and $c::c$ and $\tau::\tau$ and $ts::(string*\tau)$ list and $\Delta::\Delta$ and $s::s$
and $b::b$ and $ftq::fun\text{-}typ\text{-}q$ and $ft::fun\text{-}typ$ and $ce::ce$ and $td::type\text{-}def$
and $cs::branch\text{-}s$ and $css::branch\text{-}list$
shows $\Theta; \mathcal{B}; \Gamma \vdash_{wf} v : b \implies \Gamma = \Gamma_1 @ ((x, b', c') \#_{\Gamma} \Gamma_2) \implies atom\ x \# v \implies atom\ x \# \Gamma_1 \implies$
 $\Theta; \mathcal{B}; \Gamma_1 @ \Gamma_2 \vdash_{wf} v : b$ and

$$\begin{array}{l}
\Theta; \mathcal{B}; \Gamma \vdash_{wf} c \quad \Rightarrow \Gamma = \Gamma_1 @ ((x, b', c') \#_{\Gamma} \Gamma_2) \Rightarrow atom\ x \# c \Rightarrow atom\ x \# \Gamma_1 \Rightarrow \Theta; \\
\mathcal{B}; \Gamma_1 @ \Gamma_2 \vdash_{wf} c \text{ and} \\
\Theta; \mathcal{B} \vdash_{wf} \Gamma \quad \Rightarrow \Gamma = \Gamma_1 @ ((x, b', c') \#_{\Gamma} \Gamma_2) \Rightarrow atom\ x \# \Gamma_1 \Rightarrow \Theta; \mathcal{B} \vdash_{wf} \Gamma_1 @ \Gamma_2 \text{ and} \\
\Theta; \mathcal{B}; \Gamma \vdash_{wf} \tau \quad \Rightarrow \Gamma = \Gamma_1 @ ((x, b', c') \#_{\Gamma} \Gamma_2) \Rightarrow atom\ x \# \tau \Rightarrow atom\ x \# \Gamma_1 \Rightarrow \Theta; \\
\mathcal{B}; \Gamma_1 @ \Gamma_2 \vdash_{wf} \tau \text{ and} \\
\Theta; \mathcal{B}; \Gamma \vdash_{wf} ts \Rightarrow True \text{ and} \\
\vdash_{wf} \Theta \Rightarrow True \text{ and} \\
\Theta; \mathcal{B} \vdash_{wf} b \Rightarrow True \text{ and}
\end{array}$$
$$\begin{array}{l} \Theta; \mathcal{B}; \Gamma \vdash_{wf} ce : b \implies \Gamma = \Gamma_1 @ ((x, b', c') \#_{\Gamma} \Gamma_2) \implies atom\ x \# ce \implies atom\ x \# \Gamma_1 \implies \Theta; \\ \mathcal{B}; \Gamma_1 @ \Gamma_2 \vdash_{wf} ce : b \text{ and } \\ \Theta \vdash_{wf} td \implies True \\ \text{proof}(\text{induct } \text{arbitrary}; \Gamma_1 \text{ and } \Gamma_1 \text{ and } \Gamma_1 \text{ and } \Gamma_1 \text{ and } \Gamma_1 \text{ and } \Gamma_1 \text{ and } \Gamma_1 \text{ and } \Gamma_1 \text{ and } \Gamma_1 \text{ and } \Gamma_1 \text{ and } \\ \Gamma_1 \text{ and } \Gamma_1 \text{ and } \Gamma_1 \text{ and } \Gamma_1 \text{ and } \Gamma_1 \text{ and } \Gamma_1 \text{ and } \Gamma_1 \\ \text{rule}; wfV\text{-}wfC\text{-}wfG\text{-}wfT\text{-}wfTs\text{-}wfTh\text{-}wfB\text{-}wfCE\text{-}wfTD.\text{inducts}) \end{array}$$

```

case (wfV-varI  $\Theta \mathcal{B} \Gamma b c y$ )
hence  $y \neq x$  using v.fresh by auto
hence  $\text{Some } (b, c) = \text{lookup } (\Gamma_1 @ \Gamma_2) y$  using lookup-restrict wfV-varI by metis
then show ?case using wfV-varI wf-intros by metis
next
case (wfV-litI  $\Theta \Gamma l$ )
then show ?case using e.fresh wf-intros by metis
next
case (wfV-pairI  $\Theta \mathcal{B} \Gamma v1 b1 v2 b2$ )
show ?case proof
  show  $\Theta; \mathcal{B}; \Gamma_1 @ \Gamma_2 \vdash_{wf} v1 : b1$  using wfV-pairI by auto
  show  $\Theta; \mathcal{B}; \Gamma_1 @ \Gamma_2 \vdash_{wf} v2 : b2$  using wfV-pairI by auto
qed
next
case (wfV-consI s dclist  $\Theta dc x b c \mathcal{B} \Gamma v$ )
show ?case proof
  show AF-typedef s dclist  $\in \text{set } \Theta$  using wfV-consI by auto
  show  $(dc, \{\!| x : b \mid c \!\}) \in \text{set } dclist$  using wfV-consI by auto
  show  $\Theta; \mathcal{B}; \Gamma_1 @ \Gamma_2 \vdash_{wf} v : b$  using wfV-consI by auto
qed
next
case (wfV-conspI s bv dclist  $\Theta dc x b' c \mathcal{B} b \Gamma v$ )
show ?case proof
  show AF-typedef-poly s bv dclist  $\in \text{set } \Theta$  using wfV-conspI by auto
  show  $(dc, \{\!| x : b' \mid c \!\}) \in \text{set } dclist$  using wfV-conspI by auto
  show  $\Theta; \mathcal{B} \vdash_{wf} b$  using wfV-conspI by auto
  show  $\Theta; \mathcal{B}; \Gamma_1 @ \Gamma_2 \vdash_{wf} v : b'[bv::=b]_{bb}$  using wfV-conspI by auto
  show atom bv  $\nmid (\Theta, \mathcal{B}, \Gamma_1 @ \Gamma_2, b, v)$  unfolding fresh-prodN fresh-append-g using wfV-conspI
fresh-prodN fresh-GCons fresh-append-g by metis
qed
next
case (wfCE-valI  $\Theta \mathcal{B} \Gamma v b$ )
then show ?case using ce.fresh wf-intros by metis
next
case (wfCE-plusI  $\Theta \mathcal{B} \Gamma v1 v2$ )
then show ?case using ce.fresh wf-intros by metis
next
case (wfCE-leqI  $\Theta \mathcal{B} \Gamma v1 v2$ )
then show ?case using ce.fresh wf-intros by metis
next
case (wfCE-fstI  $\Theta \mathcal{B} \Gamma v1 b1 b2$ )
then show ?case using ce.fresh wf-intros by metis
next
case (wfCE-sndI  $\Theta \mathcal{B} \Gamma v1 b1 b2$ )
then show ?case using ce.fresh wf-intros by metis
next
case (wfCE-concatI  $\Theta \mathcal{B} \Gamma v1 v2$ )
then show ?case using ce.fresh wf-intros by metis
next
case (wfCE-lenI  $\Theta \mathcal{B} \Gamma v1$ )
then show ?case using ce.fresh wf-intros by metis
next

```

```

case (wfTI z  $\Theta$   $\mathcal{B}$   $\Gamma$  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  $\langle \text{atom } z \# (\Theta, \mathcal{B}, \Gamma_1 @ \Gamma_2) \rangle$  using wfTI fresh-restrict[of z] using wfG-fresh-x wfX-wfY wfTI
  fresh-prodN by metis
  show  $\langle \Theta; \mathcal{B} \vdash_{wf} b \rangle$  using wfTI by auto
  have  $\Theta; \mathcal{B}; ((z, b, TRUE) \#_{\Gamma} \Gamma_1) @ \Gamma_2 \vdash_{wf} c$  proof(rule wfTI(5)[of (z, b, TRUE)  $\#_{\Gamma} \Gamma_1$  ])
    show  $\langle (z, b, TRUE) \#_{\Gamma} \Gamma = ((z, b, TRUE) \#_{\Gamma} \Gamma_1) @ (x, b', c') \#_{\Gamma} \Gamma_2 \rangle$  using wfTI by auto
    show  $\langle \text{atom } x \# c \rangle$  using wfTI  $\tau.\text{fresh } \langle x \neq z \rangle$  by auto
    show  $\langle \text{atom } x \# (z, b, TRUE) \#_{\Gamma} \Gamma_1 \rangle$  using wfTI  $\langle x \neq z \rangle$  fresh-GCons by simp
  qed
  thus  $\langle \Theta; \mathcal{B}; (z, b, TRUE) \#_{\Gamma} \Gamma_1 @ \Gamma_2 \vdash_{wf} c \rangle$  by auto
qed
next
case (wfC-eqI  $\Theta$   $\mathcal{B}$   $\Gamma$  e1 b e2)
show ?case proof
  show  $\Theta; \mathcal{B}; \Gamma_1 @ \Gamma_2 \vdash_{wf} e1 : b$  using wfC-eqI c.fresh fresh-Nil by auto
  show  $\Theta; \mathcal{B}; \Gamma_1 @ \Gamma_2 \vdash_{wf} e2 : b$  using wfC-eqI c.fresh fresh-Nil by auto
qed
next
case (wfC-trueI  $\Theta$   $\Gamma$ )
then show ?case using c.fresh wf-intros by metis
next
case (wfC-falseI  $\Theta$   $\Gamma$ )
then show ?case using c.fresh wf-intros by metis
next
case (wfC-conjI  $\Theta$   $\Gamma$  c1 c2)
then show ?case using c.fresh wf-intros by metis
next
case (wfC-disjI  $\Theta$   $\Gamma$  c1 c2)
then show ?case using c.fresh wf-intros by metis
next
case (wfC-notI  $\Theta$   $\Gamma$  c1)
then show ?case using c.fresh wf-intros by metis
next
case (wfC-impI  $\Theta$   $\Gamma$  c1 c2)
then show ?case using c.fresh wf-intros by metis
next
case (wfG-nilI  $\Theta$ )
then show ?case using wfV-varI wf-intros
  by (meson GNil-append  $\Gamma.\text{simps}(3)$ )
next
case (wfG-cons1I c1  $\Theta$   $\mathcal{B}$  G x1 b1)
show ?case proof(cases  $\Gamma_1 = GNil$ )
  case True
    then show ?thesis using wfG-cons1I wfG-consI by auto
  next
  case False
    then obtain  $G'::\Gamma$  where  $*(x1, b1, c1) \#_{\Gamma} G' = \Gamma_1$  using GCons-eq-append-conv wfG-cons1I
    by auto
    hence  $*:G = G' @ (x, b', c') \#_{\Gamma} \Gamma_2$  using wfG-cons1I by auto

```



```

rule:wfE-wfS-wfCS-wfCSS-wfPhi-wfD-wfFTQ-wfFT.inducts)

case (wfE-valI  $\Theta \Phi \Gamma \Delta v b$ )
then show ?case using e.fresh wf-intros wf-restrict1 by metis
next
case (wfE-plusI  $\Theta \Phi \Gamma \Delta v1 v2$ )
then show ?case using e.fresh wf-intros wf-restrict1 by metis
next
case (wfE-leqI  $\Theta \Phi \Gamma \Delta v1 v2$ )
then show ?case using e.fresh wf-intros wf-restrict1 by metis
next
case (wfE-fstI  $\Theta \Phi \Gamma \Delta v1 b1 b2$ )
then show ?case using e.fresh wf-intros wf-restrict1 by metis
next
case (wfE-sndI  $\Theta \Phi \Gamma \Delta v1 b1 b2$ )
then show ?case using e.fresh wf-intros wf-restrict1 by metis
next
case (wfE-concatI  $\Theta \Phi \Gamma \Delta v1 v2$ )
then show ?case using e.fresh wf-intros wf-restrict1 by metis
next
case (wfE-splitI  $\Theta \Phi \Gamma \Delta v1 v2$ )
then show ?case using e.fresh wf-intros wf-restrict1 by metis
next
case (wfE-lenI  $\Theta \Phi \Gamma \Delta v1$ )
then show ?case using e.fresh wf-intros wf-restrict1 by metis
next
case (wfE-appI  $\Theta \Phi \Gamma \Delta f x b c \tau s' v$ )
then show ?case using e.fresh wf-intros wf-restrict1 by metis
next
case (wfE-appPI  $\Theta \Phi \mathcal{B} \Gamma \Delta b' bv v \tau f x b c s$ )
show ?case proof
  show  $\langle \Theta \vdash_{wf} \Phi \rangle$  using wfE-appPI by auto
  show  $\langle \Theta; \mathcal{B}; \Gamma_1 @ \Gamma_2 \vdash_{wf} \Delta \rangle$  using wfE-appPI by auto
  show  $\langle \Theta; \mathcal{B} \vdash_{wf} b' \rangle$  using wfE-appPI by auto

  have atom bv  $\# \Gamma_1 @ \Gamma_2$  using wfE-appPI fresh-prodN fresh-restrict by metis
  thus  $\langle atom bv \# (\Phi, \Theta, \mathcal{B}, \Gamma_1 @ \Gamma_2, \Delta, b', v, (b\text{-of } \tau)[bv::=b]_b) \rangle$ 
    using wfE-appPI fresh-prodN by auto

  show  $\langle Some (AF-fundef f (AF-fun-typ-some bv (AF-fun-typ x b c \tau s))) = lookup-fun \Phi f \rangle$  using
wfE-appPI by auto
  show  $\langle \Theta; \mathcal{B}; \Gamma_1 @ \Gamma_2 \vdash_{wf} v : b[bv::=b]_b \rangle$  using wfE-appPI wf-restrict1 by auto
qed

next
case (wfE-mvarI  $\Theta \Phi \Gamma \Delta u \tau$ )
then show ?case using e.fresh wf-intros by metis
next

case (wfD-emptyI  $\Theta \Gamma$ )
then show ?case using c.fresh wf-intros wf-restrict1 by metis
next

```



```

case (wfD-cons  $\Theta$   $\mathcal{B}$   $\Gamma$   $\Delta$   $\tau$   $u$ )
show ?case proof
  show  $\Theta$ ;  $\mathcal{B}$ ;  $\Gamma_1 @ \Gamma_2 \vdash_{wf} \Delta$  using wfD-cons fresh-DCons by metis
  show  $\Theta$ ;  $\mathcal{B}$ ;  $\Gamma_1 @ \Gamma_2 \vdash_{wf} \tau$  using wfD-cons fresh-DCons fresh-Pair wf-restrict1 by metis
  show  $u \notin fst \text{ ` } setD \Delta$  using wfD-cons by auto
qed
next
case (wfFTNone  $\Theta$   $ft$ )
then show ?case by auto
next
case (wfFTSome  $\Theta$   $bv$   $ft$ )
then show ?case by auto
next
case (wfFTI  $\Theta$   $B$   $b$   $\Phi$   $x$   $c$   $s$   $\tau$ )
then show ?case by auto
qed(auto)+

lemmas wf-restrict=wf-restrict1 wf-restrict2

lemma wfT-restrict2:
  fixes  $\tau::\tau$ 
  assumes wfT  $\Theta$   $\mathcal{B}$   $((x, b, c) \#_{\Gamma} \Gamma)$   $\tau$  and atom  $x \# \tau$ 
  shows  $\Theta$ ;  $\mathcal{B}$ ;  $\Gamma \vdash_{wf} \tau$ 
  using wf-restrict1(4)[of  $\Theta$   $\mathcal{B}$   $((x, b, c) \#_{\Gamma} \Gamma)$   $\tau$   $GNil$   $x$   $b$   $c$   $\Gamma$ ] assms fresh-GNil append-g.simps by
  auto

lemma wfG-intros2:
  assumes wfC  $P$   $\mathcal{B}$   $((x, b, TRUE) \#_{\Gamma} \Gamma)$   $c$ 
  shows wfG  $P$   $\mathcal{B}$   $((x, b, c) \#_{\Gamma} \Gamma)$ 
proof -
  have wfG  $P$   $\mathcal{B}$   $((x, b, TRUE) \#_{\Gamma} \Gamma)$  using wfC-wf assms by auto
  hence *:wfG  $P$   $\mathcal{B}$   $\Gamma \wedge atom\ x \# \Gamma \wedge wfB\ P\ \mathcal{B}\ b$  using wfG-elim by metis
  show ?thesis using assms proof(cases  $c \in \{TRUE, FALSE\}$ )
    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  $\Gamma::\Gamma$  and  $\Gamma':\Gamma$  and  $v::v$  and  $e::e$  and  $c::c$  and  $\tau::\tau$  and  $ts::(string*\tau)$  list and  $\Delta::\Delta$  and  $s::s$ 
  and  $b::b$  and  $\mathcal{B}::\mathcal{B}$  and  $ftq::fun\text{-}typ\text{-}q$  and  $ft::fun\text{-}typ$  and  $ce::ce$  and  $td::type\text{-}def$ 
  and  $cs::branch\text{-}s$  and  $css::branch\text{-}list$  and  $t::\tau$ 

  shows  $\Theta$ ;  $\mathcal{B}$ ;  $\Gamma \vdash_{wf} v : b \implies \vdash_{wf} \Theta' \implies set\ \Theta \subseteq set\ \Theta' \implies \Theta'; \mathcal{B}; \Gamma \vdash_{wf} v : b$  and
     $\Theta$ ;  $\mathcal{B}$ ;  $\Gamma \vdash_{wf} c \implies \vdash_{wf} \Theta' \implies set\ \Theta \subseteq set\ \Theta' \implies \Theta'; \mathcal{B}; \Gamma \vdash_{wf} c$  and
     $\Theta$ ;  $\mathcal{B} \vdash_{wf} \Gamma \implies \vdash_{wf} \Theta' \implies set\ \Theta \subseteq set\ \Theta' \implies \Theta'; \mathcal{B} \vdash_{wf} \Gamma$  and

```

```

     $\Theta; \mathcal{B}; \Gamma \vdash_{wf} \tau \implies \vdash_{wf} \Theta' \implies \text{set } \Theta \subseteq \text{set } \Theta' \implies \Theta'; \mathcal{B}; \Gamma \vdash_{wf} \tau$  and
     $\Theta; \mathcal{B}; \Gamma \vdash_{wf} ts \implies \vdash_{wf} \Theta' \implies \text{set } \Theta \subseteq \text{set } \Theta' \implies \Theta'; \mathcal{B}; \Gamma \vdash_{wf} ts$  and
     $\vdash_{wf} P \implies \text{True}$  and
     $\Theta; \mathcal{B} \vdash_{wf} b \implies \vdash_{wf} \Theta' \implies \text{set } \Theta \subseteq \text{set } \Theta' \implies \Theta'; \mathcal{B} \vdash_{wf} b$  and
     $\Theta; \mathcal{B}; \Gamma \vdash_{wf} ce : b \implies \vdash_{wf} \Theta' \implies \text{set } \Theta \subseteq \text{set } \Theta' \implies \Theta'; \mathcal{B}; \Gamma \vdash_{wf} ce : b$  and
     $\Theta \vdash_{wf} td \implies \vdash_{wf} \Theta' \implies \text{set } \Theta \subseteq \text{set } \Theta' \implies \Theta' \vdash_{wf} td$ 
proof(nominal-induct b and c and  $\Gamma$  and  $\tau$  and ts and P and b and b and td
  avoiding:  $\Theta'$ 
  rule:wfV-wfC-wfG-wfT-wfTs-wfTh-wfB-wfCE-wfTD.strong-induct)
case (wfV-consI s dclist  $\Theta$  dc x b c  $\mathcal{B}$   $\Gamma$  v)
show ?case proof
  show  $\langle AF\text{-typedef } s \text{ dclist} \in \text{set } \Theta' \rangle$  using wfV-consI by auto
  show  $\langle (dc, \{x : b \mid c\}) \in \text{set } dclist \rangle$  using wfV-consI by auto
  show  $\langle \Theta'; \mathcal{B}; \Gamma \vdash_{wf} v : b \rangle$  using wfV-consI by auto
qed
next
case (wfV-conspI s bv dclist  $\Theta$  dc x b' c  $\mathcal{B}$  b  $\Gamma$  v)
show ?case proof
  show  $\langle AF\text{-typedef-poly } s \text{ bv dclist} \in \text{set } \Theta' \rangle$  using wfV-conspI by auto
  show  $\langle (dc, \{x : b' \mid c\}) \in \text{set } dclist \rangle$  using wfV-conspI by auto
  show  $\langle \Theta'; \mathcal{B}; \Gamma \vdash_{wf} v : b'[bv::=b]_{bb} \rangle$  using wfV-conspI by auto
  show  $\Theta'; \mathcal{B} \vdash_{wf} b$  using wfV-conspI by auto
  show atom bv  $\sharp (\Theta', \mathcal{B}, \Gamma, b, v)$  using wfV-conspI fresh-prodN by auto
qed
next
case (wfTI z  $\Theta$   $\mathcal{B}$   $\Gamma$  b c)
thus ?case using Wellformed.wfTI by auto
next
case (wfB-consI  $\Theta$  s dclist)
show ?case proof
  show  $\langle \vdash_{wf} \Theta' \rangle$  using wfB-consI by auto
  show  $\langle AF\text{-typedef } s \text{ dclist} \in \text{set } \Theta' \rangle$  using wfB-consI by auto
qed
next
case (wfB-appI  $\Theta$   $\mathcal{B}$  b s bv dclist)
show ?case proof
  show  $\langle \vdash_{wf} \Theta' \rangle$  using wfB-appI by auto
  show  $\langle AF\text{-typedef-poly } s \text{ bv dclist} \in \text{set } \Theta' \rangle$  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  $\Gamma::\Gamma$  and  $\Gamma'::\Gamma$  and  $v::v$  and  $e::e$  and  $c::c$  and  $\tau::\tau$  and  $ts::(\text{string}*\tau)$  list and  $\Delta::\Delta$  and  $s::s$ 
  and  $b::b$  and  $\mathcal{B}::\mathcal{B}$  and  $ftq::\text{fun-typ-q}$  and  $ft::\text{fun-typ}$  and  $ce::ce$  and  $td::\text{type-def}$ 
  and  $cs::\text{branch-s}$  and  $css::\text{branch-list}$  and  $t::\tau$ 

  shows
     $\Theta; \Phi; \mathcal{B}; \Gamma; \Delta \vdash_{wf} e : b \implies \vdash_{wf} \Theta' \implies \text{set } \Theta \subseteq \text{set } \Theta' \implies \Theta'; \Phi; \mathcal{B}; \Gamma; \Delta \vdash_{wf} e : b$  and
     $\Theta; \Phi; \mathcal{B}; \Gamma; \Delta \vdash_{wf} s : b \implies \vdash_{wf} \Theta' \implies \text{set } \Theta \subseteq \text{set } \Theta' \implies \Theta'; \Phi; \mathcal{B}; \Gamma; \Delta \vdash_{wf} s : b$  and
     $\Theta; \Phi; \mathcal{B}; \Gamma; \Delta; tid; dc; t \vdash_{wf} cs : b \implies \vdash_{wf} \Theta' \implies \text{set } \Theta \subseteq \text{set } \Theta' \implies \Theta'; \Phi; \mathcal{B}; \Gamma; \Delta;$ 
     $tid; dc; t \vdash_{wf} cs : b$  and

```

$\Theta; \Phi; \mathcal{B}; \Gamma; \Delta; tid; dclist \vdash_{wf} css : b \implies \vdash_{wf} \Theta' \implies set \Theta \subseteq set \Theta' \implies \Theta'; \Phi; \mathcal{B}; \Gamma; \Delta$
 $; tid; dclist \vdash_{wf} css : b$ and
 $\Theta \vdash_{wf} (\Phi::\Phi) \implies \vdash_{wf} \Theta' \implies set \Theta \subseteq set \Theta' \implies \Theta' \vdash_{wf} (\Phi::\Phi)$ and
 $\Theta; \mathcal{B}; \Gamma \vdash_{wf} \Delta \implies \vdash_{wf} \Theta' \implies set \Theta \subseteq set \Theta' \implies \Theta'; \mathcal{B}; \Gamma \vdash_{wf} \Delta$ and
 $\Theta; \Phi \vdash_{wf} ftq \implies \vdash_{wf} \Theta' \implies set \Theta \subseteq set \Theta' \implies \Theta'; \Phi \vdash_{wf} ftq$ and
 $\Theta; \Phi; \mathcal{B} \vdash_{wf} ft \implies \vdash_{wf} \Theta' \implies set \Theta \subseteq set \Theta' \implies \Theta'; \Phi; \mathcal{B} \vdash_{wf} ft$

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 $\Theta \Phi \mathcal{B} \Gamma \Delta b' bv v \tau f x b c s$)
show ?case **proof**
show $\langle \Theta' \vdash_{wf} \Phi \rangle$ **using** wfE-appPI **by** auto
show $\langle \Theta'; \mathcal{B}; \Gamma \vdash_{wf} \Delta \rangle$ **using** wfE-appPI **by** auto
show $\langle \Theta'; \mathcal{B} \vdash_{wf} b' \rangle$ **using** wfE-appPI wf-theta-weakening1 **by** auto
show $\langle atom bv \# (\Phi, \Theta', \mathcal{B}, \Gamma, \Delta, b', v, (b\text{-of } \tau)[bv::=b]_b) \rangle$ **using** wfE-appPI **by** auto
show $\langle Some (AF-fundef f (AF-fun-typ-some bv (AF-fun-typ x b c \tau s))) = lookup-fun \Phi f \rangle$ **using**
 wfE-appPI **by** auto
show $\langle \Theta'; \mathcal{B}; \Gamma \vdash_{wf} v : b[bv::=b]_b \rangle$ **using** wfE-appPI wf-theta-weakening1 **by** auto
qed

next

case (wfS-matchI $\Theta \mathcal{B} \Gamma v tid dclist \Delta \Phi cs b$)
show ?case **proof**
show $\langle \Theta'; \mathcal{B}; \Gamma \vdash_{wf} v : B\text{-id } tid \rangle$ **using** wfS-matchI wf-theta-weakening1 **by** auto
show $\langle AF\text{-typedef } tid dclist \in set \Theta' \rangle$ **using** wfS-matchI **by** auto
show $\langle \Theta'; \mathcal{B}; \Gamma \vdash_{wf} \Delta \rangle$ **using** wfS-matchI **by** auto
show $\langle \Theta' \vdash_{wf} \Phi \rangle$ **using** wfS-matchI **by** auto
show $\langle \Theta'; \Phi; \mathcal{B}; \Gamma; \Delta; tid; dclist \vdash_{wf} cs : b \rangle$ **using** wfS-matchI **by** auto
qed

next

case (wfS-varI $\Theta \mathcal{B} \Gamma \tau v u \Phi \Delta b s$)
show ?case **proof**
show $\langle \Theta'; \mathcal{B}; \Gamma \vdash_{wf} \tau \rangle$ **using** wfS-varI wf-theta-weakening1 **by** auto
show $\langle \Theta'; \mathcal{B}; \Gamma \vdash_{wf} v : b\text{-of } \tau \rangle$ **using** wfS-varI wf-theta-weakening1 **by** auto
show $\langle atom u \# (\Phi, \Theta', \mathcal{B}, \Gamma, \Delta, \tau, v, b) \rangle$ **using** wfS-varI **by** auto
show $\langle \Theta'; \Phi; \mathcal{B}; \Gamma; (u, \tau) \#_{\Delta} \Delta \vdash_{wf} s : b \rangle$ **using** wfS-varI **by** auto
qed

next

case (wfS-letI $\Theta \Phi \mathcal{B} \Gamma \Delta e b' x s b$)
show ?case **proof**
show $\langle \Theta'; \Phi; \mathcal{B}; \Gamma; \Delta \vdash_{wf} e : b' \rangle$ **using** wfS-letI **by** auto
show $\langle \Theta'; \Phi; \mathcal{B}; (x, b', TRUE) \#_{\Gamma} \Gamma; \Delta \vdash_{wf} s : b \rangle$ **using** wfS-letI **by** auto
show $\langle \Theta'; \mathcal{B}; \Gamma \vdash_{wf} \Delta \rangle$ **using** wfS-letI **by** auto
show $\langle atom x \# (\Phi, \Theta', \mathcal{B}, \Gamma, \Delta, e, b) \rangle$ **using** wfS-letI **by** auto
qed

next

case (wfS-let2I $\Theta \Phi \mathcal{B} \Gamma \Delta s1 \tau x s2 b$)
show ?case **proof**
show $\langle \Theta'; \Phi; \mathcal{B}; \Gamma; \Delta \vdash_{wf} s1 : b\text{-of } \tau \rangle$ **using** wfS-let2I **by** auto
show $\langle \Theta'; \mathcal{B}; \Gamma \vdash_{wf} \tau \rangle$ **using** wfS-let2I wf-theta-weakening1 **by** auto
show $\langle \Theta'; \Phi; \mathcal{B}; (x, b\text{-of } \tau, TRUE) \#_{\Gamma} \Gamma; \Delta \vdash_{wf} s2 : b \rangle$ **using** wfS-let2I **by** auto

```

    show ⟨atom x # (Φ, Θ', B, Γ, Δ, s1, b, τ)⟩ using wfS-let2I by auto
qed
next
case (wfS-branchI Θ Φ B x τ Γ Δ s b tid dc)
show ?case proof
  show ⟨Θ'; Φ; B; (x, b-of τ, TRUE) #Γ Γ; Δ ⊢wf s : b⟩ using wfS-branchI by auto
  show ⟨atom x # (Φ, Θ', B, Γ, Δ, Γ, τ)⟩ using wfS-branchI by auto
  show ⟨Θ'; B; Γ ⊢wf Δ⟩ using wfS-branchI by auto
qed
next
case (wfPhi-consI f Φ Θ ft)
show ?case proof
  show f ∉ name-of-fun 'set Φ using wfPhi-consI by auto
  show Θ'; Φ ⊢wf ft using wfPhi-consI by auto
  show Θ' ⊢wf Φ 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 B x c Γ Δ s b)
show ?case proof
  show ⟨Θ'; Φ; B; (x, B-bool, c) #Γ Γ; Δ ⊢wf s : b⟩ using wfS-assertI wf-theta-weakening1 by
auto
  show ⟨Θ'; B; Γ ⊢wf c⟩ using wfS-assertI wf-theta-weakening1 by auto
  show ⟨Θ'; B; Γ ⊢wf Δ⟩ using wfS-assertI wf-theta-weakening1 by auto
  have atom x # Θ' using wf-supp(6)[OF ⟨⊢wf Θ'⟩] fresh-def by auto
  thus ⟨atom x # (Φ, Θ', B, Γ, Δ, 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 ∈ set Θ and ⊢wf Θ
  shows Θ ⊢wf td
  using assms proof(induct Θ )
  case Nil
  then show ?case by auto
next
case (Cons td' Θ')
then consider td = td' | td ∈ set Θ' by auto
then have Θ' ⊢wf td proof(cases)
  case 1
  then show ?thesis using Cons using wfTh-elim by auto

```

```

next
  case 2
  then show ?thesis using Cons using wfTh-elim 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  $\Theta$  and AF-typedef tid dclist1  $\in$  set  $\Theta$  and AF-typedef tid dclist2  $\in$  set  $\Theta$ 
  shows dclist1 = dclist2
using assms proof(induct  $\Theta$  rule:  $\Theta$ -induct)
  case TNil
  then show ?case by auto
next
  case (AF-typedef tid' dclist'  $\Theta'$ )
  then show ?case using wfTh-elim
    by (metis image-eqI name-of-type.simps(1) set-ConsD type-def.eq-iff(1))
next
  case (AF-typedef-poly tid bv dclist  $\Theta'$ )
  then show ?case using wfTh-elim by auto
qed

```

```

lemma wfTs-ctor-unique:
  fixes dclist::(string* $\tau$ ) list
  assumes  $\Theta$  ;  $\{\|\}$  ;  $GNil \vdash_{wf} dclist$  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-elim set-ConsD
      by (metis fst-conv image-eqI prod.inject)
  next
    case 2
    then show ?thesis using Cons wfTs-elim set-ConsD
      by (metis fst-conv image-eqI prod.inject)
  next
    case 3
    then show ?thesis using Cons wfTs-elim by (metis set-ConsD)
  qed
qed

```

```

lemma wfTD-ctor-unique:
  assumes  $\Theta \vdash_{wf} (AF-typedef tid dclist)$  and  $(c, t1) \in$  set dclist and  $(c, t2) \in$  set dclist
  shows  $t1 = t2$ 
using wfTD-elim wfTs-elim assms wfTs-ctor-unique by metis

```

lemma *wfTh-ctor-unique*:
assumes *wfTh* Θ **and** *AF-typedef* *tid* *dclist* \in *set* Θ **and** $(c, t1) \in$ *set* *dclist* **and** $(c, t2) \in$ *set* *dclist*
shows $t1 = t2$
using *lookup-wfTD* *wfTD-ctor-unique* *assms* **by** *metis*

lemma *wfTs-supp-t*:
fixes *dclist*::(*string** τ) *list*
assumes $(c, t) \in$ *set* *dclist* **and** $\Theta ; B ; GNil \vdash_{wf}$ *dclist*
shows *supp* *t* \subseteq *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* = $(c, t) \mid (c, t) \in$ *set* *dclist'* **by** *auto*
then show *?case* **proof**(*cases*)
case 1
then have $\Theta ; B ; GNil \vdash_{wf}$ *t* **using** *Cons* *wfTs-elim*s **by** *blast*
thus *?thesis* **using** *wfT-supp* *atom-dom.simps* **by** *force*
next
case 2
then show *?thesis* **using** *Cons* *wfTs-elim*s **by** *metis*
qed
qed

lemma *wfTh-lookup-supp-empty*:
fixes *t*:: τ
assumes *AF-typedef* *tid* *dclist* \in *set* Θ **and** $(c, t) \in$ *set* *dclist* **and** \vdash_{wf} Θ
shows *supp* *t* = $\{\}$
proof –
have $\Theta ; \{\mid\} ; GNil \vdash_{wf}$ *dclist* **using** *assms* *lookup-wfTD* *wfTD-elim*s **by** *metis*
thus *?thesis* **using** *wfTs-supp-t* *assms* **by** *force*
qed

lemma *wfTh-supp-b*:
assumes *AF-typedef* *tid* *dclist* \in *set* Θ **and** $(dc, \llbracket z : b \mid c \rrbracket) \in$ *set* *dclist* **and** \vdash_{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 $(dc, \llbracket x1 : b1 \mid c1 \rrbracket) \in$ *set* *dclist1* **and** $(dc, \llbracket x2 : b2 \mid c2 \rrbracket) \in$ *set* *dclist2* **and**
wfTs *P* $\{\mid bva1\mid\}$ *GNil* *dclist1* **and** *wfTs* *P* $\{\mid bva2\mid\}$ *GNil* *dclist2*
 $[[atom\ bva1]]lst.dclist1 = [[atom\ bva2]]lst.dclist2$
shows $[[atom\ bva1]]lst. (dc, \llbracket x1 : b1 \mid c1 \rrbracket) = [[atom\ bva2]]lst. (dc, \llbracket x2 : b2 \mid c2 \rrbracket)$
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-elim 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-elim 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-elim(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* \in *set P* **and** $\vdash_{wf} P$

shows $P ; \{|bva|\} ; GNil \vdash_{wf} dclist$

proof –

have $*:P \vdash_{wf} AF\text{-typedef-poly tyid bva dclist}$ **using** *lookup-wfTD assms* **by** *simp*

obtain *bv lst* **where** $*:P ; \{|bv|\} ; GNil \vdash_{wf} lst \wedge$

$(\forall c. \text{atom } c \# (dclist, lst) \longrightarrow \text{atom } c \# (bva, bv, dclist, lst) \longrightarrow (bva \leftrightarrow c) \cdot dclist = (bv \leftrightarrow c) \cdot lst)$

using *wfTD-elim(2)[OF *]* **by** *metis*

obtain *c::bv* **where** $*:\text{atom } c \# ((dclist, lst), (bva, bv, dclist, lst))$ **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* $\in set\ \Theta$ **and** *AF-typedef-poly* *tid* *bva2* *dclist2* $\in set\ \Theta$

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-elim* **by** *auto*

next

case (*AF-typedef-poly* *tid* *bv* *dclist* Θ')

then show *?case* **using** *wfTh-elim* *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* $\in set\ \Theta$ **and** $(c,t) \in set\ dclist$ **and** $\vdash_{wf}\ \Theta$

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* $\in set\ \Theta$ **and** $(dc, \llbracket z : b \mid c \rrbracket) \in set\ dclist$ **and** $\vdash_{wf}\ \Theta$

shows $supp\ b \subseteq \{atom\ bv\}$

using *assms* *wfTh-poly-lookup-supp* $\tau.supp$ **by** *force*

lemma *subst-g-inside*:
fixes $x::x$ **and** $c::c$ **and** $\Gamma::\Gamma$ **and** $\Gamma':\Gamma$
assumes $wfG\ P\ \mathcal{B}\ (\Gamma' @ (x, b, c[z::=V-var\ x]_{cv}) \#_{\Gamma} \Gamma)$
shows $(\Gamma' @ (x, b, c[z::=V-var\ x]_{cv}) \#_{\Gamma} \Gamma)[x::=v]_{\Gamma v} = (\Gamma'[x::=v]_{\Gamma v} @ \Gamma)$
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 $wfg:wfg\ P\ \mathcal{B}\ (G @ (x, b, c[z::=V-var\ x]_{cv}) \#_{\Gamma} \Gamma) \wedge atom\ x' \# (G @ (x, b, c[z::=V-var\ x]_{cv}) \#_{\Gamma} \Gamma)$ **using** *wfG-elim*(2)
using *GCons.prem*s *append-g.simps* **by** *metis*
hence $atom\ x \notin atom-dom\ ((x', b', c') \#_{\Gamma} G)$ **using** *GCons wfG-inside-fresh* **by** *fast*
hence $x \neq x'$
using *GCons append-Cons wfG-inside-fresh atom-dom.simps toSet.simps* **by** *simp*
hence $((GCons\ (x', b', c')\ G) @ (GCons\ (x, b, c[z::=V-var\ x]_{cv})\ \Gamma))[x::=v]_{\Gamma v} =$
 $(GCons\ (x', b', c')\ (G @ (GCons\ (x, b, c[z::=V-var\ x]_{cv})\ \Gamma)))[x::=v]_{\Gamma v}$ **by** *auto*
also have $\dots = GCons\ (x', b', c'[x::=v]_{cv})\ ((G @ (GCons\ (x, b, c[z::=V-var\ x]_{cv})\ \Gamma))[x::=v]_{\Gamma v})$
using *subst-gv.simps* $\langle x \neq x' \rangle$ **by** *simp*
also have $\dots = (x', b', c'[x::=v]_{cv}) \#_{\Gamma} (G[x::=v]_{\Gamma v} @ \Gamma)$ **using** *GCons wfg* **by** *blast*
also have $\dots = ((x', b', c') \#_{\Gamma} G)[x::=v]_{\Gamma v} @ \Gamma$ **using** *subst-gv.simps* $\langle x \neq x' \rangle$ **by** *simp*
finally show *?case* **by** *auto*
qed

lemma *wfTh-td-eq*:
assumes $td1 \in set\ (td2 \# P)$ **and** $wfTh\ (td2 \# P)$ **and** $name-of-type\ td1 = name-of-type\ td2$
shows $td1 = td2$
proof(*rule ccontr*)
assume $as: td1 \neq td2$
have $name-of-type\ td2 \notin name-of-type\ 'set\ P$ **using** *wfTh-elim*s(2)[*OF assms*(2)] **by** *metis*
moreover have $td1 \in set\ P$ **using** *assms* **as** **by** *simp*
ultimately have $name-of-type\ td1 \neq name-of-type\ td2$
by (*metis rev-image-eqI*)
thus *False* **using** *assms* **by** *auto*
qed

lemma *wfTh-td-unique*:
assumes $td1 \in set\ P$ **and** $td2 \in set\ P$ **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 $(Cons\ td\ \Theta')$
consider $td = td1 \mid td = td2 \mid td \neq td1 \wedge td \neq td2$ **by** *auto*
then show *?case* **proof**(*cases*)
case 1
then show *?thesis* **using** *Cons wfTh-elim wfTh-td-eq* **by** *metis*
next

```

    case 2
    then show ?thesis using Cons wfTh-elim wfTh-td-eq by metis
next
    case 3
    then show ?thesis using Cons wfTh-elim by auto
qed
qed

lemma wfTs-distinct:
  fixes dclist::(string *  $\tau$ ) list
  assumes  $\Theta ; B ; GNil \vdash_{wf} dclist$ 
  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.prem distinct.simps(2) fst-conv list.set-map list.simps(9) wfTs-elim(2))
qed

```

```

lemma wfTh-dclist-distinct:
  assumes AF-typedef s dclist  $\in$  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-elim by metis
  thus ?thesis using wfTs-distinct by metis
qed

```

```

lemma wfTh-dc-t-unique2:
  assumes AF-typedef s dclist'  $\in$  set P and  $(dc, tc') \in$  set dclist' and AF-typedef s dclist  $\in$  set P and
  wfTh P and
     $(dc, tc) \in$  set dclist
  shows  $tc = tc'$ 
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 wfTh-dc-t-unique:
  assumes AF-typedef s dclist'  $\in$  set P and  $(dc, \llbracket x' : b' \mid c' \rrbracket) \in$  set dclist' and AF-typedef s dclist
   $\in$  set P and wfTh P and
     $(dc, \llbracket x : b \mid c \rrbracket) \in$  set dclist
  shows  $\llbracket x' : b' \mid c' \rrbracket = \llbracket x : b \mid c \rrbracket$ 
using assms wfTh-dc-t-unique2 by metis

```

lemma *wfTs-wfT*:
fixes *dclist::(string * τ) list* **and** *t:: τ*
assumes $\Theta; \mathcal{B}; GNil \vdash_{wf} dclist$ **and** $(dc, t) \in set\ dclist$
shows $\Theta; \mathcal{B}; GNil \vdash_{wf} t$
using *assms* **proof**(*induct dclist rule:list.induct*)
case *Nil*
then show *?case* **by** *auto*
next
case (*Cons x1 x2*)
thus *?case* **using** *wfTs-elim*s(2)[*OF Cons*(2)] **by** *auto*
qed

lemma *wfTh-wfT*:
fixes *t:: τ*
assumes *wfTh P* **and** *AF-typedef tid dclist* $\in set\ P$ **and** $(dc, t) \in set\ dclist$
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-elim*s **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** $(dc, \{\!| x : b \mid c \!\}) \in set\ dclist1$
shows $\exists x2\ b2\ c2. (dc, \{\!| x2 : b2 \mid c2 \!\}) \in set\ dclist2$
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 $*:[[atom\ bva1]]lst.\ dclist1' = [[atom\ bva2]]lst.\ dclist2' \wedge [[atom\ bva1]]lst.\ dct1' = [[atom\ bva2]]lst.\ dct2'$
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* $\langle \bigwedge x2\ x1\ t2'\ t2a\ t2\ t1. [[atom\ x1]]lst.\ (t1, t2a) = [[atom\ x2]]lst.\ (t2, t2') \implies t1 = t2 \rangle prod.exhaust-sel$)
qed
obtain *x2 b2* **and** *c2* **where** $snd\ dct2' = \{\!| x2 : b2 \mid c2 \!\}$ **using** *obtain-fresh-z* **by** *metis*
hence $(dc, \{\!| x2 : b2 \mid c2 \!\}) = dct2'$ **using** $\langle dc = fst\ dct2' \rangle$
by (*metis prod.exhaust-sel*)
then show *?thesis* **using** *cons* **by** *force*

```

next
  case False
  hence  $(dc, \{ x : b \mid c \}) \in \text{set } dclist1'$  using Cons by auto
  then show ?thesis using Cons
    by  $(metis \text{local}.* \text{cons list.set-intros}(2))$ 
qed

qed

lemma lst-t-b-eq-iff:
  fixes  $bva1::bv$  and  $bva2::bv$ 
  assumes  $[[atom \ bva1]]lst. \{ x1 : b1 \mid c1 \} = [[atom \ bva2]]lst. \{ x2 : b2 \mid c2 \}$ 
  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  $atom \ c \# (\{ x1 : b1 \mid c1 \}, \{ x2 : b2 \mid c2 \})$  and  $atom \ c \# (bva1, bva2, b1, b2)$ 

  show  $(bva1 \leftrightarrow c) \cdot b1 = (bva2 \leftrightarrow c) \cdot b2$  using assms Abs1-eq-iff(3) assms
    by  $(metis \text{Abs1-eq-iff-fresh}(3) \langle atom \ c \# (bva1, bva2, b1, b2) \rangle \tau.fresh \ \tau.perm-simps \text{type-eq-subst-eq2}(2))$ 
qed

lemma wfTh-typedef-poly-b-eq-iff:
  assumes AF-typedef-poly tyid  $bva1 \ dclist1 \in \text{set } P$  and  $(dc, \{ x1 : b1 \mid c1 \}) \in \text{set } dclist1$ 
  and AF-typedef-poly tyid  $bva2 \ dclist2 \in \text{set } P$  and  $(dc, \{ x2 : b2 \mid c2 \}) \in \text{set } dclist2$  and  $\vdash_{wf} P$ 
  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  $[[atom \ bva1]]lst. (dc, \{ x1 : b1 \mid c1 \}) = [[atom \ bva2]]lst. (dc, \{ x2 : b2 \mid c2 \})$  using
wfTh-b-eq-iff assms wfTh-wfTs-poly by metis
  hence  $[[atom \ bva1]]lst. \{ x1 : b1 \mid c1 \} = [[atom \ bva2]]lst. \{ x2 : b2 \mid c2 \}$  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  $atom \ x \notin \text{supp } us$ 
  by  $(induct \ us, auto, simp \ add: \text{supp-fininsert} \ \text{supp-at-base})$ 

```

```

lemma wfS-flip-eq:
  fixes  $s1::s$  and  $x1::x$  and  $s2::s$  and  $x2::x$  and  $\Delta::\Delta$ 
  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  $atom \ x2 \# \Gamma$  and
     $\Theta; \mathcal{B}; \Gamma \vdash_{wf} \Delta$  and
     $\Theta; \Phi; \mathcal{B}; (x1, b, c1) \#_{\Gamma} \Gamma; \Delta \vdash_{wf} s1 : b\text{-of } t1$ 
  shows  $\Theta; \Phi; \mathcal{B}; (x2, b, c2) \#_{\Gamma} \Gamma; \Delta \vdash_{wf} s2 : b\text{-of } t2$ 
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 $atom\ x1 \# \Gamma$ **using** *wfX-wfY wfG-elim assms* **by** *metis*
 moreover hence $atom\ x1 \# \Delta \wedge atom\ x2 \# \Delta$ **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.eqv 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 $f1: x2 = x1 \wedge t2 = t1 \vee x2 \neq x1 \wedge t2 = (x2 \leftrightarrow x1) \cdot t1 \wedge atom\ x2 \# t1$
 by (*metis (full-types) Abs1-eq-iff(3) <[[atom x1]]lst. t1 = [[atom x2]]lst. t2>*)
 then have $x2 \neq x1 \wedge s2 = (x2 \leftrightarrow x1) \cdot s1 \wedge atom\ x2 \# s1 \longrightarrow b\text{-of}\ t2 = (x2 \leftrightarrow x1) \cdot b\text{-of}\ t1$
 by (*metis b-of.eqv*)
 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 \# \Gamma> <atom\ x2 \# \Gamma>*)
 qed
 qed

8.12 Lookup

lemma *wf-not-in-prefix*:
 assumes $\Theta; B \vdash_{wf} (\Gamma' @ (x, b1, c1) \#_{\Gamma} \Gamma)$
 shows $x \notin fst\ 'toSet\ \Gamma'$
 using *assms* **proof**(*induct* Γ' *rule: \Gamma.induct*)
 case *GNil*
 then show *?case* **by** *simp*
 next
 case (*GCons xbc \Gamma'*)
 then obtain x' and b' and $c'::c$ **where** $xbc: xbc=(x',b',c')$
 using *prod-cases3* **by** *blast*
 hence $*(xbc \#_{\Gamma} \Gamma') @ (x, b1, c1) \#_{\Gamma} \Gamma = ((x',b',c') \#_{\Gamma} (\Gamma' @ ((x, b1, c1) \#_{\Gamma} \Gamma)))$ **by** *simp*
 hence $atom\ x' \# (\Gamma' @ (x, b1, c1) \#_{\Gamma} \Gamma)$ **using** *wfG-elim(2) GCons* **by** *metis*

 moreover have $\Theta; B \vdash_{wf} (\Gamma' @ (x, b1, c1) \#_{\Gamma} \Gamma)$ **using** *GCons wfG-elim ** **by** *metis*
 ultimately have $atom\ x' \notin atom\text{-dom}\ (\Gamma' @ (x, b1, c1) \#_{\Gamma} \Gamma)$ **using** *wfG-dom-suppl 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 toSet.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 $\Theta ; B \vdash_{wf} (\Gamma' @ (x, b1, c1) \#_{\Gamma} \Gamma)$
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 $\Theta :: \Theta$ **and** $\Gamma :: \Gamma$ **and** $\Gamma' :: \Gamma$

assumes $Some (b, c) = lookup \Gamma x$ **and** $toSet \Gamma \subseteq toSet \Gamma'$ **and** $\Theta ; \mathcal{B} \vdash_{wf} \Gamma'$ **and** $\Theta ; \mathcal{B} \vdash_{wf} \Gamma$

shows $Some (b, c) = lookup \Gamma' x$

proof –

have $(x, b, c) \in toSet \Gamma \wedge (\forall b' c'. (x, b', c') \in toSet \Gamma \longrightarrow b' = b \wedge c' = c)$ **using** *assms lookup-iff toSet.simps* **by** *force*

hence $(x, b, c) \in toSet \Gamma'$ **using** *assms* **by** *auto*

moreover have $(\forall b' c'. (x, b', c') \in toSet \Gamma' \longrightarrow b' = b \wedge c' = c)$ **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 $\Phi :: \Phi$

assumes $\Theta \vdash_{wf} \Phi$ **and** $AF-fundef f fd \in set \Phi$

shows $Some (AF-fundef f fd) = lookup-fun \Phi f$

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 $wf : \Theta \vdash_{wf} \Phi' \wedge f' \notin name-of-fun \text{ ' set } \Phi'$ **using** *wfPhi-elim 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 $\Phi' :: \Phi$

assumes $Some fd = lookup-fun \Phi f$ **and** $set \Phi \subseteq set \Phi'$ **and** $\Theta \vdash_{wf} \Phi'$

shows $Some fd = lookup-fun \Phi' f$

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 $\langle a = AF-fundef f' fd' \rangle$ **by** *auto*

qed

qed

lemma *fundef-poly-fresh-bv*:

assumes *atom bv2* $\#$ (*bv1*, *b1*, *c1*, $\tau 1$, *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 $\tau 1$) ((*bv1* \leftrightarrow *bv2*) \cdot *s1*))) = (*AF-fun-typ-some* *bv1* (*AF-fun-typ* *x1* *b1* *c1* $\tau 1$ *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 $\tau 1$) ((*bv1* \leftrightarrow *bv2*) \cdot *s1*)
= (*bv2* \leftrightarrow *bv1*) \cdot *AF-fun-typ* *x1* *b1* *c1* $\tau 1$ *s1*
using *1 2 3 assms* **by** (*simp add: flip-commute*)
moreover have (*atom bv2* $\#$ *c1* \wedge *atom bv2* $\#$ $\tau 1$ \wedge *atom bv2* $\#$ *s1* \vee *atom bv2* \in *set* [*atom x1*]) \wedge *atom bv2* $\#$ *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 $\Gamma::\Gamma$ **and** $\Gamma':\Gamma$ **and** $v::v$ **and** $e::e$ **and** $c::c$ **and** $\tau::\tau$ **and** $ts::(\text{string}*\tau)$ *list* **and** $\Delta::\Delta$ **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 $\Theta; \mathcal{B}; \Gamma \vdash_{wf} v : b \implies \mathcal{B} \sqsubseteq \mathcal{B}' \implies \Theta; \mathcal{B}'; \Gamma \vdash_{wf} v : b$ **and**
 $\Theta; \mathcal{B}; \Gamma \vdash_{wf} c \implies \mathcal{B} \sqsubseteq \mathcal{B}' \implies \Theta; \mathcal{B}'; \Gamma \vdash_{wf} c$ **and**
 $\Theta; \mathcal{B} \vdash_{wf} \Gamma \implies \mathcal{B} \sqsubseteq \mathcal{B}' \implies \Theta; \mathcal{B}' \vdash_{wf} \Gamma$ **and**
 $\Theta; \mathcal{B}; \Gamma \vdash_{wf} \tau \implies \mathcal{B} \sqsubseteq \mathcal{B}' \implies \Theta; \mathcal{B}'; \Gamma \vdash_{wf} \tau$ **and**
 $\Theta; \mathcal{B}; \Gamma \vdash_{wf} ts \implies \mathcal{B} \sqsubseteq \mathcal{B}' \implies \Theta; \mathcal{B}'; \Gamma \vdash_{wf} ts$ **and**
 $\vdash_{wf} P \implies \text{True}$ **and**
 $wfB \ \Theta \ \mathcal{B} \ b \implies \mathcal{B} \sqsubseteq \mathcal{B}' \implies wfB \ \Theta \ \mathcal{B}' \ b$ **and**
 $\Theta; \mathcal{B}; \Gamma \vdash_{wf} ce : b \implies \mathcal{B} \sqsubseteq \mathcal{B}' \implies \Theta; \mathcal{B}'; \Gamma \vdash_{wf} ce : b$ **and**
 $\Theta \vdash_{wf} td \implies \text{True}$

proof(*nominal-induct* *b* **and** *c* **and** Γ **and** τ **and** *ts* **and** *P* **and** *b* **and** *b* **and** *td*
avoiding: 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 $\langle \text{AF-typedef-poly } s \text{ bv } dclist \in \text{set } \Theta \rangle$ **using** *wfV-conspI* **by** *metis*

show $\langle (dc, \{x : b' \mid c\}) \in \text{set } dclist \rangle$ **using** *wfV-conspI* **by** *auto*

show $\langle \Theta; \mathcal{B}' \vdash_{wf} b \rangle$ **using** *wfV-conspI* **by** *auto*

show $\langle \text{atom } bv \# (\Theta, \mathcal{B}', \Gamma, b, v) \rangle$ **using** *fresh-prodN wfV-conspI* **by** *auto*

thus $\langle \Theta; \mathcal{B}'; \Gamma \vdash_{wf} v : b'[bv::=b]_{bb} \rangle$ **using** *wfV-conspI* **by** *simp*

qed

next

case (*wfTI* *z* Θ \mathcal{B} Γ *b* *c*)

```

show ?case proof
  show atom  $z \# (\Theta, \mathcal{B}', \Gamma)$  using wfTI by auto
  show  $\Theta; \mathcal{B}' \vdash_{wf} b$  using wfTI by auto
  show  $\Theta; \mathcal{B}'; (z, b, TRUE) \#_{\Gamma} \Gamma \vdash_{wf} c$  using wfTI by auto
qed
qed( (auto simp add: wf-intros | metis wf-intros)+ )

lemma wb-b-weakening2:
  fixes  $\Gamma::\Gamma$  and  $\Gamma':\Gamma$  and  $v::v$  and  $e::e$  and  $c::c$  and  $\tau::\tau$  and  $ts::(string*\tau)$  list and  $\Delta::\Delta$  and  $s::s$ 
and  $\mathcal{B}::\mathcal{B}$  and  $ftq::fun\text{-}typ\text{-}q$  and  $ft::fun\text{-}typ$  and  $ce::ce$  and  $td::type\text{-}def$ 
and  $cs::branch\text{-}s$  and  $css::branch\text{-}list$ 

  shows
     $\Theta; \Phi; \mathcal{B}; \Gamma; \Delta \vdash_{wf} e : b \implies \mathcal{B} \sqsubseteq \mathcal{B}' \implies \Theta; \Phi; \mathcal{B}'; \Gamma; \Delta \vdash_{wf} e : b$  and
     $\Theta; \Phi; \mathcal{B}; \Gamma; \Delta \vdash_{wf} s : b \implies \mathcal{B} \sqsubseteq \mathcal{B}' \implies \Theta; \Phi; \mathcal{B}'; \Gamma; \Delta \vdash_{wf} s : b$  and
     $\Theta; \Phi; \mathcal{B}; \Gamma; \Delta; tid; dc; t \vdash_{wf} cs : b \implies \mathcal{B} \sqsubseteq \mathcal{B}' \implies \Theta; \Phi; \mathcal{B}'; \Gamma; \Delta; tid; dc; t$ 
 $\vdash_{wf} cs : b$  and
     $\Theta; \Phi; \mathcal{B}; \Gamma; \Delta; tid; dclist \vdash_{wf} css : b \implies \mathcal{B} \sqsubseteq \mathcal{B}' \implies \Theta; \Phi; \mathcal{B}'; \Gamma; \Delta; tid; dclist$ 
 $\vdash_{wf} css : b$  and
     $\Theta \vdash_{wf} (\Phi::\Phi) \implies True$  and
     $\Theta; \mathcal{B}; \Gamma \vdash_{wf} \Delta \implies \mathcal{B} \sqsubseteq \mathcal{B}' \implies \Theta; \mathcal{B}'; \Gamma \vdash_{wf} \Delta$  and
     $\Theta; \Phi \vdash_{wf} ftq \implies True$  and
     $\Theta; \Phi; \mathcal{B} \vdash_{wf} ft \implies \mathcal{B} \sqsubseteq \mathcal{B}' \implies \Theta; \Phi; \mathcal{B}' \vdash_{wf} ft$ 
proof(nominal-induct b and b and b and b and  $\Phi$  and  $\Delta$  and  $ftq$  and  $ft$ 
  avoiding:  $\mathcal{B}'$ 

  rule:wfE-wfS-wfCS-wfCSS-wfPhi-wfD-wfFTQ-wfFT.strong-induct)

  case (wfE-valI  $\Theta \Phi \mathcal{B} \Gamma \Delta v b$ )
  then show ?case using wf-intros wb-b-weakening1 by metis
next
  case (wfE-plusI  $\Theta \Phi \mathcal{B} \Gamma \Delta v1 v2$ )
  then show ?case using wf-intros wb-b-weakening1 by metis
next
  case (wfE-legI  $\Theta \Phi \mathcal{B} \Gamma \Delta v1 v2$ )
  then show ?case using wf-intros wb-b-weakening1 by metis
next
  case (wfE-fstI  $\Theta \Phi \mathcal{B} \Gamma \Delta v1 b1 b2$ )
  then show ?case using Wellformed.wfE-fstI wb-b-weakening1 by metis
next
  case (wfE-sndI  $\Theta \Phi \mathcal{B} \Gamma \Delta v1 b1 b2$ )
  then show ?case using wf-intros wb-b-weakening1 by metis
next
  case (wfE-concatI  $\Theta \Phi \mathcal{B} \Gamma \Delta v1 v2$ )
  then show ?case using wf-intros wb-b-weakening1 by metis
next
  case (wfE-splitI  $\Theta \Phi \mathcal{B} \Gamma \Delta v1 v2$ )
  then show ?case using wf-intros wb-b-weakening1 by metis
next
  case (wfE-lenI  $\Theta \Phi \mathcal{B} \Gamma \Delta v1$ )
  then show ?case using wf-intros wb-b-weakening1 by metis
next

```



```

case (wfE-appI  $\Theta \Phi \mathcal{B} \Gamma \Delta f ft v$ )
then show ?case using wf-intros using wb-b-weakening1 by meson
next
case (wfE-appPI  $\Theta \Phi \mathcal{B}1 \Gamma \Delta b' bv1 v1 \tau1 f1 x1 b1 c1 s1$ )

have  $\Theta ; \Phi ; \mathcal{B}' ; \Gamma ; \Delta \vdash_{wf} AE\text{-appP } f1 \ b' \ v1 : (b\text{-of } \tau1)[bv1 ::= b]_b$ 
proof
  show  $\Theta \vdash_{wf} \Phi$  using wfE-appPI by auto
  show  $\Theta ; \mathcal{B}' ; \Gamma \vdash_{wf} \Delta$  using wfE-appPI by auto
  show  $\Theta ; \mathcal{B}' \vdash_{wf} b'$  using wfE-appPI wb-b-weakening1 by auto
  thus atom bv1  $\# (\Phi, \Theta, \mathcal{B}', \Gamma, \Delta, b', v1, (b\text{-of } \tau1)[bv1 ::= b]_b)$ 
  using wfE-appPI fresh-prodN by auto

  show Some (AF-fundef f1 (AF-fun-typ-some bv1 (AF-fun-typ x1 b1 c1  $\tau1 s1$ ))) = lookup-fun  $\Phi f1$ 
using wfE-appPI by auto
  show  $\Theta ; \mathcal{B}' ; \Gamma \vdash_{wf} v1 : b1[bv1 ::= b]_b$  using wfE-appPI wb-b-weakening1 by auto
qed
then show ?case by auto
next
case (wfE-mvarI  $\Theta \Phi \mathcal{B} \Gamma \Delta u \tau$ )
then show ?case using wf-intros wb-b-weakening1 by metis
next
case (wfS-valI  $\Theta \Phi \mathcal{B} \Gamma v b \Delta$ )
then show ?case using wf-intros wb-b-weakening1 by metis
next
case (wfS-letI  $\Theta \Phi \mathcal{B} \Gamma \Delta e b' x s b$ )
show ?case proof
  show  $\langle \Theta ; \Phi ; \mathcal{B}' ; \Gamma ; \Delta \vdash_{wf} e : b' \rangle$  using wfS-letI by auto
  show  $\langle \Theta ; \Phi ; \mathcal{B}' ; (x, b', TRUE) \#_{\Gamma} \Gamma ; \Delta \vdash_{wf} s : b \rangle$  using wfS-letI by auto
  show  $\langle \Theta ; \mathcal{B}' ; \Gamma \vdash_{wf} \Delta \rangle$  using wfS-letI by auto
  show  $\langle atom \ x \# (\Phi, \Theta, \mathcal{B}', \Gamma, \Delta, e, b) \rangle$  using wfS-letI by auto
qed
next
case (wfS-let2I  $\Theta \Phi \mathcal{B} \Gamma \Delta s1 \tau x s2 b$ )
then show ?case using wb-b-weakening1 Wellformed.wfS-let2I by simp
next
case (wfS-ifI  $\Theta \Phi \mathcal{B} \Gamma v \Phi \Delta s1 b s2$ )
then show ?case using wb-b-weakening1 Wellformed.wfS-ifI by simp
next
case (wfS-varI  $\Theta \mathcal{B} \Gamma \tau v u \Delta \Phi s b$ )
then show ?case using wb-b-weakening1 Wellformed.wfS-varI by simp
next
case (wfS-assignI  $u \tau \Delta \Theta \mathcal{B} \Gamma \Phi v$ )
then show ?case using wb-b-weakening1 Wellformed.wfS-assignI by simp
next
case (wfS-whileI  $\Theta \Phi \mathcal{B} \Gamma \Delta s1 s2 b$ )
then show ?case using wb-b-weakening1 Wellformed.wfS-whileI by simp
next
case (wfS-seqI  $\Theta \Phi \mathcal{B} \Gamma \Delta s1 s2 b$ )
then show ?case using Wellformed.wfS-seqI by metis
next
case (wfS-matchI  $\Theta \mathcal{B} \Gamma v tid dclist \Delta \Phi cs b$ )

```

```

    then show ?case using wb-b-weakening1 Wellformed.wfS-matchI by metis
next
  case (wfS-branchI  $\Theta \Phi \mathcal{B} x \tau \Gamma \Delta s b tid dc$ )
  then show ?case using Wellformed.wfS-branchI by auto
next
  case (wfS-finalI  $\Theta \Phi \mathcal{B} \Gamma \Delta tid dclist' cs b dclist$ )
  then show ?case using wf-intros by metis
next
  case (wfS-cons  $\Theta \Phi \mathcal{B} \Gamma \Delta tid dclist' cs b css dclist$ )
  then show ?case using wf-intros by metis
next
  case (wfD-emptyI  $\Theta \mathcal{B} \Gamma$ )
  then show ?case using wf-intros wb-b-weakening1 by metis
next
  case (wfD-cons  $\Theta \mathcal{B} \Gamma \Delta \tau u$ )
  then show ?case using wf-intros wb-b-weakening1 by metis
next
  case (wfPhi-emptyI  $\Theta$ )
  then show ?case using wf-intros wb-b-weakening1 by metis
next
  case (wfPhi-consI  $f \Theta \Phi ft$ )
  then show ?case using wf-intros wb-b-weakening1 by metis
next
  case (wfFTSome  $\Theta bv ft$ )
  then show ?case using wf-intros wb-b-weakening1 by metis
next
  case (wfFTI  $\Theta B b s x c \tau \Phi$ )
  show ?case proof
    show  $\Theta; \mathcal{B}' \vdash_{wf} b$  using wfFTI wb-b-weakening1 by auto

    show  $supp\ c \subseteq \{atom\ x\}$  using wfFTI wb-b-weakening1 by auto
    show  $\Theta; \mathcal{B}'; (x, b, c) \#_{\Gamma} GNil \vdash_{wf} \tau$  using wfFTI wb-b-weakening1 by auto
    show  $\Theta \vdash_{wf} \Phi$  using wfFTI wb-b-weakening1 by auto
    from  $\langle B \mid \subseteq \mid \mathcal{B}' \rangle$  have  $supp\ B \subseteq supp\ \mathcal{B}'$  proof(induct B)
      case empty
      then show ?case by auto
    next
      case (insert  $x B$ )
      then show ?case
        by (metis fsubset-funion-eq subset-Un-eq supp-union-fset)
    qed
    thus  $supp\ s \subseteq \{atom\ x\} \cup supp\ \mathcal{B}'$  using wfFTI by auto
  qed
next
  case (wfS-assertI  $\Theta \Phi \mathcal{B} x c \Gamma \Delta s b$ )
  show ?case proof
    show  $\langle \Theta; \Phi; \mathcal{B}'; (x, B\text{-}bool, c) \#_{\Gamma} \Gamma; \Delta \vdash_{wf} s : b \rangle$  using wb-b-weakening1 wfS-assertI by simp
    show  $\langle \Theta; \mathcal{B}'; \Gamma \vdash_{wf} c \rangle$  using wb-b-weakening1 wfS-assertI by simp
    show  $\langle \Theta; \mathcal{B}'; \Gamma \vdash_{wf} \Delta \rangle$  using wb-b-weakening1 wfS-assertI by simp
    have  $atom\ x \notin \mathcal{B}'$  using x-not-in-b-set fresh-def by metis
    thus  $\langle atom\ x \notin (\Phi, \Theta, \mathcal{B}', \Gamma, \Delta, c, b, s) \rangle$  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 $\Gamma::\Gamma$

assumes $\mathcal{B} \mid\subseteq \mathcal{B}'$ **and** $\Theta; \mathcal{B} \vdash_{wf} \Gamma$

shows $\Theta; \mathcal{B}' \vdash_{wf} \Gamma$

using *wb-b-weakening* *assms* **by** *auto*

lemma *wfT-b-weakening*:

fixes $\Gamma::\Gamma$ **and** $\Theta::\Theta$ **and** $\tau::\tau$

assumes $\mathcal{B} \mid\subseteq \mathcal{B}'$ **and** $\Theta; \mathcal{B}; \Gamma \vdash_{wf} \tau$

shows $\Theta; \mathcal{B}'; \Gamma \vdash_{wf} \tau$

using *wb-b-weakening* *assms* **by** *auto*

lemma *wfB-subst-wfB*:

fixes $\tau::\tau$ **and** $b'::b$ **and** $b::b$

assumes $\Theta; \{|bv|\} \vdash_{wf} b$ **and** $\Theta; \mathcal{B} \vdash_{wf} b'$

shows $\Theta; \mathcal{B} \vdash_{wf} b[bv::=b']_{bb}$

using *assms* **proof**(*nominal-induct* *b* *rule:b.strong-induct*)

case *B-int*

hence $\Theta; \{|\}\vdash_{wf} B\text{-int}$ **using** *wfB-intI* *wfX-wfY* **by** *fast*

then show *?case* **using** *subst-bb.simps* *wb-b-weakening* **by** *fastforce*

next

case *B-bool*

hence $\Theta; \{|\}\vdash_{wf} B\text{-bool}$ **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 $\Theta; \mathcal{B} \vdash_{wf} (B\text{-id } x)$ **using** *wfB-consI* *wfB-elimI* *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-elimI*(1) *wfB-pairI*)

next

case *B-unit*

hence $\Theta; \{|\}\vdash_{wf} B\text{-unit}$ **using** *wfB-unitI* *wfX-wfY* **by** *fast*

then show *?case* **using** *subst-bb.simps* *wb-b-weakening* **by** *fastforce*

next

case *B-bitvec*

hence $\Theta; \{|\}\vdash_{wf} B\text{-bitvec}$ **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.premI*(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|} ⊢wf b using
wfB-elim by metis
  thm wfB-appI
  show ?case unfolding subst-b-simps proof
    show ⊢wf Θ using B-app wfX-wfY by metis
    show Θ ; B ⊢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) #Γ GNil ⊢wf τ and Θ ; B ⊢wf b'
  shows Θ ; B ⊢wf (b-of τ)[bv::=b]bb
proof -
  obtain b where Θ ; {|bv|} ⊢wf b ∧ b-of τ = b using wfT-elim b-of.simps assms by metis
  thus ?thesis using wfB-subst-wfB assms by auto
qed

lemma wfG-cons-unique:
  assumes (x1,b1,c1) ∈ toSet ((x,b,c) #Γ Γ) and wfG Θ B ((x,b,c) #Γ Γ) and x = x1
  shows b1 = b ∧ c1 = c
proof -
  have x1 ∉ fst ' toSet 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 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) ∈ toSet (Γ'@((x,b,c) #Γ Γ)) and wfG Θ B (Γ'@((x,b,c) #Γ Γ)) 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) ∈ toSet (Γ' @ (x, b, c) #Γ 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 $\Gamma::\Gamma$ **and** $\Gamma'::\Gamma$ **and** $v::v$ **and** $e::e$ **and** $c::c$ **and** $\tau::\tau$ **and** $ts::(\text{string}*\tau)$ *list* **and** $\Delta::\Delta$ **and** $s::s$ **and** $\mathcal{B}::\mathcal{B}$ **and** $ftq::\text{fun-tyq-q}$ **and** $ft::\text{fun-tyf}$ **and** $ce::ce$ **and** $td::\text{type-def}$ **and** $cs::\text{branch-s}$ **and** $css::\text{branch-list}$ **and** $\Phi::\Phi$

shows

$\Theta; \Phi; \mathcal{B}; \Gamma; \Delta \vdash_{wf} e : b \implies \Theta \vdash_{wf} \Phi' \implies \text{set } \Phi \subseteq \text{set } \Phi' \implies \Theta; \Phi'; \mathcal{B}; \Gamma; \Delta \vdash_{wf} e : b$

and

$\Theta; \Phi; \mathcal{B}; \Gamma; \Delta \vdash_{wf} s : b \implies \Theta \vdash_{wf} \Phi' \implies \text{set } \Phi \subseteq \text{set } \Phi' \implies \Theta; \Phi'; \mathcal{B}; \Gamma; \Delta \vdash_{wf} s : b$

and

$\Theta; \Phi; \mathcal{B}; \Gamma; \Delta; tid; dc; t \vdash_{wf} cs : b \implies \Theta \vdash_{wf} \Phi' \implies \text{set } \Phi \subseteq \text{set } \Phi' \implies \Theta; \Phi'; \mathcal{B}; \Gamma; \Delta; tid; dc; t \vdash_{wf} cs : b$ **and**

$\Theta; \Phi; \mathcal{B}; \Gamma; \Delta; tid; dc; t \vdash_{wf} cs : b$ **and**

$\Theta; \Phi; \mathcal{B}; \Gamma; \Delta; tid; dclist \vdash_{wf} css : b \implies \Theta \vdash_{wf} \Phi' \implies \text{set } \Phi \subseteq \text{set } \Phi' \implies \Theta; \Phi'; \mathcal{B}; \Gamma; \Delta; tid; dclist \vdash_{wf} css : b$ **and**

$\Theta \vdash_{wf} (\Phi::\Phi) \implies \text{True}$ **and**

$\Theta; \mathcal{B}; \Gamma \vdash_{wf} \Delta \implies \text{True}$ **and**

$\Theta; \Phi \vdash_{wf} ftq \implies \Theta \vdash_{wf} \Phi' \implies \text{set } \Phi \subseteq \text{set } \Phi' \implies \Theta; \Phi' \vdash_{wf} ftq$ **and**

$\Theta; \Phi; \mathcal{B} \vdash_{wf} ft \implies \Theta \vdash_{wf} \Phi' \implies \text{set } \Phi \subseteq \text{set } \Phi' \implies \Theta; \Phi'; \mathcal{B} \vdash_{wf} ft$

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* $\Theta \Phi \mathcal{B} \Gamma \Delta v b$)

then show *?case using wf-intros by metis*

next

case (*wfE-plusI* $\Theta \Phi \mathcal{B} \Gamma \Delta v1 v2$)

then show *?case using wf-intros by metis*

next

case (*wfE-leqI* $\Theta \Phi \mathcal{B} \Gamma \Delta v1 v2$)

then show *?case using wf-intros by metis*

next

case (*wfE-fstI* $\Theta \Phi \mathcal{B} \Gamma \Delta v1 b1 b2$)

then show *?case using wf-intros by metis*

next

case (*wfE-sndI* $\Theta \Phi \mathcal{B} \Gamma \Delta v1 b1 b2$)

then show *?case using wf-intros by metis*

next

case (*wfE-concatI* $\Theta \Phi \mathcal{B} \Gamma \Delta v1 v2$)

then show *?case using wf-intros by metis*

next

case (*wfE-splitI* $\Theta \Phi \mathcal{B} \Gamma \Delta v1 v2$)

then show *?case using wf-intros by metis*

next

case (*wfE-lenI* $\Theta \Phi \mathcal{B} \Gamma \Delta v1$)

then show *?case using wf-intros by metis*

next

case (*wfE-appI* $\Theta \Phi \mathcal{B} \Gamma \Delta f x b c \tau s v$)

then show *?case using wf-intros lookup-fun-weakening by metis*

next

case (*wfE-appPI* $\Theta \Phi \mathcal{B} \Gamma \Delta b' bv v \tau f x b c s$)

show *?case proof*

show $\langle \Theta \vdash_{wf} \Phi' \rangle$ **using** *wfE-appPI by auto*

```

  show  $\langle \Theta; \mathcal{B}; \Gamma \vdash_{wf} \Delta \rangle$  using wfE-appPI by auto
  show  $\langle \Theta; \mathcal{B} \vdash_{wf} b' \rangle$  using wfE-appPI by auto
  show  $\langle atom\ bv\ \sharp (\Phi', \Theta, \mathcal{B}, \Gamma, \Delta, b', v, (b\text{-of}\ \tau)[bv::=b]_b) \rangle$  using wfE-appPI by auto
  show  $\langle Some\ (AF\text{-fundef}\ f\ (AF\text{-fun-tyt}\text{-some}\ bv\ (AF\text{-fun-tyt}\ x\ b\ c\ \tau\ s))) = lookup\text{-fun}\ \Phi'\ f \rangle$ 
    using wfE-appPI lookup-fun-weakening by metis
  show  $\langle \Theta; \mathcal{B}; \Gamma \vdash_{wf} v : b[bv::=b]_b \rangle$  using wfE-appPI by auto
qed

next
  case (wfE-mvarI  $\Theta\ \Phi\ \mathcal{B}\ \Gamma\ \Delta\ u\ \tau$ )
  then show ?case using wf-intros by metis
next
  case (wfS-valI  $\Theta\ \Phi\ \mathcal{B}\ \Gamma\ v\ b\ \Delta$ )
  then show ?case using wf-intros by metis
next
  case (wfS-letI  $\Theta\ \Phi\ \mathcal{B}\ \Gamma\ \Delta\ e\ b'\ x\ s\ b$ )
  then show ?case using Wellformed.wfS-letI by fastforce
next
  case (wfS-let2I  $\Theta\ \Phi\ \mathcal{B}\ \Gamma\ \Delta\ s1\ b'\ x\ s2\ b$ )
  then show ?case using Wellformed.wfS-let2I by fastforce
next
  case (wfS-ifI  $\Theta\ \Phi\ \mathcal{B}\ \Gamma\ v\ \Phi\ \Delta\ s1\ b\ s2$ )
  then show ?case using wf-intros by metis
next
  case (wfS-varI  $\Theta\ \mathcal{B}\ \Gamma\ \tau\ v\ u\ \Phi\ \Delta\ b\ s$ )
  show ?case proof
    show  $\langle \Theta; \mathcal{B}; \Gamma \vdash_{wf} \tau \rangle$  using wfS-varI by simp
    show  $\langle \Theta; \mathcal{B}; \Gamma \vdash_{wf} v : b\text{-of}\ \tau \rangle$  using wfS-varI by simp
    show  $\langle atom\ u\ \sharp (\Phi', \Theta, \mathcal{B}, \Gamma, \Delta, \tau, v, b) \rangle$  using wfS-varI by simp
    show  $\langle \Theta; \Phi'; \mathcal{B}; \Gamma; (u, \tau) \#_{\Delta} \Delta \vdash_{wf} s : b \rangle$  using wfS-varI by simp
  qed
next
  case (wfS-assignI  $u\ \tau\ \Delta\ \Theta\ \mathcal{B}\ \Gamma\ \Phi\ v$ )
  then show ?case using wf-intros by metis
next
  case (wfS-whileI  $\Theta\ \Phi\ \mathcal{B}\ \Gamma\ \Delta\ s1\ s2\ b$ )
  then show ?case using wf-intros by metis
next
  case (wfS-seqI  $\Theta\ \Phi\ \mathcal{B}\ \Gamma\ \Delta\ s1\ s2\ b$ )
  then show ?case using wf-intros by metis
next
  case (wfS-matchI  $\Theta\ \mathcal{B}\ \Gamma\ v\ tid\ dclist\ \Delta\ \Phi\ cs\ b$ )
  then show ?case using wf-intros by metis
next
  case (wfS-branchI  $\Theta\ \Phi\ \mathcal{B}\ x\ \tau\ \Gamma\ \Delta\ s\ b\ tid\ dc$ )
  then show ?case using Wellformed.wfS-branchI by fastforce
next
  case (wfS-assertI  $\Theta\ \Phi\ \mathcal{B}\ x\ c\ \Gamma\ \Delta\ s\ b$ )
  show ?case proof

    show  $\langle \Theta; \Phi'; \mathcal{B}; (x, B\text{-bool}, c) \#_{\Gamma} \Gamma; \Delta \vdash_{wf} s : b \rangle$  using wfS-assertI by auto
  next

```

```

  show  $\langle \Theta; \mathcal{B}; \Gamma \vdash_{wf} c \rangle$  using wfS-assertI by auto
next
  show  $\langle \Theta; \mathcal{B}; \Gamma \vdash_{wf} \Delta \rangle$  using wfS-assertI by auto

  have atom  $x \# \Phi'$  using wfS-assertI wfPhi-supp fresh-def by blast
  thus  $\langle atom\ x \# (\Phi', \Theta, \mathcal{B}, \Gamma, \Delta, c, b, s) \rangle$  using fresh-prodN wfS-assertI wfPhi-supp fresh-def by auto
qed
next
  case  $(wfFTI\ \Theta\ B\ b\ s\ x\ c\ \tau\ \Phi)$ 
  show ?case proof

  show  $\langle \Theta; B \vdash_{wf} b \rangle$  using wfFTI by auto
next

  show  $\langle supp\ c \subseteq \{atom\ x\} \rangle$  using wfFTI by auto
next
  show  $\langle \Theta; B; (x, b, c) \#_{\Gamma} GNil \vdash_{wf} \tau \rangle$  using wfFTI by auto
next
  show  $\langle \Theta \vdash_{wf} \Phi' \rangle$  using wfFTI by auto
next
  show  $\langle supp\ s \subseteq \{atom\ x\} \cup supp\ B \rangle$  using wfFTI by auto
qed
qed(auto|metis wf-intros)+

lemma wfT-fun-return-t:
  fixes  $\tau a'::\tau$  and  $\tau'::\tau$ 
  assumes  $\Theta; \mathcal{B}; (xa, b, ca) \#_{\Gamma} GNil \vdash_{wf} \tau a'$  and  $(AF-fun-typ\ x\ b\ c\ \tau'\ s') = (AF-fun-typ\ xa\ b\ ca\ \tau a'\ sa')$ 
  shows  $\Theta; \mathcal{B}; (x, b, c) \#_{\Gamma} GNil \vdash_{wf} \tau'$ 
proof -
  obtain  $cb::x$  where  $xf: atom\ cb \# (c, \tau', s', sa', \tau a', ca, x, xa)$  using obtain-fresh by blast
  hence  $atom\ cb \# (c, \tau', s', sa', \tau a', ca) \wedge atom\ cb \# (x, xa, ((c, \tau'), s'), (ca, \tau a'), sa')$  using
fresh-prod6 fresh-prod4 fresh-prod8 by auto
  hence  $*:c[x::=V-var\ cb]_{cv} = ca[xa::=V-var\ cb]_{cv} \wedge \tau'[x::=V-var\ cb]_{\tau v} = \tau a'[xa::=V-var\ cb]_{\tau v}$  using
assms  $\tau.eq\text{-}iff\ Abs1\text{-}eq\text{-}iff\text{-}all$  by auto

  have  $**:\Theta; \mathcal{B}; (xa \leftrightarrow cb) \cdot ((xa, b, ca) \#_{\Gamma} GNil) \vdash_{wf} (xa \leftrightarrow cb) \cdot \tau a'$  using assms True-eqvt
beta-flip-eq theta-flip-eq wfG-wf
  by  $(metis\ GCons\text{-}eqvt\ GNil\text{-}eqvt\ wfT.\text{eqvt}\ wfT\text{-}wf)$ 

  have  $\Theta; \mathcal{B}; (x \leftrightarrow cb) \cdot ((x, b, c) \#_{\Gamma} GNil) \vdash_{wf} (x \leftrightarrow cb) \cdot \tau'$  proof -
  have  $(xa \leftrightarrow cb) \cdot xa = (x \leftrightarrow cb) \cdot x$  using xf by auto
  hence  $(x \leftrightarrow cb) \cdot ((x, b, c) \#_{\Gamma} GNil) = (xa \leftrightarrow cb) \cdot ((xa, b, ca) \#_{\Gamma} GNil)$  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.\text{eqvt} permute-flip-cancel
by metis
qed

```

lemma *wfFT-wf-aux*:

fixes $\tau::\tau$ **and** $\Theta::\Theta$ **and** $\Phi::\Phi$ **and** $ft :: fun\text{-}typ\text{-}q$ **and** $s::s$ **and** $\Delta::\Delta$

assumes $\Theta ; \Phi ; B \vdash_{wf} (AF\text{-}fun\text{-}typ\ x\ b\ c\ \tau\ s)$

shows $\Theta ; B ; (x, b, c) \#_{\Gamma} GNil \vdash_{wf} \tau \wedge \Theta \vdash_{wf} \Phi \wedge supp\ s \subseteq \{ atom\ x \} \cup supp\ B$

proof –

obtain xa **and** ca **and** sa **and** τ' **where** $*:\Theta ; B \vdash_{wf} b \wedge (\Theta \vdash_{wf} \Phi) \wedge$

$supp\ sa \subseteq \{ atom\ xa \} \cup supp\ B \wedge (\Theta ; B ; (xa, b, ca) \#_{\Gamma} GNil \vdash_{wf} \tau') \wedge$

$AF\text{-}fun\text{-}typ\ x\ b\ c\ \tau\ s = AF\text{-}fun\text{-}typ\ xa\ b\ ca\ \tau'\ sa$

using *wfFT.simps[of $\Theta\ \Phi\ B\ AF\text{-}fun\text{-}typ\ x\ b\ c\ \tau\ s$]* **assms** **by** *auto*

moreover **hence** $**:(AF\text{-}fun\text{-}typ\ x\ b\ c\ \tau\ s) = (AF\text{-}fun\text{-}typ\ xa\ b\ ca\ \tau'\ sa)$ **by** *simp*

ultimately **have** $\Theta ; B ; (x, b, c) \#_{\Gamma} GNil \vdash_{wf} \tau$ **using** *wfT-fun-return-t* **by** *metis*

moreover **have** $(\Theta \vdash_{wf} \Phi)$ **using** $*$ **by** *auto*

moreover **have** $supp\ s \subseteq \{ atom\ x \} \cup supp\ B$ **proof** –

have $[[atom\ x]]lst.s = [[atom\ xa]]lst.sa$ **using** $**\ fun\text{-}typ.eq\text{-}iff\ lst\text{-}fst\ lst\text{-}snd$ **by** *metis*

thus *?thesis* **using** *lst-supset-subset* $*$ **by** *metis*

qed

ultimately **show** *?thesis* **by** *auto*

qed

lemma *wfFT-simple-wf*:

fixes $\tau::\tau$ **and** $\Theta::\Theta$ **and** $\Phi::\Phi$ **and** $ft :: fun\text{-}typ\text{-}q$ **and** $s::s$ **and** $\Delta::\Delta$

assumes $\Theta ; \Phi \vdash_{wf} (AF\text{-}fun\text{-}typ\text{-}none\ (AF\text{-}fun\text{-}typ\ x\ b\ c\ \tau\ s))$

shows $\Theta ; \{||\} ; (x, b, c) \#_{\Gamma} GNil \vdash_{wf} \tau \wedge \Theta \vdash_{wf} \Phi \wedge supp\ s \subseteq \{ atom\ x \}$

proof –

have $*:\Theta ; \Phi ; \{||\} \vdash_{wf} (AF\text{-}fun\text{-}typ\ x\ b\ c\ \tau\ s)$ **using** *wfFTQ-elim*s **assms** **by** *auto*

thus *?thesis* **using** *wfFT-wf-aux* **by** *force*

qed

lemma *wfFT-poly-wf*:

fixes $\tau::\tau$ **and** $\Theta::\Theta$ **and** $\Phi::\Phi$ **and** $ftq :: fun\text{-}typ\text{-}q$ **and** $s::s$ **and** $\Delta::\Delta$

assumes $\Theta ; \Phi \vdash_{wf} (AF\text{-}fun\text{-}typ\text{-}some\ bv\ (AF\text{-}fun\text{-}typ\ x\ b\ c\ \tau\ s))$

shows $\Theta ; \{|bv|\} ; (x, b, c) \#_{\Gamma} GNil \vdash_{wf} \tau \wedge \Theta \vdash_{wf} \Phi \wedge \Theta ; \Phi ; \{|bv|\} \vdash_{wf} (AF\text{-}fun\text{-}typ\ x\ b\ c\ \tau\ s)$

proof –

obtain $bv1\ ft1$ **where** $*:\Theta ; \Phi ; \{|bv1|\} \vdash_{wf} ft1 \wedge [[atom\ bv1]]lst.\ ft1 = [[atom\ bv]]lst.\ AF\text{-}fun\text{-}typ\ x\ b\ c\ \tau\ s$

using *wfFTQ-elim*s(β)[*OF* *assms*] **by** *metis*

show *?thesis* **proof**(*cases* $bv1 = bv$)

case *True*

then **show** *?thesis* **using** $*$ *fun\text{-}typ\text{-}q.eq\text{-}iff\ Abs1\text{-}eq\text{-}iff* **by** (*metis* (*no-types*, *hide-lams*) *wfFT-wf-aux*)

next

case *False*

obtain $x1\ b1\ c1\ t1\ s1$ **where** $**:\ ft1 = AF\text{-}fun\text{-}typ\ x1\ b1\ c1\ t1\ s1$ **using** *fun\text{-}typ.eq\text{-}iff*

by (*meson* *fun\text{-}typ.exhaust*)

hence *eqv*: $(bv \leftrightarrow bv1) \cdot AF\text{-}fun\text{-}typ\ x1\ b1\ c1\ t1\ s1 = AF\text{-}fun\text{-}typ\ x\ b\ c\ \tau\ s \wedge atom\ bv1 \nmid AF\text{-}fun\text{-}typ\ x\ b\ c\ \tau\ s$ **using**

Abs1\text{-}eq\text{-}iff(β) $*$ *False* **by** *metis*


```

  have (bv ↔ bv1) · Θ ; (bv ↔ bv1) · Φ ; (bv ↔ bv1) · {|bv1|} ⊢wf (bv ↔ bv1) · ft1 using wfFT.eqv
* by metis
  moreover have (bv ↔ bv1) · Φ = Φ using phi-flip-eq wfX-wfY * by metis
  moreover have (bv ↔ bv1) · Θ = Θ using wfX-wfY * theta-flip-eq2 by metis
  moreover have (bv ↔ bv1) · ft1 = AF-fun-typ x b c τ s using eqv ** by metis
  ultimately have Θ ; Φ ; {|bv|} ⊢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 ⊢wf (AF-fun-typ-some bv (AF-fun-typ x b c τ s))
  shows Θ ; {|bv|} ; (x,b,c) #Γ GNil ⊢wf τ
  using wfFT-poly-wf assms by simp

```

```

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-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)) ∈ set Φ and Θ ⊢wf Φ and set Φ
  ⊆ set Φ' and Θ ⊢wf Φ'
  shows Θ ; {||} ; (x,b,c) #Γ GNil ⊢wf τ ∧ Θ ⊢wf Φ ∧ supp s ⊆ { atom x }
using assms proof(induct Φ rule: Φ-induct)
  case PNil
  then show ?case by auto
next
  case (PConsSome f1 bv x1 b1 c1 τ1 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-elim(3) PConsSome by auto
  ultimately show ?case using PConsSome wfPhi-elim 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.prem(1) PConsNone.prem(2) True fun-def.eq-iff image-eqI name-of-fun.simp)
  hence *:Θ ; Φ'' ⊢wf AF-fun-typ-none (AF-fun-typ x b c τ s) using wfPhi-elim(2)[OF PCon-
  sNone(3)] by metis
  hence Θ ; Φ'' ; {||} ⊢wf (AF-fun-typ x b c τ s) using wfFTQ-elim(1) by metis
  thus ?thesis using wfFT-simple-wf[OF *] wf-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 $\Theta \vdash_{wf} \Phi'' \wedge \text{set } \Phi'' \subseteq \text{set } \Phi'$ **using** *wfPhi-elim3* *PConsNone* **by** *auto*
 ultimately show *?thesis* **using** *PConsNone* *wfPhi-elim* *wfT-simple-wf* **by** *auto*
 qed
 qed

lemma *wfPhi-f-simple-wfT*:
 fixes $\tau::\tau$ and $\Theta::\Theta$ and $\Phi::\Phi$ and $ft::\text{fun-ty-p-q}$
 assumes *Some* (*AF-fundef* f (*AF-fun-ty-none* (*AF-fun-ty* x b c τ s))) = *lookup-fun* Φ f and Θ
 $\vdash_{wf} \Phi$
 shows $\Theta ; \{||\} ; (x,b,c) \#_{\Gamma} GNil \vdash_{wf} \tau$
using *wfPhi-f-simple-wf* *assms* **using** *lookup-fun-member* **by** *blast*

lemma *wfPhi-f-simple-supb*:
 fixes $\tau::\tau$ and $\Theta::\Theta$ and $\Phi::\Phi$ and $ft::\text{fun-ty-p-q}$
 assumes *Some* (*AF-fundef* f (*AF-fun-ty-none* (*AF-fun-ty* x b c τ s))) = *lookup-fun* Φ f and Θ
 $\vdash_{wf} \Phi$
 shows *supp* b = $\{\}$
proof –
 have $\Theta ; \{||\} ; (x,b,c) \#_{\Gamma} GNil \vdash_{wf} \tau$ **using** *wfPhi-f-simple-wfT* *assms* **by** *auto*
 thus *?thesis* **using** *wfT-wf* *wfG-cons* *wfB-supb* **by** *fastforce*
 qed

lemma *wfPhi-f-simple-supp-t*:
 fixes $\tau::\tau$ and $\Theta::\Theta$ and $\Phi::\Phi$ and $ft::\text{fun-ty-p-q}$
 assumes *Some* (*AF-fundef* f (*AF-fun-ty-none* (*AF-fun-ty* x b c τ s))) = *lookup-fun* Φ f and Θ
 $\vdash_{wf} \Phi$
 shows *supp* $\tau \subseteq \{ \text{atom } x \}$
using *wfPhi-f-simple-wfT* *wfT-supp* *assms* **by** *fastforce*

lemma *wfPhi-f-simple-supp-c*:
 fixes $\tau::\tau$ and $\Theta::\Theta$ and $\Phi::\Phi$ and $ft::\text{fun-ty-p-q}$
 assumes *Some* (*AF-fundef* f (*AF-fun-ty-none* (*AF-fun-ty* x b c τ s))) = *lookup-fun* Φ f and Θ
 $\vdash_{wf} \Phi$
 shows *supp* $c \subseteq \{ \text{atom } x \}$
proof –
 have $\Theta ; \{||\} ; (x,b,c) \#_{\Gamma} GNil \vdash_{wf} \tau$ **using** *wfPhi-f-simple-wfT* *assms* **by** *auto*
 thus *?thesis* **using** *wfG-wfC* *wfC-supp* *wfT-wf* **by** *fastforce*
 qed

lemma *wfPhi-f-simple-supp-s*:
 fixes $\tau::\tau$ and $\Theta::\Theta$ and $\Phi::\Phi$ and $ft::\text{fun-ty-p-q}$
 assumes *Some* (*AF-fundef* f (*AF-fun-ty-none* (*AF-fun-ty* x b c τ s))) = *lookup-fun* Φ f and Θ
 $\vdash_{wf} \Phi$
 shows *supp* $s \subseteq \{ \text{atom } x \}$
proof –
 have *AF-fundef* f (*AF-fun-ty-none* (*AF-fun-ty* x b c τ s)) $\in \text{set } \Phi$ **using** *lookup-fun-member* *assms*
by *auto*
 hence *supp* $s \subseteq \{ \text{atom } x \}$ **using** *wfPhi-f-simple-wf* *assms* **by** *blast*

```

thus ?thesis using wf-suppl(3) atom-dom.simps toSet.simps x-not-in-u-set x-not-in-b-set setD.simps

using wf-suppl2(2) by fastforce
qed

lemma wfPhi-f-poly-wf:
  fixes  $\tau::\tau$  and  $\Theta::\Theta$  and  $\Phi::\Phi$  and  $ft::fun\text{-}typ\text{-}q$  and  $s::s$  and  $\Phi'::\Phi$ 
  assumes  $AF\text{-}fundef\ f\ (AF\text{-}fun\text{-}typ\text{-}some\ bv\ (AF\text{-}fun\text{-}typ\ x\ b\ c\ \tau\ s)) \in set\ \Phi$  and  $\Theta \vdash_{wf} \Phi$  and  $set\ \Phi \subseteq set\ \Phi'$  and  $\Theta \vdash_{wf} \Phi'$ 
  shows  $\Theta ; \{|bv|\} ; (x,b,c) \#_{\Gamma} GNil \vdash_{wf} \tau \wedge \Theta \vdash_{wf} \Phi' \wedge \Theta ; \Phi' ; \{|bv|\} \vdash_{wf} (AF\text{-}fun\text{-}typ\ x\ b\ c\ \tau\ s)$ 
using assms proof(induct  $\Phi$  rule:  $\Phi$ -induct)
  case PNil
  then show ?case by auto
next
  case (PConsNone  $f\ x\ b\ c\ \tau\ s'\ \Phi''$ )
  moreover have  $\Theta \vdash_{wf} \Phi'' \wedge set\ \Phi'' \subseteq set\ \Phi'$  using wfPhi-elimsl(3) PConsNone by auto
  ultimately show ?case using PConsNone wfPhi-elimsl wfFT-poly-wf by auto
next
  case (PConsSome  $f1\ bv1\ x1\ b1\ c1\ \tau1\ s1\ \Phi''$ )
  show ?case proof(cases  $f=f1$ )
  case True
  have  $AF\text{-}fun\text{-}typ\text{-}some\ bv1\ (AF\text{-}fun\text{-}typ\ x1\ b1\ c1\ \tau1\ s1) = AF\text{-}fun\text{-}typ\text{-}some\ bv\ (AF\text{-}fun\text{-}typ\ x\ b\ c\ \tau\ s)$ 
  by (metis PConsSome.premsl(1) PConsSome.premsl(2) True fun-def.eq-iff list.set-intros(1) option.inject wfPhi-lookup-fun-unique)
  hence  $*:\Theta ; \Phi'' \vdash_{wf} AF\text{-}fun\text{-}typ\text{-}some\ bv\ (AF\text{-}fun\text{-}typ\ x\ b\ c\ \tau\ s)$  using wfPhi-elimsl PConsSome
by metis
  thus ?thesis using wfFT-poly-wf * wb-phi-weakening PConsSome
  by (meson set-subset-Cons)
next
  case False
  hence  $AF\text{-}fundef\ f\ (AF\text{-}fun\text{-}typ\text{-}some\ bv\ (AF\text{-}fun\text{-}typ\ x\ b\ c\ \tau\ s)) \in set\ \Phi''$  using PConsSome
  by (meson fun-def.eq-iff set-ConsD)
  moreover have  $\Theta \vdash_{wf} \Phi'' \wedge set\ \Phi'' \subseteq set\ \Phi'$  using wfPhi-elimsl(3) PConsSome
  by (meson dual-order.trans set-subset-Cons)
  ultimately show ?thesis using PConsSome wfPhi-elimsl wfFT-poly-wf
  by blast
qed
qed

```

```

lemma wfPhi-f-poly-wfT:
  fixes  $\tau::\tau$  and  $\Theta::\Theta$  and  $\Phi::\Phi$  and  $ft::fun\text{-}typ\text{-}q$ 
  assumes  $Some\ (AF\text{-}fundef\ f\ (AF\text{-}fun\text{-}typ\text{-}some\ bv\ (AF\text{-}fun\text{-}typ\ x\ b\ c\ \tau\ s))) = lookup\text{-}fun\ \Phi\ f$  and  $\Theta \vdash_{wf} \Phi$ 
  shows  $\Theta ; \{|bv|\} ; (x,b,c) \#_{\Gamma} GNil \vdash_{wf} \tau$ 
using assms proof(induct  $\Phi$  rule:  $\Phi$ -induct)
  case PNil
  then show ?case by auto
next
  case (PConsSome  $f1\ bv1\ x1\ b1\ c1\ \tau1\ s'\ \Phi'$ )
  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  $\tau$ 1 s'))) #  $\Phi'$  f =
        Some (AF-fundef f1 (AF-fun-typ-some bv1 (AF-fun-typ x1 b1 c1  $\tau$ 1 s'))) using
          lookup-fun.simps using PConsSome.premis by simp
      then show ?thesis using PConsSome.premis wfPhi-elimis wfFT-poly-wfT
        by (metis option.inject)
    next
      case False
      then show ?thesis using PConsSome using lookup-fun.simps
        using wfPhi-elimis(3) by auto
    qed
  next
    case (PConsNone f' x' b' c'  $\tau'$  s'  $\Phi'$ )
    then show ?case proof(cases f'=f)
      case True
        then have *: $\Theta$  ;  $\Phi' \vdash_{wf} AF\text{-fun-typ-none}$  (AF-fun-typ x' b' c'  $\tau'$  s') using lookup-fun.simps
          PConsNone wfPhi-elimis by metis
        thus ?thesis using PConsNone wfFT-poly-wfT wfPhi-elimis lookup-fun.simps
          by (metis fun-def.eq-iff fun-typ-q.distinct(1) option.inject)
      next
        case False
        thus ?thesis using PConsNone wfPhi-elimis
          by (metis False lookup-fun.simps(2))
    qed
  qed

```

lemma wfPhi-f-poly-supp-b:

fixes $\tau::\tau$ and $\Theta::\Theta$ and $\Phi::\Phi$ and $ft::fun\text{-typ-q}$

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 $supp\ b \subseteq supp\ bv$

proof –

have $\Theta ; \{|bv|\} ; (x,b,c) \#_{\Gamma} GNil \vdash_{wf} \tau$ 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 $\tau::\tau$ and $\Theta::\Theta$ and $\Phi::\Phi$ and $ft::fun\text{-typ-q}$

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 $supp\ \tau \subseteq \{atom\ x, atom\ bv\}$

using wfPhi-f-poly-wfT[OF assms, THEN wfT-supp] atom-dom.simps supp-at-base by auto

lemma wfPhi-f-poly-supp-b-of-t:

fixes $\tau::\tau$ and $\Theta::\Theta$ and $\Phi::\Phi$ and $ft::fun\text{-typ-q}$

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 $supp\ (b\text{-of}\ \tau) \subseteq \{atom\ bv\}$

proof –

have $atom\ x \notin supp\ (b\text{-of}\ \tau)$ using x-fresh-b by auto

moreover have $supp\ (b\text{-of}\ \tau) \subseteq \{atom\ x, atom\ bv\}$ 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-poly-supp-c:*

fixes $\tau::\tau$ and $\Theta::\Theta$ and $\Phi::\Phi$ and $ft::fun\text{-}typ\text{-}q$
 assumes *Some (AF-fundef f (AF-fun-typ-some bv (AF-fun-typ x b c τ s))) = lookup-fun Φ f and Θ*
 $\vdash_{wf} \Phi$
 shows $supp\ c \subseteq \{atom\ x, atom\ bv\}$
proof –
 have $\Theta ; \{|bv|\} ; (x, b, c) \#_{\Gamma} GNil \vdash_{wf} \tau$ 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-poly-supp-s:*

fixes $\tau::\tau$ and $\Theta::\Theta$ and $\Phi::\Phi$ and $ft::fun\text{-}typ\text{-}q$
 assumes *Some (AF-fundef f (AF-fun-typ-some bv (AF-fun-typ x b c τ s))) = lookup-fun Φ f and Θ*
 $\vdash_{wf} \Phi$
 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 $*:\Theta ; \Phi ; \{|bv|\} \vdash_{wf} (AF-fun-typ\ x\ b\ c\ \tau\ s)$ using *assms wfPhi-f-poly-wf* by simp
 thus *?thesis* using *wfFT-wf-aux[OF *]* using *supp-at-base* by auto
 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-poly-supp-s wfPhi-f-simple-supp-s

lemma *fun-typ-eq-ret-unique:*

assumes $(AF-fun-typ\ x1\ b1\ c1\ \tau1'\ s1') = (AF-fun-typ\ x2\ b2\ c2\ \tau2'\ s2')$
 shows $\tau1'[x1::=v]_{\tau v} = \tau2'[x2::=v]_{\tau v}$
proof –
 have $[[atom\ x1]]lst.\ \tau1' = [[atom\ x2]]lst.\ \tau2'$ using *assms lst-fst fun-typ.eq-iff lst-snd* by metis
 thus *?thesis* using *subst-v-flip-eq-two[of x1 $\tau1'$ x2 $\tau2'$ 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 $(AF-fun-typ\ x1\ b1\ c1\ \tau1'\ s1') = (AF-fun-typ\ x2\ b2\ c2\ \tau2'\ s2')$
 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-fundef f* (*AF-fun-typ-none* (*AF-fun-typ* $x1\ b1\ c1\ \tau1'\ s1'$))) = *lookup-fun* $\Phi\ f$
and *Some* (*AF-fundef f* (*AF-fun-typ-none* (*AF-fun-typ* $x2\ b2\ c2\ \tau2'\ s2'$))) = *lookup-fun* $\Phi\ f$
shows $\tau1'[x1::=v]_{\tau v} = \tau2'[x2::=v]_{\tau v}$

proof –

have *: (*AF-fundef f* (*AF-fun-typ-none* (*AF-fun-typ* $x1\ b1\ c1\ \tau1'\ s1'$))) = (*AF-fundef f* (*AF-fun-typ-none* (*AF-fun-typ* $x2\ b2\ c2\ \tau2'\ s2'$)))) **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** $\tau1::\tau$ **and** $\tau2::\tau$
assumes $[[atom\ bv1]]lst. (AF-fun-typ\ x1\ b1\ c1\ \tau1\ s1) = [[atom\ bv2]]lst. (AF-fun-typ\ x2\ b2\ c2\ \tau2\ s2)$
(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** *: $atom\ c \# (b, b1, b2, c1, c2) \wedge atom\ c \# (bv1, bv2, AF-fun-typ\ x1\ b1\ c1\ \tau1\ s1, AF-fun-typ\ x2\ b2\ c2\ \tau2\ s2)$ **using** *obtain-fresh fresh-Pair* **by** *metis*

hence $(bv1 \leftrightarrow c) \cdot AF-fun-typ\ x1\ b1\ c1\ \tau1\ s1 = (bv2 \leftrightarrow c) \cdot AF-fun-typ\ x2\ b2\ c2\ \tau2\ s2$ **using** *Abs1-eq-iff-all(3)*[of $?x\ ?a\ ?y\ ?b$] *assms* **by** *metis*

hence $AF-fun-typ\ x1\ ((bv1 \leftrightarrow c) \cdot b1)\ ((bv1 \leftrightarrow c) \cdot c1)\ ((bv1 \leftrightarrow c) \cdot \tau1)\ ((bv1 \leftrightarrow c) \cdot s1) = AF-fun-typ\ x2\ ((bv2 \leftrightarrow c) \cdot b2)\ ((bv2 \leftrightarrow c) \cdot c2)\ ((bv2 \leftrightarrow c) \cdot \tau2)\ ((bv2 \leftrightarrow c) \cdot s2)$

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-fundef f* (*AF-fun-typ-some* $bv1\ (AF-fun-typ\ x1\ b1\ c1\ \tau1'\ s1')$))) = *lookup-fun* $\Phi\ f$
and *Some* (*AF-fundef f* (*AF-fun-typ-some* $bv2\ (AF-fun-typ\ x2\ b2\ c2\ \tau2'\ s2')$))) = *lookup-fun* $\Phi\ f$
shows $\tau1'[bv1::=b]_{\tau b}[x1::=v]_{\tau v} = \tau2'[bv2::=b]_{\tau b}[x2::=v]_{\tau v}$

proof –

have *: (*AF-fundef f* (*AF-fun-typ-some* $bv1\ (AF-fun-typ\ x1\ b1\ c1\ \tau1'\ s1')$))) = (*AF-fundef f* (*AF-fun-typ-some* $bv2\ (AF-fun-typ\ x2\ b2\ c2\ \tau2'\ s2')$))) **using** *option.inject assms* **by** *metis*

hence $AF-fun-typ-some\ bv1\ (AF-fun-typ\ x1\ b1\ c1\ \tau1'\ s1') = AF-fun-typ-some\ bv2\ (AF-fun-typ\ x2\ b2\ c2\ \tau2'\ s2')$

(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 $[[atom\ x1]]lst. \tau1'[bv1::=b]_{\tau b} = [[atom\ x2]]lst. \tau2'[bv2::=b]_{\tau b}$

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 τ1' s1'*))) = *lookup-fun* Φ *f* **and** *Some* (*AF-fundef* *f* (*AF-fun-typ-some* *bv2* (*AF-fun-typ* *x2 b2 c2 τ2' 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 τ1' s1'*))) = (*AF-fundef* *f* (*AF-fun-typ-some* *bv2* (*AF-fun-typ* *x2 b2 c2 τ2' s2'*)))

using *option.inject assms* **by** *metis*

hence *AF-fun-typ-some* *bv1* (*AF-fun-typ* *x1 b1 c1 τ1' s1'*) = *AF-fun-typ-some* *bv2* (*AF-fun-typ* *x2 b2 c2 τ2' 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 $\llbracket x' : b \mid c' \rrbracket = \llbracket x : b \mid c \rrbracket \wedge s'[x'::=v]_{sv} = s[x::=v]_{sv} \wedge \tau'[x'::=v]_{\tau v} = \tau[x::=v]_{\tau v}$

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 $\Gamma::\Gamma$ **and** $\Gamma':\Gamma$ **and** *v::v* **and** *e::e* **and** *c::c* **and** $\tau::\tau$ **and** *ts::(string*τ) list* **and** $\Delta::\Delta$ **and** *s::s* **and** *b::b* **and** $\mathcal{B}::\mathcal{B}$ **and** $\Phi::\Phi$ **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*: $\Theta; \mathcal{B}; \Gamma \vdash_{wf} v : b \implies \Theta; \mathcal{B} \vdash_{wf} b$ **and**

$\Theta; \mathcal{B}; \Gamma \vdash_{wf} c \implies \text{True}$ **and**

$\Theta; \mathcal{B} \vdash_{wf} \Gamma \implies \text{True}$ **and**

wfT-wfB: $\Theta; \mathcal{B}; \Gamma \vdash_{wf} \tau \implies \Theta; \mathcal{B} \vdash_{wf} b\text{-of } \tau$ **and**

$\Theta; \mathcal{B}; \Gamma \vdash_{wf} ts \implies \text{True}$ **and**

$\vdash_{wf} \Theta \implies \text{True}$ **and**

$\Theta; \mathcal{B} \vdash_{wf} b \implies \text{True}$ **and**

$wfCE-wfB: \Theta; \mathcal{B}; \Gamma \vdash_{wf} ce : b \implies \Theta; \mathcal{B} \vdash_{wf} b$ **and**
 $\Theta \vdash_{wf} td \implies True$
proof(*induct rule:wfV-wfC-wfG-wfT-wfTs-wfTh-wfB-wfCE-wfTD.inducts*)
case ($wfV-varI \Theta \mathcal{B} \Gamma b c x$)
hence $(x, b, c) \in toSet \Gamma$ **using** *lookup-iff wfV-wf* **using** *lookup-in-g* **by** *presburger*
hence $b \in fst'snd'toSet \Gamma$ **by** *force*
hence $wfB \Theta \mathcal{B} b$ **using** $wfG-wfB$ $wfV-varI$ **by** *metis*
then show $?case$ **using** $wfV-elim$ $wfG-wf$ $wf-intros$ **by** *metis*
next
case ($wfV-litI \Theta \Gamma l$)
moreover have $wfTh \Theta$ **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 ($wfV-pairI \Theta \Gamma v1 b1 v2 b2$)
then show $?case$ **using** $wfG-wf$ $wf-intros$ **by** *metis*
next
case ($wfV-consI s dclist \Theta dc x b c B \Gamma v$)
then show $?case$
using $wfV-wf$ $wfG-wf$ $wfB-consI$ **by** *metis*
next
case ($wfV-conspI s bv dclist \Theta dc x b' c \mathcal{B} b \Gamma v$)
then show $?case$
using $wfV-wf$ $wfG-wf$ **using** $wfB-appI$ **by** *metis*
next
case ($wfCE-valI \Theta \mathcal{B} \Gamma v b$)
then show $?case$ **using** $wfB-elim$ **by** *auto*
next
case ($wfCE-plusI \Theta \mathcal{B} \Gamma v1 v2$)
then show $?case$ **using** $wfB-elim$ **by** *auto*
next
case ($wfCE-leqI \Theta \mathcal{B} \Gamma v1 v2$)
then show $?case$ **using** $wfV-wf$ $wfG-wf$ $wf-intros$ $wfX-wfY$ **by** *metis*
next
case ($wfCE-fstI \Theta \mathcal{B} \Gamma v1 b1 b2$)
then show $?case$ **using** $wfB-elim$ **by** *metis*
next
case ($wfCE-sndI \Theta \mathcal{B} \Gamma v1 b1 b2$)
then show $?case$ **using** $wfB-elim$ **by** *metis*
next
case ($wfCE-concatI \Theta \mathcal{B} \Gamma v1 v2$)
then show $?case$ **using** $wfB-elim$ **by** *auto*
next
case ($wfCE-lenI \Theta \mathcal{B} \Gamma v1$)
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 $\Gamma::\Gamma$ **and** $\Gamma'::\Gamma$ **and** $v::v$ **and** $e::e$ **and** $c::c$ **and** $\tau::\tau$ **and** $ts::(string*\tau)$ *list* **and** $\Delta::\Delta$ **and** $s::s$
and $b::b$ **and** $\mathcal{B}::\mathcal{B}$ **and** $\Phi::\Phi$ **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

$wfE-wfB: \Theta; \Phi; \mathcal{B}; \Gamma; \Delta \vdash_{wf} e : b \implies \Theta; \mathcal{B} \vdash_{wf} b$ **and**

$wfS\text{-}wfB: \Theta; \Phi; \mathcal{B}; \Gamma; \Delta \vdash_{wf} s : b \implies \Theta; \mathcal{B} \vdash_{wf} b$ **and**
 $wfCS\text{-}wfB: \Theta; \Phi; \mathcal{B}; \Gamma; \Delta; tid; dc; t \vdash_{wf} cs : b \implies \Theta; \mathcal{B} \vdash_{wf} b$ **and**
 $wfCSS\text{-}wfB: \Theta; \Phi; \mathcal{B}; \Gamma; \Delta; tid; dclist \vdash_{wf} css : b \implies \Theta; \mathcal{B} \vdash_{wf} b$ **and**
 $\Theta \vdash_{wf} \Phi \implies True$ **and**
 $\Theta; \mathcal{B}; \Gamma \vdash_{wf} \Delta \implies True$ **and**
 $\Theta; \Phi \vdash_{wf} ftq \implies True$ **and**
 $\Theta; \Phi; \mathcal{B} \vdash_{wf} ft \implies \mathcal{B} \sqsubseteq \mathcal{B}' \implies \Theta; \Phi; \mathcal{B}' \vdash_{wf} ft$

proof(*induct rule:wfE-wfS-wfCS-wfCSS-wfPhi-wfD-wfFTQ-wfFT.inducts*)

case (*wfE-valI* $\Theta \Phi \mathcal{B} \Gamma \Delta v b$)

then show *?case* **using** *wfB-elim* *wfX-wfB1* **by** *metis*

next

case (*wfE-plusI* $\Theta \Phi \mathcal{B} \Gamma \Delta v1 v2$)

then show *?case* **using** *wfB-elim* *wfX-wfB1* **by** *metis*

next

case (*wfE-fstI* $\Theta \Phi \Gamma \Delta v1 b1 b2$)

then show *?case* **using** *wfB-elim* *wfX-wfB1* **by** *metis*

next

case (*wfE-sndI* $\Theta \Phi \Gamma \Delta v1 b1 b2$)

then show *?case* **using** *wfB-elim* *wfX-wfB1* **by** *metis*

next

case (*wfE-concatI* $\Theta \Phi \mathcal{B} \Gamma \Delta v1 v2$)

then show *?case* **using** *wfB-elim* *wfX-wfB1* **by** *metis*

next

case (*wfE-splitI* $\Theta \Phi \mathcal{B} \Gamma \Delta v1 v2$)

then show *?case* **using** *wfB-elim* *wfX-wfB1*

using *wfB-pairI* **by** *auto*

next

case (*wfE-lenI* $\Theta \Phi \mathcal{B} \Gamma \Delta v1$)

then show *?case* **using** *wfB-elim* *wfX-wfB1*

using *wfB-intI* *wfX-wfY1(1)* **by** *auto*

next

case (*wfE-appI* $\Theta \Phi \mathcal{B} \Gamma \Delta f x b c \tau s v$)

hence $\Theta; \mathcal{B}; (x, b, c) \#_{\Gamma} GNil \vdash_{wf} \tau$ **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-elim(2)*)

next

case (*wfE-appPI* $\Theta \Phi \mathcal{B} \Gamma \Delta b' bv v \tau f x b c s$)

hence $\Theta; \{ | bv | \}; (x, b, c) \#_{\Gamma} GNil \vdash_{wf} \tau$ **using** *wfPhi-f-poly-wfT* *wfX-wfY* **by** *blast*

then show *?case* **using** *wfE-appPI* *b-of.simps* **using** *wfT-b-weakening* *wfT-elim* *wfT-subst-wfB*

subst-b-b-def **by** *metis*

next

case (*wfE-mvarI* $\Theta \Phi \mathcal{B} \Gamma \Delta u \tau$)

hence $\Theta; \mathcal{B}; \Gamma \vdash_{wf} \tau$ **using** *wfD-wfT* **by** *fast*

then show *?case* **using** *wfT-elim* *b-of.simps* **by** *metis*

next

case (*wfFTNone* Θft)

then show *?case* **by** *auto*

next

case (*wfFTSome* $\Theta bv ft$)

then show *?case* **by** *auto*

next

case (*wfS-valI* $\Theta \Phi \mathcal{B} \Gamma v b \Delta$)

```

    then show ?case using wfX-wfB1 by auto
next
  case (wfS-letI  $\Theta \Phi \mathcal{B} \Gamma \Delta e b' x s b$ )
  then show ?case using wfX-wfB1 by auto
next
  case (wfS-let2I  $\Theta \Phi \mathcal{B} \Gamma \Delta s1 \tau x s2 b$ )
  then show ?case using wfX-wfB1 by auto
next
  case (wfS-ifI  $\Theta \mathcal{B} \Gamma v \Phi \Delta s1 b s2$ )
  then show ?case using wfX-wfB1 by auto
next
  case (wfS-varI  $\Theta \mathcal{B} \Gamma \tau v u \Phi \Delta b s$ )
  then show ?case using wfX-wfB1 by auto
next
  case (wfS-assignI  $u \tau \Delta \Theta \mathcal{B} \Gamma \Phi v$ )
  then show ?case using wfX-wfB1
    using wfB-unitI wfX-wfY2(5) by auto
next
  case (wfS-whileI  $\Theta \Phi \mathcal{B} \Gamma \Delta s1 s2 b$ )
  then show ?case using wfX-wfB1 by auto
next
  case (wfS-seqI  $\Theta \Phi \mathcal{B} \Gamma \Delta s1 s2 b$ )
  then show ?case using wfX-wfB1 by auto
next
  case (wfS-matchI  $\Theta \mathcal{B} \Gamma v tid dclist \Delta \Phi cs b$ )
  then show ?case using wfX-wfB1 by auto
next
  case (wfS-branchI  $\Theta \Phi \mathcal{B} x \tau \Gamma \Delta s b tid dc$ )
  then show ?case using wfX-wfB1 by auto
next
  case (wfS-finalI  $\Theta \Phi \mathcal{B} \Gamma \Delta tid dc t cs b$ )
  then show ?case using wfX-wfB1 by auto
next
  case (wfS-cons  $\Theta \Phi \mathcal{B} \Gamma \Delta tid dc t cs b dclist css$ )
  then show ?case using wfX-wfB1 by auto
next
  case (wfD-emptyI  $\Theta \mathcal{B} \Gamma$ )
  then show ?case using wfX-wfB1 by auto
next
  case (wfD-cons  $\Theta \mathcal{B} \Gamma \Delta \tau u$ )
  then show ?case using wfX-wfB1 by auto
next
  case (wfPhi-emptyI  $\Theta$ )
  then show ?case using wfX-wfB1 by auto
next
  case (wfPhi-consI  $f \Theta \Phi ft$ )
  then show ?case using wfX-wfB1 by auto
next
  case (wfFTI  $\Theta B b \Phi x c s \tau$ )
  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 $wfX\text{-}wfB = wfX\text{-}wfB1\ wfX\text{-}wfB2$

lemma $wf\text{-}weakening1$:

fixes $\Gamma::\Gamma$ **and** $\Gamma'::\Gamma$ **and** $v::v$ **and** $e::e$ **and** $c::c$ **and** $\tau::\tau$ **and** $ts::(string*\tau)$ **list** **and** $\Delta::\Delta$ **and** $s::s$ **and** $B::B$ **and** $ftq::fun\text{-}typ\text{-}q$ **and** $ft::fun\text{-}typ$ **and** $ce::ce$ **and** $td::type\text{-}def$ **and** $cs::branch\text{-}s$ **and** $css::branch\text{-}list$

shows $wfV\text{-}weakening$: $\Theta; \mathcal{B}; \Gamma \vdash_{wf} v : b \implies \Theta; \mathcal{B} \vdash_{wf} \Gamma' \implies toSet\ \Gamma \subseteq toSet\ \Gamma' \implies \Theta; \mathcal{B}; \Gamma' \vdash_{wf} v : b$ **and**

$wfC\text{-}weakening$: $\Theta; \mathcal{B}; \Gamma \vdash_{wf} c \implies \Theta; \mathcal{B} \vdash_{wf} \Gamma' \implies toSet\ \Gamma \subseteq toSet\ \Gamma' \implies \Theta; \mathcal{B}; \Gamma' \vdash_{wf} c$ **and**

$\Theta; \mathcal{B} \vdash_{wf} \Gamma \implies True$ **and**

$wfT\text{-}weakening$: $\Theta; \mathcal{B}; \Gamma \vdash_{wf} \tau \implies \Theta; \mathcal{B} \vdash_{wf} \Gamma' \implies toSet\ \Gamma \subseteq toSet\ \Gamma' \implies \Theta; \mathcal{B}; \Gamma' \vdash_{wf} \tau$ **and**

$\Theta; \mathcal{B}; \Gamma \vdash_{wf} ts \implies True$ **and**

$\vdash_{wf} P \implies True$ **and**

$wfB\text{-}weakening$: $wfB\ \Theta\ \mathcal{B}\ b \implies \mathcal{B} \mid\subseteq \mathcal{B}' \implies wfB\ \Theta\ \mathcal{B}\ b$ **and**

$wfCE\text{-}weakening$: $\Theta; \mathcal{B}; \Gamma \vdash_{wf} ce : b \implies \Theta; \mathcal{B} \vdash_{wf} \Gamma' \implies toSet\ \Gamma \subseteq toSet\ \Gamma' \implies \Theta; \mathcal{B}; \Gamma' \vdash_{wf} ce : b$ **and**

$\Theta \vdash_{wf} td \implies True$

proof($nominal\text{-}induct$

b **and** c **and** Γ **and** τ **and** ts **and** P **and** b **and** b **and** td

$avoiding$: Γ'

$rule$: $wfV\text{-}wfC\text{-}wfG\text{-}wfT\text{-}wfTs\text{-}wfTh\text{-}wfB\text{-}wfCE\text{-}wfTD$. $strong\text{-}induct$)

case ($wfV\text{-}varI\ \Theta\ \mathcal{B}\ \Gamma\ b\ c\ x$)

hence $Some\ (b, c) = lookup\ \Gamma'\ x$ **using** $lookup\text{-}weakening$ **by** $metis$

then show $?case$ **using** $Wellformed.wfV\text{-}varI\ wfV\text{-}varI$ **by** $metis$

next

case ($wfTI\ z\ \Theta\ \mathcal{B}\ \Gamma\ b\ c$)

show $?case$ **proof**

show $\langle atom\ z\ \# (\Theta, \mathcal{B}, \Gamma') \rangle$ **using** $wfTI$ **by** $auto$

show $\langle \Theta; \mathcal{B} \vdash_{wf} b \rangle$ **using** $wfTI$ **by** $auto$

have $*:toSet\ ((z, b, TRUE) \#_{\Gamma} \Gamma) \subseteq toSet\ ((z, b, TRUE) \#_{\Gamma} \Gamma')$ **using** $toSet.simps\ wfTI$ **by**

$auto$

thus $\langle \Theta; \mathcal{B}; (z, b, TRUE) \#_{\Gamma} \Gamma' \vdash_{wf} c \rangle$ **using** $wfTI(8)[OF - *]\ wfTI\ wfX\text{-}wfY$

by ($simp\ add$: $wfG\text{-}cons\text{-}TRUE$)

qed

next

case ($wfV\text{-}conspI\ s\ bv\ dclist\ \Theta\ dc\ x\ b'\ c\ \mathcal{B}\ b\ \Gamma\ v$)

show $?case$ **proof**

show $\langle AF\text{-}typedef\text{-}poly\ s\ bv\ dclist \in set\ \Theta \rangle$ **using** $wfV\text{-}conspI$ **by** $auto$

show $\langle (dc, \{x : b' \mid c\}) \in set\ dclist \rangle$ **using** $wfV\text{-}conspI$ **by** $auto$

show $\langle \Theta; \mathcal{B} \vdash_{wf} b \rangle$ **using** $wfV\text{-}conspI$ **by** $auto$

show $\langle atom\ bv\ \# (\Theta, \mathcal{B}, \Gamma', b, v) \rangle$ **using** $wfV\text{-}conspI$ **by** $simp$

show $\langle \Theta; \mathcal{B}; \Gamma' \vdash_{wf} v : b'[bv::=b]_{bb} \rangle$ **using** $wfV\text{-}conspI$ **by** $auto$

qed

qed($metis\ wf\text{-}intros$)+

lemma $wf\text{-}weakening2$:

fixes $\Gamma::\Gamma$ and $\Gamma'::\Gamma$ and $v::v$ and $e::e$ and $c::c$ and $\tau::\tau$ and $ts::(\text{string}*\tau)$ list and $\Delta::\Delta$ and $s::s$
and $\mathcal{B}::\mathcal{B}$ and $ftq::\text{fun-typ-}q$ and $ft::\text{fun-typ}$ and $ce::ce$ and $td::\text{type-def}$
and $cs::\text{branch-}s$ and $css::\text{branch-list}$
shows
 $wfE\text{-weakening}: \Theta; \Phi; \mathcal{B}; \Gamma; \Delta \vdash_{wf} e : b \implies \Theta; \mathcal{B} \vdash_{wf} \Gamma' \implies toSet \Gamma \subseteq toSet \Gamma' \implies \Theta; \Phi;$
 $\mathcal{B}; \Gamma'; \Delta \vdash_{wf} e : b$ **and**
 $wfS\text{-weakening}: \Theta; \Phi; \mathcal{B}; \Gamma; \Delta \vdash_{wf} s : b \implies \Theta; \mathcal{B} \vdash_{wf} \Gamma' \implies toSet \Gamma \subseteq toSet \Gamma' \implies \Theta; \Phi; \mathcal{B};$
 $\Gamma'; \Delta \vdash_{wf} s : b$ **and**
 $\Theta; \Phi; \mathcal{B}; \Gamma; \Delta; tid; dc; t \vdash_{wf} cs : b \implies \Theta; \mathcal{B} \vdash_{wf} \Gamma' \implies toSet \Gamma \subseteq toSet \Gamma' \implies \Theta; \Phi;$
 $\mathcal{B}; \Gamma'; \Delta; tid; dc; t \vdash_{wf} cs : b$ **and**
 $\Theta; \Phi; \mathcal{B}; \Gamma; \Delta; tid; dclist \vdash_{wf} css : b \implies \Theta; \mathcal{B} \vdash_{wf} \Gamma' \implies toSet \Gamma \subseteq toSet \Gamma' \implies \Theta; \Phi;$
 $\mathcal{B}; \Gamma'; \Delta; tid; dclist \vdash_{wf} css : b$ **and**
 $\Theta \vdash_{wf} (\Phi::\Phi) \implies True$ **and**
 $wfD\text{-weakning}: \Theta; \mathcal{B}; \Gamma \vdash_{wf} \Delta \implies \Theta; \mathcal{B} \vdash_{wf} \Gamma' \implies toSet \Gamma \subseteq toSet \Gamma' \implies \Theta; \mathcal{B}; \Gamma' \vdash_{wf} \Delta$
and
 $\Theta; \Phi \vdash_{wf} ftq \implies True$ **and**
 $\Theta; \Phi; \mathcal{B} \vdash_{wf} ft \implies True$
proof(*nominal-induct*
 b **and** b **and** b **and** b **and** Φ **and** Δ **and** ftq **and** ft
avoiding: Γ'
rule: $wfE\text{-}wfS\text{-}wfCS\text{-}wfCSS\text{-}wfPhi\text{-}wfD\text{-}wfFTQ\text{-}wfFT\text{-}strong\text{-induct}$)

case ($wfE\text{-appPI} \Theta \Phi \mathcal{B} \Gamma \Delta b' bv v \tau f x b c s$)
show *?case proof*
show $\langle \Theta \vdash_{wf} \Phi \rangle$ **using** $wfE\text{-appPI}$ **by** *auto*
show $\langle \Theta; \mathcal{B}; \Gamma' \vdash_{wf} \Delta \rangle$ **using** $wfE\text{-appPI}$ **by** *auto*
show $\langle \Theta; \mathcal{B} \vdash_{wf} b' \rangle$ **using** $wfE\text{-appPI}$ **by** *auto*
show $\langle atom bv \# (\Phi, \Theta, \mathcal{B}, \Gamma', \Delta, b', v, (b\text{-of } \tau)[bv::=b]_b) \rangle$ **using** $wfE\text{-appPI}$ **by** *auto*
show $\langle Some (AF\text{-fundef } f (AF\text{-fun-typ-some } bv (AF\text{-fun-typ } x b c \tau s))) = lookup\text{-fun } \Phi f \rangle$ **using**
 $wfE\text{-appPI}$ **by** *auto*
show $\langle \Theta; \mathcal{B}; \Gamma' \vdash_{wf} v : b[bv::=b]_b \rangle$ **using** $wfE\text{-appPI}$ $wf\text{-weakening1}$ **by** *auto*
qed
next
case ($wfS\text{-letI} \Theta \Phi \mathcal{B} \Gamma \Delta e b' x s b$)
show *?case proof*(*rule*)
show $\langle \Theta; \Phi; \mathcal{B}; \Gamma'; \Delta \vdash_{wf} e : b' \rangle$ **using** $wfS\text{-letI}$ **by** *auto*
have $toSet ((x, b', TRUE) \#_{\Gamma} \Gamma) \subseteq toSet ((x, b', TRUE) \#_{\Gamma} \Gamma')$ **using** $wfS\text{-letI}$ **by** *auto*
thus $\langle \Theta; \Phi; \mathcal{B}; (x, b', TRUE) \#_{\Gamma} \Gamma'; \Delta \vdash_{wf} s : b \rangle$ **using** $wfS\text{-letI}$ **by** (*meson* $wfG\text{-cons}$
 $wfG\text{-cons-TRUE } wfS\text{-wf}$)
show $\langle \Theta; \mathcal{B}; \Gamma' \vdash_{wf} \Delta \rangle$ **using** $wfS\text{-letI}$ **by** *auto*
show $\langle atom x \# (\Phi, \Theta, \mathcal{B}, \Gamma', \Delta, e, b) \rangle$ **using** $wfS\text{-letI}$ **by** *auto*
qed
next
case ($wfS\text{-let2I} \Theta \Phi \mathcal{B} \Gamma \Delta s1 \tau x s2 b$)
show *?case proof*
show $\langle \Theta; \Phi; \mathcal{B}; \Gamma'; \Delta \vdash_{wf} s1 : b\text{-of } \tau \rangle$ **using** $wfS\text{-let2I}$ **by** *auto*
show $\langle \Theta; \mathcal{B}; \Gamma' \vdash_{wf} \tau \rangle$ **using** $wfS\text{-let2I}$ $wf\text{-weakening1}$ **by** *auto*
have $toSet ((x, b\text{-of } \tau, TRUE) \#_{\Gamma} \Gamma) \subseteq toSet ((x, b\text{-of } \tau, TRUE) \#_{\Gamma} \Gamma')$ **using** $wfS\text{-let2I}$ **by**
auto
thus $\langle \Theta; \Phi; \mathcal{B}; (x, b\text{-of } \tau, TRUE) \#_{\Gamma} \Gamma'; \Delta \vdash_{wf} s2 : b \rangle$ **using** $wfS\text{-let2I}$ **by** (*meson* $wfG\text{-cons}$
 $wfG\text{-cons-TRUE } wfS\text{-wf}$)
show $\langle atom x \# (\Phi, \Theta, \mathcal{B}, \Gamma', \Delta, s1, b, \tau) \rangle$ **using** $wfS\text{-let2I}$ **by** *auto*

```

qed
next
case (wfS-varI  $\Theta \mathcal{B} \Gamma \tau v u \Phi \Delta b s$ )
show ?case proof
  show  $\Theta; \mathcal{B}; \Gamma' \vdash_{wf} \tau$  using wfS-varI wf-weakening1 by auto
  show  $\Theta; \mathcal{B}; \Gamma' \vdash_{wf} v : b\text{-of } \tau$  using wfS-varI wf-weakening1 by auto
  show  $\text{atom } u \# (\Phi, \Theta, \mathcal{B}, \Gamma', \Delta, \tau, v, b)$  using wfS-varI by auto
  show  $\Theta; \Phi; \mathcal{B}; \Gamma'; (u, \tau) \#_{\Delta} \Delta \vdash_{wf} s : b$  using wfS-varI by auto
qed
next
case (wfS-branchI  $\Theta \Phi \mathcal{B} x \tau \Gamma \Delta s b \text{tid } dc$ )
show ?case proof
  have  $\text{toSet } ((x, b\text{-of } \tau, \text{TRUE}) \#_{\Gamma} \Gamma) \subseteq \text{toSet } ((x, b\text{-of } \tau, \text{TRUE}) \#_{\Gamma} \Gamma')$  using wfS-branchI
by auto
  thus  $\langle \Theta; \Phi; \mathcal{B}; (x, b\text{-of } \tau, \text{TRUE}) \#_{\Gamma} \Gamma'; \Delta \vdash_{wf} s : b \rangle$  using wfS-branchI by (meson
wfG-cons wfG-cons-TRUE wfS-wf)
  show  $\langle \text{atom } x \# (\Phi, \Theta, \mathcal{B}, \Gamma', \Delta, \Gamma', \tau) \rangle$  using wfS-branchI by auto
  show  $\langle \Theta; \mathcal{B}; \Gamma' \vdash_{wf} \Delta \rangle$  using wfS-branchI by auto
qed
next
case (wfS-finalI  $\Theta \Phi \mathcal{B} \Gamma \Delta \text{tid } dclist' cs b dclist$ )
then show ?case using wf-intros by metis
next
case (wfS-cons  $\Theta \Phi \mathcal{B} \Gamma \Delta \text{tid } dclist' cs b css dclist$ )
then show ?case using wf-intros by metis
next
case (wfS-assertI  $\Theta \Phi \mathcal{B} x c \Gamma \Delta s b$ )
show ?case proof(rule)
show  $\langle \Theta; \mathcal{B}; \Gamma' \vdash_{wf} c \rangle$  using wfS-assertI wf-weakening1 by auto
  have  $\Theta; \mathcal{B} \vdash_{wf} (x, B\text{-bool}, c) \#_{\Gamma} \Gamma'$  proof(rule wfG-consI)
  show  $\langle \Theta; \mathcal{B} \vdash_{wf} \Gamma' \rangle$  using wfS-assertI by auto
  show  $\langle \text{atom } x \# \Gamma' \rangle$  using wfS-assertI by auto
  show  $\langle \Theta; \mathcal{B} \vdash_{wf} B\text{-bool} \rangle$  using wfS-assertI wfB-boolI wfX-wfY by metis
  have  $\Theta; \mathcal{B} \vdash_{wf} (x, B\text{-bool}, \text{TRUE}) \#_{\Gamma} \Gamma'$  proof
  show  $(\text{TRUE}) \in \{\text{TRUE}, \text{FALSE}\}$  by auto
  show  $\langle \Theta; \mathcal{B} \vdash_{wf} \Gamma' \rangle$  using wfS-assertI by auto
  show  $\langle \text{atom } x \# \Gamma' \rangle$  using wfS-assertI by auto
  show  $\langle \Theta; \mathcal{B} \vdash_{wf} B\text{-bool} \rangle$  using wfS-assertI wfB-boolI wfX-wfY by metis
qed
thus  $\langle \Theta; \mathcal{B}; (x, B\text{-bool}, \text{TRUE}) \#_{\Gamma} \Gamma' \vdash_{wf} c \rangle$ 
using wf-weakening1(2)[OF  $\langle \Theta; \mathcal{B}; \Gamma' \vdash_{wf} c \rangle \langle \Theta; \mathcal{B} \vdash_{wf} (x, B\text{-bool}, \text{TRUE}) \#_{\Gamma} \Gamma' \rangle$ ] by force
qed

  thus  $\langle \Theta; \Phi; \mathcal{B}; (x, B\text{-bool}, c) \#_{\Gamma} \Gamma'; \Delta \vdash_{wf} s : b \rangle$  using wfS-assertI by fastforce

  show  $\langle \Theta; \mathcal{B}; \Gamma' \vdash_{wf} \Delta \rangle$  using wfS-assertI by auto
  show  $\langle \text{atom } x \# (\Phi, \Theta, \mathcal{B}, \Gamma', \Delta, c, b, s) \rangle$  using wfS-assertI by auto
qed

qed(metis wf-intros wf-weakening1)+

lemmas wf-weakening = wf-weakening1 wf-weakening2

```

lemma *wfV-weakening-cons*:

fixes $\Gamma::\Gamma$ **and** $\Gamma':\Gamma$ **and** $v::v$ **and** $c::c$

assumes $\Theta; \mathcal{B}; \Gamma \vdash_{wf} v : b$ **and** $atom\ y \# \Gamma$ **and** $\Theta; \mathcal{B}; ((y, b', TRUE) \#_{\Gamma} \Gamma) \vdash_{wf} c$

shows $\Theta; \mathcal{B}; (y, b', c) \#_{\Gamma} \Gamma \vdash_{wf} v : b$

proof –

have $wfG\ \Theta\ \mathcal{B}\ ((y, b', c) \#_{\Gamma} \Gamma)$ **using** *wfG-intros2* **assms** **by** *auto*

moreover **have** $toSet\ \Gamma \subseteq toSet\ ((y, b', c) \#_{\Gamma} \Gamma)$ **using** *toSet.simps* **by** *auto*

ultimately **show** *?thesis* **using** *wf-weakening* **using** *assms(1)* **by** *blast*

qed

lemma *wfG-cons-weakening*:

fixes $\Gamma':\Gamma$

assumes $\Theta; \mathcal{B} \vdash_{wf} ((x, b, c) \#_{\Gamma} \Gamma)$ **and** $\Theta; \mathcal{B} \vdash_{wf} \Gamma'$ **and** $toSet\ \Gamma \subseteq toSet\ \Gamma'$ **and** $atom\ x \# \Gamma'$

shows $\Theta; \mathcal{B} \vdash_{wf} ((x, b, c) \#_{\Gamma} \Gamma')$

proof(*cases* $c \in \{TRUE, FALSE\}$)

case *True*

then **show** *?thesis* **using** *wfG-wfB* *wfG-cons2I* **assms** **by** *auto*

next

case *False*

hence $\Theta; \mathcal{B} \vdash_{wf} \Gamma \wedge atom\ x \# \Gamma \wedge \Theta; \mathcal{B}; (x, b, TRUE) \#_{\Gamma} \Gamma \vdash_{wf} c$

using *wfG-elim(2)*[*OF* *assms(1)*] **by** *auto*

have $a1:\Theta; \mathcal{B} \vdash_{wf} (x, b, TRUE) \#_{\Gamma} \Gamma'$ **using** *wfG-wfB* *wfG-cons2I* **assms** **by** *simp*

moreover **have** $a2:toSet\ ((x, b, TRUE) \#_{\Gamma} \Gamma) \subseteq toSet\ ((x, b, TRUE) \#_{\Gamma} \Gamma')$ **using** *toSet.simps*

assms **by** *blast*

moreover **have** $\Theta; \mathcal{B} \vdash_{wf} (x, b, TRUE) \#_{\Gamma} \Gamma'$ **proof**

show $(TRUE) \in \{TRUE, FALSE\}$ **by** *auto*

show $\Theta; \mathcal{B} \vdash_{wf} \Gamma'$ **using** *assms* **by** *auto*

show $atom\ x \# \Gamma'$ **using** *assms* **by** *auto*

show $\Theta; \mathcal{B} \vdash_{wf} b$ **using** *assms* *wfG-elim* **by** *metis*

qed

hence $\Theta; \mathcal{B}; (x, b, TRUE) \#_{\Gamma} \Gamma' \vdash_{wf} c$ **using** *wf-weakening* $a1\ a2\ *$ **by** *auto*

then **show** *?thesis* **using** *wfG-cons1I*[*of* $c\ \Theta\ \mathcal{B}\ \Gamma'\ x\ b,\ OF\ False$] *wfG-wfB* **assms** **by** *simp*

qed

lemma *wfT-weakening-aux*:

fixes $\Gamma::\Gamma$ **and** $\Gamma':\Gamma$ **and** $c::c$

assumes $\Theta; \mathcal{B}; \Gamma \vdash_{wf} \llbracket z : b \mid c \rrbracket$ **and** $\Theta; \mathcal{B} \vdash_{wf} \Gamma'$ **and** $toSet\ \Gamma \subseteq toSet\ \Gamma'$ **and** $atom\ z \# \Gamma'$

shows $\Theta; \mathcal{B}; \Gamma' \vdash_{wf} \llbracket z : b \mid c \rrbracket$

proof

show $\langle atom\ z \# (\Theta, \mathcal{B}, \Gamma') \rangle$

using *wf-sup* *wfX-wfY* *assms* *fresh-prodN* *fresh-def* *x-not-in-b-set* *wfG-fresh-x* **by** *metis*

show $\langle \Theta; \mathcal{B} \vdash_{wf} b \rangle$ **using** *assms* *wfT-elim* **by** *metis*

show $\langle \Theta; \mathcal{B}; (z, b, TRUE) \#_{\Gamma} \Gamma' \vdash_{wf} c \rangle$ **proof** –

have $\Theta; \mathcal{B}; (z, b, TRUE) \#_{\Gamma} \Gamma \vdash_{wf} c$ **using** *wfT-wfC* *fresh-weakening* *assms* **by** *auto*

moreover **have** $a1:\Theta; \mathcal{B} \vdash_{wf} (z, b, TRUE) \#_{\Gamma} \Gamma'$ **using** *wfG-cons2I* *assms* $\langle \Theta; \mathcal{B} \vdash_{wf} b \rangle$ **by**

simp

moreover **have** $a2:toSet\ ((z, b, TRUE) \#_{\Gamma} \Gamma) \subseteq toSet\ ((z, b, TRUE) \#_{\Gamma} \Gamma')$ **using** *toSet.simps*

assms **by** *blast*

moreover **have** $\Theta; \mathcal{B} \vdash_{wf} (z, b, TRUE) \#_{\Gamma} \Gamma'$ **proof**

show $(TRUE) \in \{TRUE, FALSE\}$ **by** *auto*

show $\Theta; \mathcal{B} \vdash_{wf} \Gamma'$ **using** *assms* **by** *auto*

```

    show atom z #  $\Gamma'$  using assms by auto
    show  $\Theta; \mathcal{B} \vdash_{wf} b$  using assms wfT-elim by metis
  qed
  thus ?thesis using wf-weakening a1 a2 * by auto
  qed
  qed

```

```

lemma wfT-weakening-all:
  fixes  $\Gamma::\Gamma$  and  $\Gamma':\Gamma$  and  $\tau::\tau$ 
  assumes  $\Theta; \mathcal{B}; \Gamma \vdash_{wf} \tau$  and  $\Theta; \mathcal{B}' \vdash_{wf} \Gamma'$  and  $toSet \Gamma \subseteq toSet \Gamma'$  and  $\mathcal{B} \sqsubseteq \mathcal{B}'$ 
  shows  $\Theta; \mathcal{B}'; \Gamma' \vdash_{wf} \tau$ 
  using wb-b-weakening assms wfT-weakening by metis

```

```

lemma wfT-weakening-nil:
  fixes  $\Gamma::\Gamma$  and  $\Gamma':\Gamma$  and  $\tau::\tau$ 
  assumes  $\Theta; \{||\}; GNil \vdash_{wf} \tau$  and  $\Theta; \mathcal{B}' \vdash_{wf} \Gamma'$ 
  shows  $\Theta; \mathcal{B}'; \Gamma' \vdash_{wf} \tau$ 
  using wfT-weakening-all
  using assms(1) assms(2) toSet.simps(1) by blast

```

```

lemma wfTh-wfT2:
  fixes  $x::x$  and  $v::v$  and  $\tau::\tau$  and  $G::\Gamma$ 
  assumes wfTh  $\Theta$  and AF-typedef s dclist  $\in set \Theta$  and
     $(dc, \tau) \in set dclist$  and  $\Theta; B \vdash_{wf} G$ 
  shows  $supp \tau = \{\}$  and  $\tau[x::=v]_{\tau v} = \tau$  and wfT  $\Theta B G \tau$ 
proof -
  show  $supp \tau = \{\}$  proof(rule ccontr)
    assume a1:  $supp \tau \neq \{\}$ 
    have  $supp \Theta \neq \{\}$  proof -
      obtain dclist where dc: AF-typedef s dclist  $\in set \Theta \wedge (dc, \tau) \in set dclist$ 
      using assms by auto
      hence  $supp (dc, \tau) \neq \{\}$ 
      using a1 by (simp add: supp-Pair)
      hence  $supp dclist \neq \{\}$  using dc supp-list-member by auto
      hence  $supp (AF-typedef s dclist) \neq \{\}$  using type-def.supp by auto
      thus ?thesis using supp-list-member dc by auto
    qed
    thus False using assms wfTh-sup by simp
  qed
  thus  $\tau[x::=v]_{\tau v} = \tau$  by (simp add: fresh-def)
  have wfT  $\Theta \{||\} GNil \tau$  using assms wfTh-wfT by auto
  thus wfT  $\Theta B G \tau$  using assms wfT-weakening-nil by simp

```

```

lemma wf-d-weakening:
  fixes  $\Gamma::\Gamma$  and  $\Gamma':\Gamma$  and  $v::v$  and  $e::e$  and  $c::c$  and  $\tau::\tau$  and  $ts::(string*\tau)$  list and  $\Delta::\Delta$  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
 $\Theta; \Phi; \mathcal{B}; \Gamma; \Delta \vdash_{wf} e : b \implies \Theta; \mathcal{B}; \Gamma \vdash_{wf} \Delta' \implies setD \Delta \subseteq setD \Delta' \implies \Theta; \Phi; \mathcal{B}; \Gamma; \Delta' \vdash_{wf} e : b$ **and**
 $\Theta; \Phi; \mathcal{B}; \Gamma; \Delta \vdash_{wf} s : b \implies \Theta; \mathcal{B}; \Gamma \vdash_{wf} \Delta' \implies setD \Delta \subseteq setD \Delta' \implies \Theta; \Phi; \mathcal{B}; \Gamma; \Delta' \vdash_{wf} s : b$ **and**
 $\Theta; \Phi; \mathcal{B}; \Gamma; \Delta; tid; dc; t \vdash_{wf} cs : b \implies \Theta; \mathcal{B}; \Gamma \vdash_{wf} \Delta' \implies setD \Delta \subseteq setD \Delta' \implies \Theta; \Phi; \mathcal{B}; \Gamma; \Delta'; tid; dc; t \vdash_{wf} cs : b$ **and**
 $\Theta; \Phi; \mathcal{B}; \Gamma; \Delta; tid; dclist \vdash_{wf} css : b \implies \Theta; \mathcal{B}; \Gamma \vdash_{wf} \Delta' \implies setD \Delta \subseteq setD \Delta' \implies \Theta; \Phi; \mathcal{B}; \Gamma; \Delta'; tid; dclist \vdash_{wf} css : b$ **and**
 $\Theta \vdash_{wf} (\Phi :: \Phi) \implies True$ **and**
 $\Theta; \mathcal{B}; \Gamma \vdash_{wf} \Delta \implies True$ **and**
 $\Theta; \Phi \vdash_{wf} ftq \implies True$ **and**
 $\Theta; \Phi; \mathcal{B} \vdash_{wf} ft \implies True$
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* $\Theta \Phi \mathcal{B} \Gamma \Delta v b$)
then show *?case* **using** *wf-intros* **by** *metis*
next
case (*wfE-plusI* $\Theta \Phi \mathcal{B} \Gamma \Delta v1 v2$)
then show *?case* **using** *wf-intros* **by** *metis*
next
case (*wfE-leqI* $\Theta \Phi \mathcal{B} \Gamma \Delta v1 v2$)
then show *?case* **using** *wf-intros* **by** *metis*
next
case (*wfE-fstI* $\Theta \Phi \mathcal{B} \Gamma \Delta v1 b1 b2$)
then show *?case* **using** *wf-intros* **by** *metis*
next
case (*wfE-sndI* $\Theta \Phi \mathcal{B} \Gamma \Delta v1 b1 b2$)
then show *?case* **using** *wf-intros* **by** *metis*
next
case (*wfE-concatI* $\Theta \Phi \mathcal{B} \Gamma \Delta v1 v2$)
then show *?case* **using** *wf-intros* **by** *metis*
next
case (*wfE-splitI* $\Theta \Phi \mathcal{B} \Gamma \Delta v1 v2$)
then show *?case* **using** *wf-intros* **by** *metis*
next
case (*wfE-lenI* $\Theta \Phi \mathcal{B} \Gamma \Delta v1$)
then show *?case* **using** *wf-intros* **by** *metis*
next
case (*wfE-appI* $\Theta \Phi \mathcal{B} \Gamma \Delta f x b c \tau s v$)
then show *?case* **using** *wf-intros* **by** *metis*
next
case (*wfE-appPI* $\Theta \Phi \mathcal{B} \Gamma \Delta b' bv v \tau f x b c s$)
show *?case* **proof**(*rule*, (*rule* *wfE-appPI*)**+**)
show $\langle atom \ bv \ \# \ (\Phi, \Theta, \mathcal{B}, \Gamma, \Delta', b', v, (b-of \ \tau)[bv::=b]_b) \rangle$ **using** *wfE-appPI* **by** *auto*
show $\langle Some \ (AF-fundef \ f \ (AF-fun-typ-some \ bv \ (AF-fun-typ \ x \ b \ c \ \tau \ s))) = lookup-fun \ \Phi \ f \rangle$ **using**
wfE-appPI **by** *auto*
show $\langle \Theta; \mathcal{B}; \Gamma \vdash_{wf} v : b[bv::=b]_b \rangle$ **using** *wfE-appPI* **by** *auto*
qed
next


```

case (wfE-mvarI  $\Theta \Phi \mathcal{B} \Gamma \Delta u \tau$ )
show ?case proof
  show  $\langle \Theta \vdash_{wf} \Phi \rangle$  using wfE-mvarI by auto
  show  $\langle \Theta; \mathcal{B}; \Gamma \vdash_{wf} \Delta' \rangle$  using wfE-mvarI by auto
  show  $\langle (u, \tau) \in \text{setD } \Delta' \rangle$  using wfE-mvarI by auto
qed
next
case (wfS-valI  $\Theta \Phi \mathcal{B} \Gamma v b \Delta$ )
then show ?case using wf-intros by metis
next
case (wfS-letI  $\Theta \Phi \mathcal{B} \Gamma \Delta e b' x s b$ )
show ?case proof(rule)
  show  $\langle \Theta; \Phi; \mathcal{B}; \Gamma; \Delta' \vdash_{wf} e : b' \rangle$  using wfS-letI by auto
  have  $\Theta; \mathcal{B} \vdash_{wf} (x, b', \text{TRUE}) \#_{\Gamma} \Gamma$  using wfG-cons2I wfX-wfY wfS-letI by metis
  hence  $\Theta; \mathcal{B}; (x, b', \text{TRUE}) \#_{\Gamma} \Gamma \vdash_{wf} \Delta'$  using wf-weakening2(6) wfS-letI by force
  thus  $\langle \Theta; \Phi; \mathcal{B}; (x, b', \text{TRUE}) \#_{\Gamma} \Gamma; \Delta' \vdash_{wf} s : b \rangle$  using wfS-letI by metis
  show  $\langle \Theta; \mathcal{B}; \Gamma \vdash_{wf} \Delta' \rangle$  using wfS-letI by auto
  show  $\langle \text{atom } x \# (\Phi, \Theta, \mathcal{B}, \Gamma, \Delta', e, b) \rangle$  using wfS-letI by auto
qed
next
case (wfS-assertI  $\Theta \Phi \mathcal{B} x c \Gamma \Delta s b$ )
show ?case proof
  have  $\Theta; \mathcal{B}; (x, B\text{-bool}, c) \#_{\Gamma} \Gamma \vdash_{wf} \Delta'$  proof(rule wf-weakening2(6))
  show  $\langle \Theta; \mathcal{B}; \Gamma \vdash_{wf} \Delta' \rangle$  using wfS-assertI by auto
next
show  $\langle \Theta; \mathcal{B} \vdash_{wf} (x, B\text{-bool}, c) \#_{\Gamma} \Gamma \rangle$  using wfS-assertI wfX-wfY by metis
next
show  $\langle \text{toSet } \Gamma \subseteq \text{toSet } ((x, B\text{-bool}, c) \#_{\Gamma} \Gamma) \rangle$  using wfS-assertI by auto
qed
thus  $\langle \Theta; \Phi; \mathcal{B}; (x, B\text{-bool}, c) \#_{\Gamma} \Gamma; \Delta' \vdash_{wf} s : b \rangle$  using wfS-assertI wfX-wfY by metis
next
show  $\langle \Theta; \mathcal{B}; \Gamma \vdash_{wf} c \rangle$  using wfS-assertI by auto
next
show  $\langle \Theta; \mathcal{B}; \Gamma \vdash_{wf} \Delta' \rangle$  using wfS-assertI by auto
next
show  $\langle \text{atom } x \# (\Phi, \Theta, \mathcal{B}, \Gamma, \Delta', c, b, s) \rangle$  using wfS-assertI by auto
qed
next
case (wfS-let2I  $\Theta \Phi \mathcal{B} \Gamma \Delta s1 \tau x s2 b$ )
show ?case proof
  show  $\langle \Theta; \Phi; \mathcal{B}; \Gamma; \Delta' \vdash_{wf} s1 : b\text{-of } \tau \rangle$  using wfS-let2I by auto
  show  $\langle \Theta; \mathcal{B}; \Gamma \vdash_{wf} \tau \rangle$  using wfS-let2I by auto
  have  $\Theta; \mathcal{B} \vdash_{wf} (x, b\text{-of } \tau, \text{TRUE}) \#_{\Gamma} \Gamma$  using wfG-cons2I wfX-wfY wfS-let2I by metis
  hence  $\Theta; \mathcal{B}; (x, b\text{-of } \tau, \text{TRUE}) \#_{\Gamma} \Gamma \vdash_{wf} \Delta'$  using wf-weakening2(6) wfS-let2I by force
  thus  $\langle \Theta; \Phi; \mathcal{B}; (x, b\text{-of } \tau, \text{TRUE}) \#_{\Gamma} \Gamma; \Delta' \vdash_{wf} s2 : b \rangle$  using wfS-let2I by metis
  show  $\langle \text{atom } x \# (\Phi, \Theta, \mathcal{B}, \Gamma, \Delta', s1, b, \tau) \rangle$  using wfS-let2I by auto
qed
next
case (wfS-ifI  $\Theta \mathcal{B} \Gamma v \Phi \Delta s1 b s2$ )
then show ?case using wf-intros by metis
next
case (wfS-varI  $\Theta \mathcal{B} \Gamma \tau v u \Phi \Delta b s$ )

```

```

show ?case proof
  show  $\langle \Theta; \mathcal{B}; \Gamma \vdash_{wf} \tau \rangle$  using wfS-varI by auto
  show  $\langle \Theta; \mathcal{B}; \Gamma \vdash_{wf} v : b\text{-of } \tau \rangle$  using wfS-varI by auto
  show  $\langle atom\ u \ \# \ (\Phi, \Theta, \mathcal{B}, \Gamma, \Delta', \tau, v, b) \rangle$  using wfS-varI setD.simps by auto
  have  $\Theta; \mathcal{B}; \Gamma \vdash_{wf} (u, \tau) \#_{\Delta} \Delta'$  using wfS-varI wfD-cons setD.simps u-fresh-d by metis
  thus  $\langle \Theta; \Phi; \mathcal{B}; \Gamma; (u, \tau) \#_{\Delta} \Delta' \vdash_{wf} s : b \rangle$  using wfS-varI setD.simps by blast
qed
next
case (wfS-assignI u  $\tau$   $\Delta$   $\Theta$   $\mathcal{B}$   $\Gamma$   $\Phi$  v)
show ?case proof
  show  $\langle (u, \tau) \in setD\ \Delta' \rangle$  using wfS-assignI setD.simps by auto
  show  $\langle \Theta; \mathcal{B}; \Gamma \vdash_{wf} \Delta' \rangle$  using wfS-assignI by auto
  show  $\langle \Theta \vdash_{wf} \Phi \rangle$  using wfS-assignI by auto
  show  $\langle \Theta; \mathcal{B}; \Gamma \vdash_{wf} v : b\text{-of } \tau \rangle$  using wfS-assignI by auto
qed
next
case (wfS-whileI  $\Theta$   $\Phi$   $\mathcal{B}$   $\Gamma$   $\Delta$  s1 s2 b)
then show ?case using wf-intros by metis
next
case (wfS-seqI  $\Theta$   $\Phi$   $\mathcal{B}$   $\Gamma$   $\Delta$  s1 s2 b)
then show ?case using wf-intros by metis
next
case (wfS-matchI  $\Theta$   $\mathcal{B}$   $\Gamma$  v tid dclist  $\Delta$   $\Phi$  cs b)
then show ?case using wf-intros by metis
next
case (wfS-branchI  $\Theta$   $\Phi$   $\mathcal{B}$  x  $\tau$   $\Gamma$   $\Delta$  s b tid dc)
show ?case proof
  have  $\Theta; \mathcal{B} \vdash_{wf} (x, b\text{-of } \tau, TRUE) \#_{\Gamma} \Gamma$  using wfG-cons2I wfX-wfY wfS-branchI by metis
  hence  $\Theta; \mathcal{B}; (x, b\text{-of } \tau, TRUE) \#_{\Gamma} \Gamma \vdash_{wf} \Delta'$  using wf-weakening2(6) wfS-branchI by force
  thus  $\langle \Theta; \Phi; \mathcal{B}; (x, b\text{-of } \tau, TRUE) \#_{\Gamma} \Gamma; \Delta' \vdash_{wf} s : b \rangle$  using wfS-branchI by simp
  show  $\langle atom\ x \ \# \ (\Phi, \Theta, \mathcal{B}, \Gamma, \Delta', \Gamma, \tau) \rangle$  using wfS-branchI by auto
  show  $\langle \Theta; \mathcal{B}; \Gamma \vdash_{wf} \Delta' \rangle$  using wfS-branchI by auto
qed
next
case (wfS-finalI  $\Theta$   $\Phi$   $\mathcal{B}$   $\Gamma$   $\Delta$  tid dclist' cs b dclist)
then show ?case using wf-intros by metis
next
case (wfS-cons  $\Theta$   $\Phi$   $\mathcal{B}$   $\Gamma$   $\Delta$  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** $\Gamma::\Gamma$

assumes $\Theta; \mathcal{B}; \Gamma \vdash_{wf} ce : b$ **and** $atom\ x \ \# \ \Gamma$

shows $\Theta; \mathcal{B}; ((x, b, TRUE) \#_{\Gamma} \Gamma) \vdash_{wf} (CE\text{-val } (V\text{-var } x) == ce)$

proof –

have $\Theta; \mathcal{B} \vdash_{wf} b$ **using** assms wfX-wfB **by** auto

hence wbg: $\Theta; \mathcal{B} \vdash_{wf} \Gamma$ **using** wfX-wfY assms **by** auto

```

show ?thesis proof
  show *:Θ ; B ; (x, b, TRUE) #Γ Γ ⊢wf CE-val (V-var x) : b
  proof(rule)
    show Θ ; B ; (x, b, TRUE) #Γ Γ ⊢wf V-var x : b proof
      show Θ ; B ⊢wf (x, b, TRUE) #Γ Γ using wfG-cons2I wfX-wfY assms ⟨Θ ; B ⊢wf b⟩ by auto
      show Some (b, TRUE) = lookup ((x, b, TRUE) #Γ Γ) x using lookup.simps by auto
    qed
  qed
  show Θ ; B ; (x, b, TRUE) #Γ Γ ⊢wf ce : b
    using assms wf-weakening1(8)[OF assms(1), of (x, b, TRUE) #Γ Γ] * toSet.simps wfX-wfY
    by (metis Un-subset-iff equalityE)
  qed
qed

lemma wfC-e-eq2:
  fixes e1::ce and e2::ce
  assumes Θ ; B ; Γ ⊢wf e1 : b and Θ ; B ; Γ ⊢wf e2 : b and ⊢wf Θ and atom x # Γ
  shows Θ ; B ; (x, b, (CE-val (V-var x)) == e1) #Γ Γ ⊢wf (CE-val (V-var x)) == e2
  proof(rule wfC-eqI)
    have *: Θ ; B ⊢wf (x, b, CE-val (V-var x)) == e1 #Γ Γ proof(rule wfG-cons1I)
      show (CE-val (V-var x)) == e1 ∉ {TRUE, FALSE} by auto
      show Θ ; B ⊢wf Γ using assms wfX-wfY by metis
      show *:atom x # Γ using assms by auto
      show Θ ; B ; (x, b, TRUE) #Γ Γ ⊢wf CE-val (V-var x) == e1 using wfC-e-eq assms * by auto
      show Θ ; B ⊢wf b using assms wfX-wfB by auto
    qed
    show Θ ; B ; (x, b, CE-val (V-var x)) == e1 #Γ Γ ⊢wf CE-val (V-var x) : b using assms *
    wfCE-valI wfV-varI by auto
    show Θ ; B ; (x, b, CE-val (V-var x)) == e1 #Γ Γ ⊢wf e2 : b proof(rule wf-weakening1(8))
      show Θ ; B ; Γ ⊢wf e2 : b using assms by auto
      show Θ ; B ⊢wf (x, b, CE-val (V-var x)) == e1 #Γ Γ using * by auto
      show toSet Γ ⊆ toSet ((x, b, CE-val (V-var x)) == e1 #Γ Γ) by auto
    qed
  qed

lemma wfT-wfT-if-rev:
  assumes wfV P B Γ v (base-for-lit l) and wfT P B Γ t and ⟨atom z1 # Γ⟩
  shows wfT P B Γ (⟦ z1 : b-of t | CE-val v == CE-val (V-lit l) IMP (c-of t z1) ⋈ ⟩)
  proof
    show ⟨ P ; B ⊢wf b-of t ⟩ using wfX-wfY assms by meson
    have wfg: P ; B ⊢wf (z1, b-of t, TRUE) #Γ Γ using assms wfV-wf wfG-cons2I wfX-wfY
    by (meson wfG-cons-TRUE)
    show ⟨ P ; B ; (z1, b-of t, TRUE) #Γ Γ ⊢wf [ v ]ce == [ [ l ]v ]ce IMP c-of t z1 ⟩ proof
      show *: ⟨ P ; B ; (z1, b-of t, TRUE) #Γ Γ ⊢wf [ v ]ce == [ [ l ]v ]ce ⟩
      proof(rule wfC-eqI[where b=base-for-lit l])
        show P ; B ; (z1, b-of t, TRUE) #Γ Γ ⊢wf [ v ]ce : base-for-lit l
          using assms wf-intros wf-weakening wfg by (meson wfV-weakening-cons)
        show P ; B ; (z1, b-of t, TRUE) #Γ Γ ⊢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
  qed

```

thus $\langle P; \mathcal{B}; (z1, b\text{-of } t, \text{TRUE}) \rangle \#_{\Gamma} \Gamma \vdash_{wf} c\text{-of } t \ z1 \rangle$ **using** *wfT-wfC* *assms* *wfG-elim*s * **by**
simp
qed
show $\langle atom \ z1 \ \# \ (P, \mathcal{B}, \Gamma) \rangle$ **using** *assms* *wfG-fresh-x* *wfX-wfY* **by** *metis*
qed

lemma *wfT-eq-imp*:

fixes *zz::x* **and** *ll::l* **and** $\tau'::\tau$
assumes *base-for-lit* *ll* = *B-bool* **and** $\Theta; \{\|\} ; GNil \vdash_{wf} \tau'$ **and**
 $\Theta; \{\|\} \vdash_{wf} (x, b\text{-of } \llbracket z' : B\text{-bool} \mid \text{TRUE} \rrbracket, c\text{-of } \llbracket z' : B\text{-bool} \mid \text{TRUE} \rrbracket x) \#_{\Gamma} GNil$ **and**
 $atom \ zz \ \# \ x$
shows $\Theta; \{\|\} ; (x, b\text{-of } \llbracket z' : B\text{-bool} \mid \text{TRUE} \rrbracket, c\text{-of } \llbracket z' : B\text{-bool} \mid \text{TRUE} \rrbracket x) \#_{\Gamma}$
 $GNil \vdash_{wf} \llbracket zz : b\text{-of } \tau' \mid [\llbracket x \rrbracket^v]^{ce} == [\llbracket ll \rrbracket^v]^{ce} \text{ IMP } c\text{-of } \tau' \ zz \rrbracket$
proof(*rule wfT-wfT-if-rev*)
show $\langle \Theta; \{\|\} ; (x, b\text{-of } \llbracket z' : B\text{-bool} \mid \text{TRUE} \rrbracket, c\text{-of } \llbracket z' : B\text{-bool} \mid \text{TRUE} \rrbracket x) \#_{\Gamma} GNil \vdash_{wf} [$
 $x \rrbracket^v : \text{base-for-lit } ll \rangle$
using *wfV-varI* *lookup.simps* *base-for-lit.simps* *assms* **by** *simp*
show $\langle \Theta; \{\|\} ; (x, b\text{-of } \llbracket z' : B\text{-bool} \mid \text{TRUE} \rrbracket, c\text{-of } \llbracket z' : B\text{-bool} \mid \text{TRUE} \rrbracket x) \#_{\Gamma} GNil \vdash_{wf}$
 $\tau' \rangle$
using *wf-weakening* *assms* *toSet.simps* **by** *auto*
show $\langle atom \ zz \ \# \ (x, b\text{-of } \llbracket z' : B\text{-bool} \mid \text{TRUE} \rrbracket, c\text{-of } \llbracket z' : B\text{-bool} \mid \text{TRUE} \rrbracket x) \#_{\Gamma} GNil \rangle$
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** $\Gamma::\Gamma$ **and** *v::v*
assumes $\Theta; \mathcal{B}; \Gamma \vdash_{wf} v : b$ **and** $atom \ x \ \# \ \Gamma$
shows $\Theta; \mathcal{B}; ((x, b, \text{TRUE}) \#_{\Gamma} \Gamma) \vdash_{wf} (CE\text{-val } (V\text{-var } x) == CE\text{-val } v)$
using *wfC-e-eq* *wfCE-valI* *assms* *wfX-wfY* **by** *auto*

lemma *wfT-e-eq*:

fixes *ce::ce*
assumes $\Theta; \mathcal{B}; \Gamma \vdash_{wf} ce : b$ **and** $atom \ z \ \# \ \Gamma$
shows $\Theta; \mathcal{B}; \Gamma \vdash_{wf} \llbracket z : b \mid CE\text{-val } (V\text{-var } z) == ce \rrbracket$
proof
show $\Theta; \mathcal{B} \vdash_{wf} b$ **using** *wfX-wfB* *assms* **by** *auto*
show $atom \ z \ \# \ (\Theta, \mathcal{B}, \Gamma)$ **using** *assms* *wfG-fresh-x* *wfX-wfY* **by** *metis*
show $\Theta; \mathcal{B}; (z, b, \text{TRUE}) \#_{\Gamma} \Gamma \vdash_{wf} CE\text{-val } (V\text{-var } z) == ce$
using *wfTI* *wfC-e-eq* *assms* *wfTI* **by** *auto*
qed

lemma *wfT-v-eq*:

assumes *wfB* $\Theta \ \mathcal{B} \ b$ **and** *wfV* $\Theta \ \mathcal{B} \ \Gamma \ v \ b$ **and** $atom \ z \ \# \ \Gamma$
shows *wfT* $\Theta \ \mathcal{B} \ \Gamma \ \llbracket z : b \mid C\text{-eq } (CE\text{-val } (V\text{-var } z)) (CE\text{-val } v) \rrbracket$
using *wfT-e-eq* *wfE-valI* *assms* *wfX-wfY*
by (*simp* *add: wfCE-valI*)

lemma *wfC-wfG*:

fixes $\Gamma::\Gamma$ **and** *c::c* **and** *b::b*
assumes $\Theta; \mathcal{B}; \Gamma \vdash_{wf} c$ **and** $\Theta; \mathcal{B} \vdash_{wf} b$ **and** $atom \ x \ \# \ \Gamma$

shows $\Theta ; B \vdash_{wf} (x, b, c) \#_{\Gamma} \Gamma$
proof –
 have $\Theta ; B \vdash_{wf} (x, b, TRUE) \#_{\Gamma} \Gamma$ **using** *wfG-cons2I* *assms* *wfX-wfY* **by** *fast*
 hence $\Theta ; B ; (x, b, TRUE) \#_{\Gamma} \Gamma \vdash_{wf} c$ **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 $\Gamma'::\Gamma$
 assumes $wfG\ P\ \mathcal{B}\ ((x', b', c') \#_{\Gamma} \Gamma' @ (x, b, c) \#_{\Gamma} \Gamma)$
 shows $x' \neq x$
proof –
 have $atom\ x' \# (\Gamma' @ (x, b, c) \#_{\Gamma} \Gamma)$
 using *assms* *wfG-elim2* **by** *blast*
 thus *?thesis*
 using *fresh-gamma-append* [*of* $atom\ x' \Gamma' (x, b, c) \#_{\Gamma} \Gamma$] *fresh-GCons* *fresh-prod3* [*of* $atom\ x' x\ b\ c$] **by** *auto*
qed

lemma *replace-in-g-inside*:
 fixes $\Gamma::\Gamma$
 assumes $wfG\ P\ \mathcal{B}\ (\Gamma' @ ((x, b0, c0') \#_{\Gamma} \Gamma))$
 shows $replace-in-g\ (\Gamma' @ ((x, b0, c0') \#_{\Gamma} \Gamma))\ x\ c0 = (\Gamma' @ ((x, b0, c0) \#_{\Gamma} \Gamma))$
using *assms* **proof** (*induct* Γ' *rule*: Γ -*induct*)
 case *GNil*
 then show *?case* **using** *replace-in-g.simps* **by** *auto*
next
 case ($GCons\ x'\ b'\ c'\ \Gamma''$)
 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-suppl-rig-eq*:
 fixes $\Gamma::\Gamma$
 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 $\dots = supp\ x \cup supp\ b0 \cup supp\ c0' \cup supp\ \Gamma \cup supp\ \mathcal{B}$ **using** *supp-GCons* **by** *auto*
 also have $\dots = supp\ x \cup supp\ b0 \cup supp\ c0 \cup supp\ \Gamma \cup supp\ \mathcal{B}$ **using** *GNil* *wfG-wfC* [*THEN* *wfC-suppl-cons*(2)] **by** *fastforce*
 also have $\dots = (supp\ ((x, b0, c0) \#_{\Gamma} \Gamma)) \cup supp\ \mathcal{B}$ **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 ($GCons\ xbc\ \Gamma1$)

moreover have $(x b c \#_{\Gamma} \Gamma 1) @ (x, b 0, c 0) \#_{\Gamma} \Gamma = (x b c \#_{\Gamma} (\Gamma 1 @ (x, b 0, c 0) \#_{\Gamma} \Gamma)) \#_{\Gamma} \Gamma$ **by**
simp
moreover have $(x b c \#_{\Gamma} \Gamma 1) @ (x, b 0, c 0') \#_{\Gamma} \Gamma = (x b c \#_{\Gamma} (\Gamma 1 @ (x, b 0, c 0') \#_{\Gamma} \Gamma)) \#_{\Gamma} \Gamma$ **by**
simp
ultimately have $(P; \mathcal{B} \vdash_{wf} \Gamma 1 @ ((x, b 0, c 0) \#_{\Gamma} \Gamma)) \wedge P; \mathcal{B} \vdash_{wf} \Gamma 1 @ ((x, b 0, c 0') \#_{\Gamma} \Gamma)$
using *wfG-cons2* **by** *metis*
thus ?case using *GCons supp-GCons* **by** *auto*
qed

lemma *fresh-replace-inside[ms-fresh]*:

fixes $y::x$ **and** $\Gamma::\Gamma$
assumes $wfG P \mathcal{B} (\Gamma'' @ (x, b, c) \#_{\Gamma} \Gamma)$ **and** $wfG P \mathcal{B} (\Gamma'' @ (x, b, c') \#_{\Gamma} \Gamma)$
shows $atom y \# (\Gamma'' @ (x, b, c) \#_{\Gamma} \Gamma) = atom y \# (\Gamma'' @ (x, b, c') \#_{\Gamma} \Gamma)$
unfolding *fresh-def* **using** *wfG-supp-rig-eq* *assms x-not-in-b-set* **by** *fast*

lemma *wf-replace-inside1*:

fixes $\Gamma::\Gamma$ **and** $\Phi::\Phi$ **and** $\Theta::\Theta$ **and** $\Gamma':\Gamma$ **and** $v::v$ **and** $e::e$ **and** $c::c$ **and** $c'::c$ **and** $c'::c$ **and** $\tau::\tau$
and $ts::(string*\tau)$ **list** **and** $\Delta::\Delta$ **and** $b'::b$ **and** $b::b$ **and** $s::s$
and $ftq::fun\text{-}typ\text{-}q$ **and** $ft::fun\text{-}typ$ **and** $ce::ce$ **and** $td::type\text{-}def$ **and** $cs::branch\text{-}s$ **and**
 $css::branch\text{-}list$

shows *wfV-replace-inside*: $\Theta; \mathcal{B}; G \vdash_{wf} v : b' \implies G = (\Gamma' @ (x, b, c') \#_{\Gamma} \Gamma) \implies \Theta; \mathcal{B}; ((x, b, TRUE) \#_{\Gamma} \Gamma) \vdash_{wf} c \implies \Theta; \mathcal{B}; (\Gamma' @ (x, b, c) \#_{\Gamma} \Gamma) \vdash_{wf} v : b'$ **and**
wfC-replace-inside: $\Theta; \mathcal{B}; G \vdash_{wf} c'' \implies G = (\Gamma' @ (x, b, c') \#_{\Gamma} \Gamma) \implies \Theta; \mathcal{B}; ((x, b, TRUE) \#_{\Gamma} \Gamma) \vdash_{wf} c \implies \Theta; \mathcal{B}; (\Gamma' @ (x, b, c) \#_{\Gamma} \Gamma) \vdash_{wf} c''$ **and**
wfG-replace-inside: $\Theta; \mathcal{B} \vdash_{wf} G \implies G = (\Gamma' @ (x, b, c') \#_{\Gamma} \Gamma) \implies \Theta; \mathcal{B}; ((x, b, TRUE) \#_{\Gamma} \Gamma) \vdash_{wf} c \implies \Theta; \mathcal{B} \vdash_{wf} (\Gamma' @ (x, b, c) \#_{\Gamma} \Gamma)$ **and**
wfT-replace-inside: $\Theta; \mathcal{B}; G \vdash_{wf} \tau \implies G = (\Gamma' @ (x, b, c') \#_{\Gamma} \Gamma) \implies \Theta; \mathcal{B}; ((x, b, TRUE) \#_{\Gamma} \Gamma) \vdash_{wf} c \implies \Theta; \mathcal{B}; (\Gamma' @ (x, b, c) \#_{\Gamma} \Gamma) \vdash_{wf} \tau$ **and**
 $\Theta; \mathcal{B}; \Gamma \vdash_{wf} ts \implies True$ **and**
 $\vdash_{wf} P \implies True$ **and**
 $\Theta; \mathcal{B} \vdash_{wf} b \implies True$ **and**
wfCE-replace-inside: $\Theta; \mathcal{B}; G \vdash_{wf} ce : b' \implies G = (\Gamma' @ (x, b, c') \#_{\Gamma} \Gamma) \implies \Theta; \mathcal{B}; ((x, b, TRUE) \#_{\Gamma} \Gamma) \vdash_{wf} c \implies \Theta; \mathcal{B}; (\Gamma' @ (x, b, c) \#_{\Gamma} \Gamma) \vdash_{wf} ce : b'$ **and**
 $\Theta \vdash_{wf} td \implies True$

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* $\Theta \mathcal{B} \Gamma 2 b 2 c 2 x 2$)

then show ?case **using** *wf-intros* **by** (*metis lookup-in-rig-eq lookup-in-rig-neq replace-in-g-inside*)

next

case (*wfV-conspI* $s bv dclist \Theta dc x 1 b' c 1 \mathcal{B} b 1 \Gamma 1 v$)

show ?case **proof**

show $\langle AF\text{-}typedef\text{-}poly s bv dclist \in set \Theta \rangle$ **using** *wfV-conspI* **by** *auto*

show $\langle (dc, \{ x 1 : b' \mid c 1 \}) \in set dclist \rangle$ **using** *wfV-conspI* **by** *auto*

show $\langle \Theta; \mathcal{B} \vdash_{wf} b 1 \rangle$ **using** *wfV-conspI* **by** *auto*

show *: $\langle \Theta; \mathcal{B}; \Gamma' @ (x, b, c) \#_{\Gamma} \Gamma \vdash_{wf} v : b'[bv::=b 1]_{bb} \rangle$ **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, B, Γ' @ (x, b, c) #Γ Γ, b1, v)⟩
      unfolding fresh-prodN using fresh-prodN wfV-conspI by metis
qed
next
case (wfTI z Θ B G b1 c1)
show ?case proof
  show ⟨Θ; B ⊢wf b1⟩ using wfTI by auto

  have Θ; B ⊢wf (x, b, c) #Γ Γ using wfG-consI wfTI wfG-cons wfX-wfY by metis
  moreover hence *:wfG Θ B (Γ' @ (x, b, c) #Γ Γ) using wfX-wfY
    by (metis append-g.simps(2) wfG-cons2 wfTI.hyps wfTI.prem(1) wfTI.prem(2))
  hence ⟨atom z # Γ' @ (x, b, c) #Γ Γ⟩
    using fresh-replace-inside[of Θ B Γ' x b c Γ c' z, OF *] wfTI wfX-wfY wfG-elim by metis
  thus ⟨atom z # (Θ, B, Γ' @ (x, b, c) #Γ Γ)⟩ using wfG-fresh-x[OF *] by auto

  have (z, b1, TRUE) #Γ G = ((z, b1, TRUE) #Γ Γ') @ (x, b, c') #Γ Γ
    using wfTI append-g.simps by metis
  thus ⟨Θ; B; (z, b1, TRUE) #Γ Γ' @ (x, b, c) #Γ Γ ⊢wf c1⟩
    using wfTI(9)[OF - wfTI(11)] by fastforce
qed
next
case (wfG-nilI Θ)
hence GNil = (x, b, c') #Γ Γ 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 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) #Γ G' = Γ' using wfG-cons1I wfG-cons1I(7) GCons-eq-append-conv
    by auto
    hence *: G = G' @ (x, b, c') #Γ Γ using wfG-cons1I by auto
    hence Θ; B ⊢wf G' @ (x, b, c) #Γ Γ using wfG-cons1I by auto
    have Θ; B ⊢wf (x1, b1, c1) #Γ G' @ (x, b, c) #Γ Γ proof(rule Wellformed.wfG-cons1I)
      show c1 ∉ {TRUE, FALSE} using wfG-cons1I by auto
      show Θ; B ⊢wf G' @ (x, b, c) #Γ Γ using wfG-cons1I(3)[of G', OF **] wfG-cons1I by auto
      show atom x1 # G' @ (x, b, c) #Γ Γ using wfG-cons1I * ** fresh-replace-inside by metis
      show Θ; B; (x1, b1, TRUE) #Γ G' @ (x, b, c) #Γ Γ ⊢wf c1 using wfG-cons1I(6)[of (x1, b1,
TRUE) #Γ G'] wfG-cons1I ** by auto
      show Θ; B ⊢wf b1 using wfG-cons1I by auto
    qed
    thus ?thesis using * by auto
  qed
next
case (wfG-cons2I c1 Θ 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 $G'::\Gamma$ **where** $*(x1, b1, c1) \#_{\Gamma} G' = \Gamma'$ **using** *wfG-cons2I GCons-eq-append-conv*
by *auto*
hence $** : G = G' @ (x, b, c') \#_{\Gamma} \Gamma$ **using** *wfG-cons2I* **by** *auto*
moreover have $\Theta; \mathcal{B} \vdash_{wf} G' @ (x, b, c) \#_{\Gamma} \Gamma$ **using** *wfG-cons2I * ** by auto*
moreover hence $atom\ x1 \# G' @ (x, b, c) \#_{\Gamma} \Gamma$ **using** *wfG-cons2I * ** fresh-replace-inside* **by**
metis
ultimately show *?thesis* **using** *Wellformed.wfG-cons2I[OF wfG-cons2I(1), of $\Theta \mathcal{B} G' @ (x, b, c)$*
 $\#_{\Gamma} \Gamma\ x1\ b1]$ *wfG-cons2I * ** by auto*
qed
qed(*metis wf-intros*)+

lemma *wf-replace-inside2*:

fixes $\Gamma::\Gamma$ **and** $\Phi::\Phi$ **and** $\Theta::\Theta$ **and** $\Gamma'::\Gamma$ **and** $v::v$ **and** $e::e$ **and** $c::c$ **and** $c'::c$ **and** $c'::c$ **and** $\tau::\tau$
and $ts::(string*\tau)$ **list** **and** $\Delta::\Delta$ **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

$\Theta; \Phi; \mathcal{B}; G; D \vdash_{wf} e : b' \implies G = (\Gamma' @ (x, b, c') \#_{\Gamma} \Gamma) \implies \Theta; \mathcal{B}; ((x, b, TRUE) \#_{\Gamma} \Gamma)$
 $\vdash_{wf} c \implies \Theta; \Phi; \mathcal{B}; (\Gamma' @ (x, b, c) \#_{\Gamma} \Gamma); D \vdash_{wf} e : b'$ **and**
 $\Theta; \Phi; \mathcal{B}; \Gamma; \Delta \vdash_{wf} s : b \implies True$ **and**
 $\Theta; \Phi; \mathcal{B}; \Gamma; \Delta; tid; dc; t \vdash_{wf} cs : b \implies True$ **and**
 $\Theta; \Phi; \mathcal{B}; \Gamma; \Delta; tid; dclist \vdash_{wf} css : b \implies True$ **and**
 $\Theta \vdash_{wf} \Phi \implies True$ **and**
 $\Theta; \mathcal{B}; G \vdash_{wf} \Delta \implies G = (\Gamma' @ (x, b, c') \#_{\Gamma} \Gamma) \implies \Theta; \mathcal{B}; ((x, b, TRUE) \#_{\Gamma} \Gamma) \vdash_{wf} c \implies \Theta;$
 $\mathcal{B}; (\Gamma' @ (x, b, c) \#_{\Gamma} \Gamma) \vdash_{wf} \Delta$ **and**
 $\Theta; \Phi \vdash_{wf} ftq \implies True$ **and**
 $\Theta; \Phi; \mathcal{B} \vdash_{wf} ft \implies True$

proof(*nominal-induct*

b' **and** b **and** b **and** b **and** Φ **and** Δ **and** ftq **and** ft
avoiding: $\Gamma' c'$
rule:wfE-wfS-wfCS-wfCSS-wfPhi-wfD-wfFTQ-wfFT.strong-induct)
case (*wfE-valI* $\Theta \Phi \mathcal{B} \Gamma \Delta v b$)
then show *?case* **using** *wf-replace-inside1 Wellformed.wfE-valI* **by** *auto*
next
case (*wfE-plusI* $\Theta \Phi \mathcal{B} \Gamma \Delta v1 v2$)
then show *?case* **using** *wf-replace-inside1 Wellformed.wfE-plusI* **by** *auto*
next
case (*wfE-leqI* $\Theta \Phi \mathcal{B} \Gamma \Delta v1 v2$)
then show *?case* **using** *wf-replace-inside1 Wellformed.wfE-leqI* **by** *auto*
next
case (*wfE-fstI* $\Theta \Phi \mathcal{B} \Gamma \Delta v1 b1 b2$)
then show *?case* **using** *wf-replace-inside1 Wellformed.wfE-fstI* **by** *metis*
next
case (*wfE-sndI* $\Theta \Phi \mathcal{B} \Gamma \Delta v1 b1 b2$)
then show *?case* **using** *wf-replace-inside1 Wellformed.wfE-sndI* **by** *metis*
next
case (*wfE-concatI* $\Theta \Phi \mathcal{B} \Gamma \Delta v1 v2$)
then show *?case* **using** *wf-replace-inside1 Wellformed.wfE-concatI* **by** *auto*
next
case (*wfE-splitI* $\Theta \Phi \mathcal{B} \Gamma \Delta v1 v2$)
then show *?case* **using** *wf-replace-inside1 Wellformed.wfE-splitI* **by** *auto*


```

next
  case (wfE-lenI  $\Theta \Phi \mathcal{B} \Gamma \Delta v1$ )
  then show ?case using wf-replace-inside1 Wellformed.wfE-lenI by metis
next
  case (wfE-appI  $\Theta \Phi \mathcal{B} \Gamma \Delta f x b c \tau s v$ )
  then show ?case using wf-replace-inside1 Wellformed.wfE-appI by metis
next
  case (wfE-appPI  $\Theta \Phi \mathcal{B} \Gamma'' \Delta b' bv v \tau f x1 b1 c1 s$ )
  show ?case proof
    show  $\langle \Theta \vdash_{wf} \Phi \rangle$  using wfE-appPI by auto
    show  $\langle \Theta; \mathcal{B}; \Gamma' @ (x, b, c) \#_{\Gamma} \Gamma \vdash_{wf} \Delta \rangle$  using wfE-appPI by auto
    show  $\langle \Theta; \mathcal{B} \vdash_{wf} b' \rangle$  using wfE-appPI by auto
    show  $\ast: \langle \Theta; \mathcal{B}; \Gamma' @ (x, b, c) \#_{\Gamma} \Gamma \vdash_{wf} v : b1[bv::=b]_b \rangle$  using wfE-appPI wf-replace-inside1 by
auto
    moreover have  $\Theta; \mathcal{B} \vdash_{wf} \Gamma' @ (x, b, c') \#_{\Gamma} \Gamma$  using wfV-wf wfE-appPI by metis
    ultimately have  $atom bv \# \Gamma' @ (x, b, c) \#_{\Gamma} \Gamma$ 
      unfolding fresh-def using wfV-wf wfG-supp-rig-eq wfE-appPI Un-iff fresh-def by metis

    thus  $\langle atom bv \# (\Phi, \Theta, \mathcal{B}, \Gamma' @ (x, b, c) \#_{\Gamma} \Gamma, \Delta, b', v, (b-of \tau)[bv::=b]_b) \rangle$ 
      using wfE-appPI fresh-prodN by metis
    show  $\langle Some (AF-fundef f (AF-fun-typ-some bv (AF-fun-typ x1 b1 c1 \tau s))) = lookup-fun \Phi f \rangle$ 
using wfE-appPI by auto
  qed
next
  case (wfE-mvarI  $\Theta \Phi \mathcal{B} \Gamma \Delta u \tau$ )
  then show ?case using wf-replace-inside1 Wellformed.wfE-mvarI by metis
next
  case (wfD-emptyI  $\Theta \mathcal{B} \Gamma$ )
  then show ?case using wf-replace-inside1 Wellformed.wfD-emptyI by metis
next
  case (wfD-cons  $\Theta \mathcal{B} \Gamma \Delta \tau u$ )
  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  $\Theta \Phi ft$ )
  then show ?case using wf-replace-inside1 Wellformed.wfD-emptyI by metis
next
  case (wfFTSome  $\Theta \Phi bv ft$ )
  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  $P; \mathcal{B}; (x, b, TRUE) \#_{\Gamma} \Gamma \vdash_{wf} c2$  using wfG-elim2 assms by auto
    show  $\langle (x, b, TRUE) \#_{\Gamma} \Gamma = GNil @ (x, b, TRUE) \#_{\Gamma} \Gamma \rangle$  using append-g.simps by auto
    show  $\langle P; \mathcal{B}; (x, b, TRUE) \#_{\Gamma} \Gamma \vdash_{wf} c1 \rangle$  using wfG-elim2 assms by auto
  qed

```

qed
 thus *?thesis* using *append-g.simps* by auto
 qed

lemma *wfC-refl*:
 assumes *wfG* $\Theta \mathcal{B} ((x, b', c') \#_{\Gamma} \Gamma)$
 shows *wfC* $\Theta \mathcal{B} ((x, b', c') \#_{\Gamma} \Gamma) c'$
 using *wfG-wfC* *assms* *wfC-replace-cons* by auto

lemma *wfG-wfC-inside*:
 assumes $(x, b, c) \in \text{toSet } G$ and *wfG* $\Theta B G$
 shows *wfC* $\Theta B G c$
 using *assms* **proof**(*induct* *G* rule: Γ -*induct*)
 case *GNil*
 then show *?case* by auto
 next
 case (*GCons* $x' b' c' \Gamma'$)
 then consider (*hd*) $(x, b, c) = (x', b', c') \mid$ (*tail*) $(x, b, c) \in \text{toSet } \Gamma'$ using *toSet.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* *toSet.simps*(2) *wfG-cons2*)
 qed
 qed

lemma *wfT-wf-cons3*:
 assumes $\Theta; \mathcal{B}; \Gamma \vdash_{wf} \{ z : b \mid c \}$ and *atom* $y \nmid (c, \Gamma)$
 shows $\Theta; \mathcal{B} \vdash_{wf} (y, b, c[z ::= V\text{-var } y]_{cv}) \#_{\Gamma} \Gamma$
proof –
 have $\{ z : b \mid c \} = \{ y : b \mid (y \leftrightarrow z) \cdot c \}$ using *type-eq-flip* *assms* by auto
 moreover hence $(y \leftrightarrow z) \cdot c = c[z ::= V\text{-var } y]_{cv}$ using *assms* *subst-v-c-def* by auto
 ultimately have $\{ z : b \mid c \} = \{ y : b \mid c[z ::= V\text{-var } y]_{cv} \}$ by *metis*
 thus *?thesis* using *assms* *wfT-wf-cons*[*of* $\Theta \mathcal{B} \Gamma y b$] *fresh-Pair* by *metis*
 qed

lemma *wfT-wfC-cons*:
 assumes *wfT* $P \mathcal{B} \Gamma \{ z1 : b \mid c1 \}$ and *wfT* $P \mathcal{B} \Gamma \{ z2 : b \mid c2 \}$ and *atom* $x \nmid (c1, c2, \Gamma)$
 shows *wfC* $P \mathcal{B} ((x, b, c1[z1 ::= V\text{-var } x]_v) \#_{\Gamma} \Gamma) (c2[z2 ::= V\text{-var } x]_v)$ (**is** *wfC* $P \mathcal{B} ?G ?c$)
proof –
 have *eq*: $\{ z2 : b \mid c2 \} = \{ x : b \mid c2[z2 ::= V\text{-var } x]_{cv} \}$ using *type-eq-subst* *assms* *fresh-prod3* by *simp*
 have *eq2*: $\{ z1 : b \mid c1 \} = \{ x : b \mid c1[z1 ::= V\text{-var } x]_{cv} \}$ using *type-eq-subst* *assms* *fresh-prod3* by *simp*
 moreover have *wfT* $P \mathcal{B} \Gamma \{ x : b \mid c1[z1 ::= V\text{-var } x]_{cv} \}$ using *assms* *eq2* by auto
 moreover hence *wfG* $P \mathcal{B} ((x, b, c1[z1 ::= V\text{-var } x]_{cv}) \#_{\Gamma} \Gamma)$ using *wfT-wf-cons* *fresh-prod3* *assms* by auto
 moreover have *wfT* $P \mathcal{B} \Gamma \{ x : b \mid c2[z2 ::= V\text{-var } x]_{cv} \}$ using *assms* *eq* by auto
 moreover hence *wfC* $P \mathcal{B} ((x, b, TRUE) \#_{\Gamma} \Gamma) (c2[z2 ::= V\text{-var } x]_{cv})$ 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 $\Theta; \mathcal{B}; \Gamma \vdash_{wf} \{ z : b \mid c \}$ and *atom x # Γ*
 shows $\Theta; \mathcal{B}; (x, b, TRUE) \#_{\Gamma} \Gamma \vdash_{wf} c[z::=[x]^v]_v$
 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 $\{ x : b \mid c[z::=[x]^v]_v \} = \{ z : b \mid c \}$
 using $\tau.eq-iff Abs1-eq-iff(\exists)[of\ x\ c[z::=[x]^v]_v\ z\ c]$
 by (*metis flip-subst-v type-eq-flip*)
 hence $\Theta; \mathcal{B}; \Gamma \vdash_{wf} \{ x : b \mid c[z::=[x]^v]_v \}$ using *assms* by *metis*
 thus *?thesis* using *wfT-wfC assms* by *auto*
 qed

lemma *wfT-wfG*:
 fixes *x::x* and $\Gamma::\Gamma$ and *z::x* and *c::c* and *b::b*
 assumes $\Theta; \mathcal{B}; \Gamma \vdash_{wf} \{ z : b \mid c \}$ and *atom x # Γ*
 shows $\Theta; \mathcal{B} \vdash_{wf} (x, b, c[z::=[x]^v]_v) \#_{\Gamma} \Gamma$
 proof –
 have $\Theta; \mathcal{B}; (x, b, TRUE) \#_{\Gamma} \Gamma \vdash_{wf} c[z::=[x]^v]_v$ 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 $\Gamma::\Gamma$
 assumes *wfG P B* ($\Gamma' @ (x, b, c') \#_{\Gamma} \Gamma$) and *wfG P B* ($(x, b, c) \#_{\Gamma} \Gamma$)
 shows *wfG P B* ($\Gamma' @ (x, b, c) \#_{\Gamma} \Gamma$)
 proof –
 have *wfC P 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 $\Gamma::\Gamma$
 assumes *wfG P B* ($\Gamma' @ (x, b, c') \#_{\Gamma} \Gamma$) and *wfG P B* ($\Gamma' @ ((x, b, c) \#_{\Gamma} \Gamma)$)
 shows *wfG P B* ($\Gamma' @ (x, b, c) \#_{\Gamma} \Gamma$)
 proof –
 have *wfG P 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 $wfT \Theta \mathcal{B} (\Gamma' @ (x, b, c') \#_{\Gamma} \Gamma) t$ **and** $wfG \Theta \mathcal{B} (\Gamma' @ ((x, b, c) \#_{\Gamma} \Gamma))$
shows $wfT \Theta \mathcal{B} (\Gamma' @ (x, b, c) \#_{\Gamma} \Gamma) t$

proof –

have $wfG \Theta \mathcal{B} (((x, b, c) \#_{\Gamma} \Gamma))$ **using** *wfG-suffix assms* **by** *auto*
hence $wfC \Theta \mathcal{B} ((x, b, TRUE) \#_{\Gamma} \Gamma) c$ **using** *wfG-wfC* **by** *auto*
thus *?thesis* **using** *wf-replace-inside assms* **by** *metis*

qed

lemma *wfD-unique*:

assumes $wfD P \mathcal{B} \Gamma \Delta$ **and** $(u, \tau') \in \text{setD } \Delta$ **and** $(u, \tau) \in \text{setD } \Delta$
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 $*$: $wfD P \mathcal{B} \Gamma ((u', t') \#_{\Delta} D)$ **using** *Cons* **by** *auto*

show *?case* **proof**(*cases* $u = u'$)

case *True*

then have $u \notin \text{fst } \text{setD } D$ **using** *wfD-elim* $*$ **by** *blast*

then show *?thesis* **using** *DCons* **by** *force*

next

case *False*

then show *?thesis* **using** *DCons* *wfD-elim* $*$ **by** (*metis* *fst-conv* *setD-ConsD*)

qed

qed

lemma *replace-in-g-forget*:

fixes $x :: x$

assumes $wfG P B G$

shows $\text{atom } x \notin \text{atom-dom } G \implies (G[x \mapsto c]) = G$ **and**

$\text{atom } x \# G \implies (G[x \mapsto c]) = G$

proof –

show $\text{atom } x \notin \text{atom-dom } G \implies G[x \mapsto c] = G$ **by** (*induct* G *rule*: Γ -*induct*, *auto*)

thus $\text{atom } x \# G \implies (G[x \mapsto c]) = G$ **using** *wfG-x-fresh assms* **by** *simp*

qed

lemma *replace-in-g-fresh-single*:

fixes $G :: \Gamma$ **and** $x :: x$

assumes $\langle \Theta; \mathcal{B} \vdash_{wf} G[x' \mapsto c''] \rangle$ **and** $\text{atom } x \# G$ **and** $\langle \Theta; \mathcal{B} \vdash_{wf} G \rangle$

shows $\text{atom } x \# G[x' \mapsto c']$

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 $\text{atom } x \# \Gamma \wedge \text{wfG } \Theta \mathcal{B} \Gamma \wedge \Theta; \mathcal{B} \vdash_{\text{wf}} b$ **using** *wfG-cons* **by** *auto*
 hence $\Theta; \mathcal{B}; (x, b, \text{TRUE}) \#_{\Gamma} \Gamma \vdash_{\text{wf}} \text{TRUE}$ **using** *wfC-trueI wfG-cons2I* **by** *simp*
 hence $\Theta; \mathcal{B}; (x, b, \text{TRUE}) \#_{\Gamma} \Gamma \vdash_{\text{wf}} c'$
using *wf-replace-inside1(2)[of $\Theta \mathcal{B} (x, b, c) \#_{\Gamma} \Gamma c' \text{GNil } x b c \Gamma \text{TRUE}]$ assms* **by** *auto*
 hence $\text{wfG } \Theta \mathcal{B} ((x, b, c') \#_{\Gamma} \Gamma)$ **using** *wf-replace-inside1(3)[OF *, of $\text{GNil } x b c \Gamma c']$* **by** *auto*
 moreover have $\text{wfC } \Theta \mathcal{B} ((x, b, \text{TRUE}) \#_{\Gamma} \Gamma) c$ **proof**(*cases* $c \in \{ \text{TRUE}, \text{FALSE} \}$)
 case *True*
 have $\Theta; \mathcal{B} \vdash_{\text{wf}} \Gamma \wedge \text{atom } x \# \Gamma \wedge \Theta; \mathcal{B} \vdash_{\text{wf}} b$ **using** *wfG-elim(2)[OF *]* **by** *auto*
 hence $\Theta; \mathcal{B} \vdash_{\text{wf}} (x, b, \text{TRUE}) \#_{\Gamma} \Gamma$ **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-elim(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 $\text{wfG } P \mathcal{B} (\Gamma_1 @ ((x, b, c) \#_{\Gamma} \Gamma_2))$
 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 $*; P; \mathcal{B} \vdash_{\text{wf}} (x', b', c') \#_{\Gamma} (G @ (x, b, c) \#_{\Gamma} \Gamma_2)$ **by** *auto*
 hence $x \neq x'$
using *GCons append-Cons wfG-cons-fresh2[OF *]* **by** *auto*
 hence $((\text{GCons } (x', b', c') G) @ (\text{GCons } (x, b, c) \Gamma_2))[x ::= v]_{\Gamma_v} =$
 $(\text{GCons } (x', b', c') (G @ (\text{GCons } (x, b, c) \Gamma_2)))[x ::= v]_{\Gamma_v}$ **by** *auto*
 also have $\dots = \text{GCons } (x', b', c'[x ::= v]_{c_v}) ((G @ (\text{GCons } (x, b, c) \Gamma_2))[x ::= v]_{\Gamma_v})$
using *subst-gv.simps $\langle x \neq x' \rangle$* **by** *simp*
 also have $\dots = (x', b', c'[x ::= v]_{c_v}) \#_{\Gamma} (G[x ::= v]_{\Gamma_v} @ \Gamma_2)$ **using** *GCons * wfG-elim* **by** *metis*
 also have $\dots = ((x', b', c') \#_{\Gamma} G)[x ::= v]_{\Gamma_v} @ \Gamma_2$ **using** *subst-gv.simps $\langle x \neq x' \rangle$* **by** *simp*
 finally show *?case* **by** *blast*
 qed

lemma *subst-c-TRUE-FALSE*:

fixes $c :: c$
 assumes $c \notin \{ \text{TRUE}, \text{FALSE} \}$
 shows $c[x ::= v]_{c_v} \notin \{ \text{TRUE}, \text{FALSE} \}$
using *assms* **by**(*nominal-induct c rule:c.strong-induct, auto simp add: subst-cv.simps*)

lemma *lookup-subst*:

assumes *Some* $(b, c) = \text{lookup } \Gamma x$ **and** $x \neq x'$
 shows $\exists c'. \text{Some } (b, c') = \text{lookup } \Gamma[x' ::= v]_{\Gamma_v} x$
using *assms* **proof**(*induct* Γ *rule: Γ -induct*)
 case *GNil*
 then show *?case* **by** *auto*
 next
 case (*GCons* $x1 b1 c1 \Gamma1$)

```

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) #Γ Γ1)[x'::=v]Γv = ((x1, b1, c1[x'::=v]cv) #Γ Γ1[x'::=v]Γv) using
subst-gv.simps by auto
then show ?thesis proof(cases x1=x)
  case True
  then show ?thesis using lookup.simps *
  using GCons.premis(1) by auto
next
case False
then show ?thesis using lookup.simps *
using GCons.premis(1) by (simp add: GCons.hyps assms(2))
qed
qed
qed

```

lemma lookup-subst2:

assumes Some (b, c) = lookup (Γ'@((x', b₁, c0[z0::=[x]_v)]#_ΓΓ)) x and x ≠ x' and
 Θ; B ⊢_{wf} (Γ'@((x', b₁, c0[z0::=[x]_v)]#_ΓΓ))
 shows ∃ c'. Some (b, c') = lookup (Γ[x'::=v]_{Γv}@Γ) x
 using assms lookup-subst subst-g-inside by metis

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; B; Γ ⊢_{wf} v : b ⇒ Γ=Γ₁@((x, b', c') #_ΓΓ₂) ⇒ Θ; B; Γ₂ ⊢_{wf} v' : b' ⇒
 Θ ; B ; Γ[x::=v]_{Γv} ⊢_{wf} v[x::=v]_v : b and
 wfC-subst: Θ; B; Γ ⊢_{wf} c ⇒ Γ=Γ₁@((x, b', c') #_ΓΓ₂) ⇒ Θ; B; Γ₂ ⊢_{wf} v' : b' ⇒
 Θ; B; Γ[x::=v]_{Γv} ⊢_{wf} c[x::=v]_{cv} and
 wfG-subst: Θ; B ⊢_{wf} Γ ⇒ Γ=Γ₁@((x, b', c') #_ΓΓ₂) ⇒ Θ; B ; Γ₂ ⊢_{wf} v' : b' ⇒
 Θ; B ⊢_{wf} Γ[x::=v]_{Γv} and
 wfT-subst: Θ; B; Γ ⊢_{wf} τ ⇒ Γ=Γ₁@((x, b', c') #_ΓΓ₂) ⇒ Θ; B ; Γ₂ ⊢_{wf} v' : b' ⇒
 Θ; B; Γ[x::=v]_{Γv} ⊢_{wf} τ[x::=v]_{τv} and
 Θ; B; Γ ⊢_{wf} ts ⇒ True and
 ⊢_{wf} Θ ⇒ True and
 Θ; B ⊢_{wf} b ⇒ True and
 wfCE-subst: Θ; B; Γ ⊢_{wf} ce : b ⇒ Γ=Γ₁@((x, b', c') #_ΓΓ₂) ⇒ Θ; B ; Γ₂ ⊢_{wf} v' : b' ⇒
 Θ ; B ; Γ[x::=v]_{Γv} ⊢_{wf} ce[x::=v]_{cev} : b and
 Θ ⊢_{wf} td ⇒ 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: Γ₁ and Γ₁ and Γ₁ and Γ₁ and Γ₁ and Γ₁ and Γ₁ and Γ₁ and Γ₁ and Γ₁ and Γ₁
 and Γ₁ and Γ₁ and Γ₁ and Γ₁ and Γ₁

rule:wfV-wfC-wfG-wfT-wfTs-wfTh-wfB-wfCE-wfTD.strong-induct)

case (wfV-varI Θ 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  $\Theta; \mathcal{B}; \Gamma[x::=v]_{\Gamma_v} \vdash_{wf} v' : b'$  using wfV-varI subst-g-inside-simple wf-weakening

  append-g-toSetU 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  $\Theta; \mathcal{B} \vdash_{wf} \Gamma[x::=v]_{\Gamma_v}$  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  $\Theta \mathcal{B} \Gamma[x::=v]_{\Gamma_v}$  b1 c1' x1] by metis
qed
next
case (wfV-litI  $\Theta \Gamma l$ )

  then show ?case using subst-vv.simps wf-intros by auto
next
case (wfV-pairI  $\Theta \Gamma v1 b1 v2 b2$ )
  then show ?case using subst-vv.simps wf-intros by auto
next
case (wfV-consI s dclist  $\Theta dc x b c \Gamma v$ )
  then show ?case using subst-vv.simps wf-intros by auto
next
case (wfV-conspl s bv dclist  $\Theta dc x' b' c \mathcal{B} b \Gamma va$ )
  show ?case unfolding subst-vv.simps proof
    show  $\langle AF\text{-typedef-poly } s \text{ bv dclist} \in \text{set } \Theta \rangle$  and  $\langle (dc, \{x' : b' \mid c\}) \in \text{set dclist} \rangle$  using wfV-conspl
  by auto
  show  $\langle \Theta; \mathcal{B} \vdash_{wf} b \rangle$  using wfV-conspl by auto
  have atom bv  $\# \Gamma[x::=v]_{\Gamma_v}$  using fresh-subst-gv-if wfV-conspl by metis
  moreover have atom bv  $\# va[x::=v]_{vv}$  using wfV-conspl fresh-subst-if by simp
  ultimately show  $\langle atom \text{ bv } \# (\Theta, \mathcal{B}, \Gamma[x::=v]_{\Gamma_v}, b, va[x::=v]_{vv}) \rangle$  unfolding fresh-prodN using
wfV-conspl by auto
  show  $\langle \Theta; \mathcal{B}; \Gamma[x::=v]_{\Gamma_v} \vdash_{wf} va[x::=v]_{vv} : b'[bv::=b]_{bb} \rangle$  using wfV-conspl by auto
  qed
next
case (wfTI z  $\Theta \mathcal{B} \Gamma b c$ )
  have  $\Theta; \mathcal{B}; \Gamma[x::=v]_{\Gamma_v} \vdash_{wf} \{z : b \mid c[x::=v]_{cv}\}$  proof
    have  $\langle \Theta; \mathcal{B}; ((z, b, TRUE) \#_{\Gamma} \Gamma)[x::=v]_{\Gamma_v} \vdash_{wf} c[x::=v]_{cv} \rangle$ 
    proof(rule wfTI(9))
      show  $\langle (z, b, TRUE) \#_{\Gamma} \Gamma = ((z, b, TRUE) \#_{\Gamma} \Gamma_1) @ (x, b', c') \#_{\Gamma} \Gamma_2 \rangle$  using wfTI
    append-g.simps by simp
    show  $\langle \Theta; \mathcal{B}; \Gamma_2 \vdash_{wf} v' : b' \rangle$  using wfTI by auto
  qed
  thus  $\langle \Theta; \mathcal{B}; (z, b, TRUE) \#_{\Gamma} \Gamma[x::=v]_{\Gamma_v} \vdash_{wf} c[x::=v]_{cv} \rangle$ 
  using subst-gv.simps subst-cv.simps wfTI fresh-x-neq by auto

  have atom z  $\# \Gamma[x::=v]_{\Gamma_v}$  using fresh-subst-gv-if wfTI by metis

```

```

    moreover have  $\Theta; \mathcal{B} \vdash_{wf} \Gamma[x::=v]_{\Gamma v}$  using wfTI wfX-wfY wfG-elim subst-gv.simps * by metis
    ultimately show  $\langle atom\ z\ \sharp\ (\Theta, \mathcal{B}, \Gamma[x::=v]_{\Gamma v}) \rangle$  using wfG-fresh-x by metis
    show  $\langle \Theta; \mathcal{B} \vdash_{wf} b \rangle$  using wfTI by auto

qed
thus ?case using subst-tv.simps wfTI by auto
next
case (wfC-trueI  $\Theta\ \Gamma$ )
then show ?case using subst-cv.simps wf-intros by auto
next
case (wfC-falseI  $\Theta\ \Gamma$ )
then show ?case using subst-cv.simps wf-intros by auto
next
case (wfC-eqI  $\Theta\ \mathcal{B}\ \Gamma\ e1\ b\ e2$ )
show ?case proof(subst subst-cv.simps, rule)
  show  $\Theta; \mathcal{B}; \Gamma[x::=v]_{\Gamma v} \vdash_{wf} e1[x::=v]_{cev} : b$  using wfC-eqI subst-dv.simps by auto
  show  $\Theta; \mathcal{B}; \Gamma[x::=v]_{\Gamma v} \vdash_{wf} e2[x::=v]_{cev} : b$  using wfC-eqI by auto
qed
next
case (wfC-conjI  $\Theta\ \Gamma\ c1\ c2$ )
then show ?case using subst-cv.simps wf-intros by auto
next
case (wfC-disjI  $\Theta\ \Gamma\ c1\ c2$ )
then show ?case using subst-cv.simps wf-intros by auto
next
case (wfC-notI  $\Theta\ \Gamma\ c1$ )
then show ?case using subst-cv.simps wf-intros by auto
next
case (wfC-impI  $\Theta\ \Gamma\ c1\ c2$ )
then show ?case using subst-cv.simps wf-intros by auto
next
case (wfG-nilI  $\Theta$ )
then show ?case using subst-cv.simps wf-intros by auto
next
case (wfG-cons1I  $c\ \Theta\ \mathcal{B}\ \Gamma\ y\ b$ )

show ?case proof(cases x=y)
  case True
  hence  $((y, b, c) \#_{\Gamma} \Gamma)[x::=v]_{\Gamma v} = \Gamma$  using subst-gv.simps by auto
  moreover have  $\Theta; \mathcal{B} \vdash_{wf} \Gamma$  using wfG-cons1I by auto
  ultimately show ?thesis by auto
next
  case False
  have  $\Gamma_1 \neq GNil$  using wfG-cons1I False by auto
  then obtain G where  $\Gamma_1 = (y, b, c) \#_{\Gamma} G$  using GCons-eq-append-conv wfG-cons1I by auto
  hence  $*\Gamma = G @ (x, b', c') \#_{\Gamma} \Gamma_2$  using wfG-cons1I by auto
  hence  $((y, b, c) \#_{\Gamma} \Gamma)[x::=v]_{\Gamma v} = (y, b, c[x::=v]_{cv}) \#_{\Gamma} \Gamma[x::=v]_{\Gamma v}$  using subst-gv.simps False
  by auto
  moreover have  $\Theta; \mathcal{B} \vdash_{wf} (y, b, c[x::=v]_{cv}) \#_{\Gamma} \Gamma[x::=v]_{\Gamma v}$  proof(rule Wellformed.wfG-cons1I)
    show  $\langle c[x::=v]_{cv} \notin \{TRUE, FALSE\} \rangle$  using wfG-cons1I subst-c-TRUE-FALSE by auto
    show  $\langle \Theta; \mathcal{B} \vdash_{wf} \Gamma[x::=v]_{\Gamma v} \rangle$  using wfG-cons1I * by auto
    have  $\Gamma = (G @ ((x, b', c') \#_{\Gamma} GNil)) @ \Gamma_2$  using * append-g-assoc by auto

```



```

    hence atom y #  $\Gamma_2$  using fresh-suffix  $\langle \text{atom } y \# \Gamma \rangle$  by auto
    hence atom y #  $v'$  using wfG-cons1I wfV-x-fresh by metis
    thus  $\langle \text{atom } y \# \Gamma[x::=v]_{\Gamma v} \rangle$  using fresh-subst-gv wfG-cons1I by auto
    have  $((y, b, \text{TRUE}) \#_{\Gamma} \Gamma[x::=v]_{\Gamma v} = (y, b, \text{TRUE}) \#_{\Gamma} \Gamma[x::=v]_{\Gamma v})$  using subst-gv.simps
subst-cv.simps False by auto
    thus  $\langle \Theta; \mathcal{B}; (y, b, \text{TRUE}) \#_{\Gamma} \Gamma[x::=v]_{\Gamma v} \vdash_{wf} c[x::=v]_{cv} \rangle$  using wfG-cons1I(6)[of (y,b,TRUE)
 $\#_{\Gamma} G] * \text{subst-gv.simps}$ 
    wfG-cons1I by fastforce
    show  $\Theta; \mathcal{B} \vdash_{wf} b$  using wfG-cons1I by auto
qed
ultimately show ?thesis by auto
qed
next
case (wfG-cons2I c  $\Theta \mathcal{B} \Gamma y b$ )

show ?case proof(cases x=y)
  case True
    hence  $((y, b, c) \#_{\Gamma} \Gamma[x::=v]_{\Gamma v} = \Gamma)$  using subst-gv.simps by auto
    moreover have  $\Theta; \mathcal{B} \vdash_{wf} \Gamma$  using wfG-cons2I by auto
    ultimately show ?thesis by auto
  next
  case False
    have  $\Gamma_1 \neq \text{GNil}$  using wfG-cons2I False by auto
    then obtain G where  $\Gamma_1 = (y, b, c) \#_{\Gamma} G$  using GCons-eq-append-conv wfG-cons2I by auto
    hence  $*\Gamma = G @ (x, b', c') \#_{\Gamma} \Gamma_2$  using wfG-cons2I by auto
    hence  $((y, b, c) \#_{\Gamma} \Gamma[x::=v]_{\Gamma v} = (y, b, c[x::=v]_{cv}) \#_{\Gamma} \Gamma[x::=v]_{\Gamma v})$  using subst-gv.simps False
by auto
    moreover have  $\Theta; \mathcal{B} \vdash_{wf} (y, b, c[x::=v]_{cv}) \#_{\Gamma} \Gamma[x::=v]_{\Gamma v}$  proof(rule Wellformed.wfG-cons2I)
      show  $\langle c[x::=v]_{cv} \in \{\text{TRUE}, \text{FALSE}\} \rangle$  using subst-cv.simps wfG-cons2I by auto
      show  $\langle \Theta; \mathcal{B} \vdash_{wf} \Gamma[x::=v]_{\Gamma v} \rangle$  using wfG-cons2I * by auto
      have  $\Gamma = (G @ ((x, b', c') \#_{\Gamma} \text{GNil})) @ \Gamma_2$  using * append-g-assoc by auto
      hence atom y #  $\Gamma_2$  using fresh-suffix wfG-cons2I by metis
      hence atom y #  $v'$  using wfG-cons2I wfV-x-fresh by metis
      thus  $\langle \text{atom } y \# \Gamma[x::=v]_{\Gamma v} \rangle$  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 $\Gamma::\Gamma$ **and** $\Gamma':\Gamma$ **and** $v::v$ **and** $e::e$ **and** $c::c$ **and** $\tau::\tau$ **and** $ts::(\text{string}*\tau)$ **list** **and** $\Delta::\Delta$ **and** $b::b$
and $ftq::\text{fun-tyt-p}$ **and** $ft::\text{fun-tyt-p}$ **and** $ce::ce$ **and** $td::\text{type-def}$

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**

$\Theta; \Phi; \mathcal{B}; \Gamma; \Delta; tid; dc; t \vdash_{wf} cs : b \implies \Gamma = \Gamma_1 @ ((x, b', c') \#_{\Gamma} \Gamma_2) \implies \Theta; \mathcal{B}; \Gamma_2 \vdash_{wf} v' : b'$
 $\implies \Theta; \Phi; \mathcal{B}; \Gamma[x::=v]_{\Gamma v}; \Delta[x::=v]_{\Delta v}; tid; dc; t \vdash_{wf} \text{subst-branchv } cs \ x \ v' : b$ **and**

$\Theta; \Phi; \mathcal{B}; \Gamma; \Delta; tid; dclist \vdash_{wf} css : b \implies \Gamma = \Gamma_1 @ ((x, b', c') \#_{\Gamma} \Gamma_2) \implies \Theta; \mathcal{B}; \Gamma_2 \vdash_{wf} v' : b'$
 $\implies \Theta; \Phi; \mathcal{B}; \Gamma[x::=v]_{\Gamma v}; \Delta[x::=v]_{\Delta v}; tid; dclist \vdash_{wf} \text{subst-branchlv } css \ x \ v' : b$ **and**

$\Theta \vdash_{wf} (\Phi::\Phi) \implies \text{True}$ **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**

$\Theta; \Phi \vdash_{wf} ftq \implies \text{True}$ **and**

$\Theta; \Phi; \mathcal{B} \vdash_{wf} ft \implies \text{True}$

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 **and** Γ_1

rule:*wfE-wfS-wfCS-wfCSS-wfPhi-wfD-wfFTQ-wfFT.strong-induct*)

case (*wfE-valI* $\Theta \Gamma v b$)

then show *?case using subst-vv.simps wf-intros wf-subst1*

by (*metis subst-ev.simps(1)*)

next

case (*wfE-plusI* $\Theta \Gamma v1 v2$)

then show *?case using subst-vv.simps wf-intros wf-subst1 by auto*

next

case (*wfE-leqI* $\Theta \Phi \Gamma \Delta v1 v2$)

then show *?case*

using *subst-vv.simps subst-ev.simps subst-ev.simps wf-subst1 Wellformed.wfE-leqI*

by *auto*

next

case (*wfE-fstI* $\Theta \Gamma v1 b1 b2$)

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.prem(1)*

wfE-fstI.prem(2) wfE-wfS-wfCS-wfCSS-wfPhi-wfD-wfFTQ-wfFT.wfE-fstI wf-subst1(1))

qed

next

case (*wfE-sndI* $\Theta \Gamma v1 b1 b2$)

```

then show ?case
  by (metis (full-types) subst-ev.simps wfE-sndI Wellformed.wfE-sndI wf-subst1(1))
next
case (wfE-concatI  $\Theta \Phi \Gamma \Delta v1 v2$ )
then show ?case
  by (metis (full-types) subst-ev.simps wfE-sndI Wellformed.wfE-concatI wf-subst1(1))
next
case (wfE-splitI  $\Theta \Phi \Gamma \Delta v1 v2$ )
then show ?case
  by (metis (full-types) subst-ev.simps wfE-sndI Wellformed.wfE-splitI wf-subst1(1))
next
case (wfE-lenI  $\Theta \Phi \Gamma \Delta v1$ )
then show ?case
  by (metis (full-types) subst-ev.simps wfE-sndI Wellformed.wfE-lenI wf-subst1(1))
next
case (wfE-appI  $\Theta \Phi \Gamma \Delta f x b c \tau s' v$ )
then show ?case
  by (metis (full-types) subst-ev.simps wfE-sndI Wellformed.wfE-appI wf-subst1(1))
next
case (wfE-appPI  $\Theta \Phi \mathcal{B} \Gamma \Delta b' bv1 v1 \tau 1 f1 x1 b1 c1 s1$ )
show ?case proof(subst subst-ev.simps, rule)
  show  $\Theta \vdash_{wf} \Phi$  using wfE-appPI wfX-wfY by metis
  show  $\Theta; \mathcal{B}; \Gamma[x::=v]_{\Gamma v} \vdash_{wf} \Delta[x::=v]_{\Delta v}$  using wfE-appPI by auto
  show  $Some (AF-fundef f1 (AF-fun-typ-some bv1 (AF-fun-typ x1 b1 c1 \tau 1 s1))) = lookup-fun \Phi f1$ 
using wfE-appPI by auto
  show  $\Theta; \mathcal{B}; \Gamma[x::=v]_{\Gamma v} \vdash_{wf} v1[x::=v]_{vv} : b1[bv1::=b]_b$  using wfE-appPI wf-subst1 by auto
  show  $\Theta; \mathcal{B} \vdash_{wf} b'$  using wfE-appPI by auto
  have  $atom bv1 \# \Gamma[x::=v]_{\Gamma v}$  using fresh-subst-gv-if wfE-appPI by metis
  moreover have  $atom bv1 \# v1[x::=v]_{vv}$  using wfE-appPI fresh-subst-if by simp
  moreover have  $atom bv1 \# \Delta[x::=v]_{\Delta v}$  using wfE-appPI fresh-subst-dv-if by simp
  ultimately show  $atom bv1 \# (\Phi, \Theta, \mathcal{B}, \Gamma[x::=v]_{\Gamma v}, \Delta[x::=v]_{\Delta v}, b', v1[x::=v]_{vv}, (b-of \tau 1)[bv1::=b]_b)$ 

  using wfE-appPI fresh-prodN by metis
qed
next
case (wfE-mvarI  $\Theta \Phi \mathcal{B} \Gamma \Delta u \tau$ )
have  $\Theta; \Phi; \mathcal{B}; \Gamma[x::=v]_{\Gamma v}; \Delta[x::=v]_{\Delta v} \vdash_{wf} (AE-mvar u) : b-of \tau[x::=v]_{\tau v}$  proof
  show  $\Theta \vdash_{wf} \Phi$  using wfE-mvarI by auto
  show  $\Theta; \mathcal{B}; \Gamma[x::=v]_{\Gamma v} \vdash_{wf} \Delta[x::=v]_{\Delta v}$  using wfE-mvarI by auto
  show  $(u, \tau[x::=v]_{\tau v}) \in setD \Delta[x::=v]_{\Delta v}$  using wfE-mvarI subst-dv-member by auto
qed
thus ?case using subst-ev.simps b-of-subst by auto
next
case (wfD-emptyI  $\Theta \Gamma$ )
then show ?case using subst-dv.simps wf-intros wf-subst1 by auto
next
case (wfD-cons  $\Theta \mathcal{B} \Gamma \Delta \tau u$ )
moreover hence  $u \notin fst 'setD \Delta[x::=v]_{\Delta v}$  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  $\Theta$ )

```

```

then show ?case by auto
next
case (wfPhi-consI f  $\Theta$   $\Phi$  ft)
then show ?case by auto
next
case (wfS-assertI  $\Theta$   $\Phi$   $\mathcal{B}$   $x2$   $c$   $\Gamma$   $\Delta$   $s$   $b$ )
show ?case unfolding subst-sv.simps proof
  show  $\langle \Theta; \Phi; \mathcal{B}; (x2, B\text{-bool}, c[x::=v]_{cv}) \#_{\Gamma} \Gamma[x::=v]_{\Gamma v}; \Delta[x::=v]_{\Delta v} \vdash_{wf} s[x::=v]_{sv} : b \rangle$ 
    using wfS-assertI(4)[of  $(x2, B\text{-bool}, c) \#_{\Gamma} \Gamma_1 x$ ] wfS-assertI by auto

  show  $\langle \Theta; \mathcal{B}; \Gamma[x::=v]_{\Gamma v} \vdash_{wf} c[x::=v]_{cv} \rangle$  using wfS-assertI wf-subst1 by auto
  show  $\langle \Theta; \mathcal{B}; \Gamma[x::=v]_{\Gamma v} \vdash_{wf} \Delta[x::=v]_{\Delta v} \rangle$  using wfS-assertI wf-subst1 by auto
  show  $\langle atom\ x2 \# (\Phi, \Theta, \mathcal{B}, \Gamma[x::=v]_{\Gamma v}, \Delta[x::=v]_{\Delta v}, c[x::=v]_{cv}, b, s[x::=v]_{sv}) \rangle$ 
    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- $\tau$ -def wfS-assertI)
    by(simp add: fresh-prodN fresh-subst-v-if subst-v-s-def wfS-assertI)
qed
next
case (wfS-letI  $\Theta$   $\Phi$   $\mathcal{B}$   $\Gamma$   $\Delta$   $e$   $b1$   $y$   $s$   $b2$ )
have  $\Theta; \Phi; \mathcal{B}; \Gamma[x::=v]_{\Gamma v}; \Delta[x::=v]_{\Delta v} \vdash_{wf} LET\ y = (e[x::=v]_{ev})\ IN\ (s[x::=v]_{sv}) : b2$ 
proof
  show  $\langle \Theta; \Phi; \mathcal{B}; \Gamma[x::=v]_{\Gamma v}; \Delta[x::=v]_{\Delta v} \vdash_{wf} e[x::=v]_{ev} : b1 \rangle$  using wfS-letI by auto
  have  $\langle \Theta; \Phi; \mathcal{B}; ((y, b1, TRUE) \#_{\Gamma} \Gamma)[x::=v]_{\Gamma v}; \Delta[x::=v]_{\Delta v} \vdash_{wf} s[x::=v]_{sv} : b2 \rangle$ 
    using wfS-letI(6) wfS-letI append-g.simps by metis
  thus  $\langle \Theta; \Phi; \mathcal{B}; (y, b1, TRUE) \#_{\Gamma} \Gamma[x::=v]_{\Gamma v}; \Delta[x::=v]_{\Delta v} \vdash_{wf} s[x::=v]_{sv} : b2 \rangle$ 
    using wfS-letI subst-gv.simps by auto
  show  $\langle \Theta; \mathcal{B}; \Gamma[x::=v]_{\Gamma v} \vdash_{wf} \Delta[x::=v]_{\Delta v} \rangle$  using wfS-letI by auto
  show  $\langle atom\ y \# (\Phi, \Theta, \mathcal{B}, \Gamma[x::=v]_{\Gamma v}, \Delta[x::=v]_{\Delta v}, e[x::=v]_{ev}, b2) \rangle$ 
    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- $\tau$ -def wfS-letI)
done
qed
thus ?case using subst-sv.simps wfS-letI by auto
next
case (wfS-let2I  $\Theta$   $\Phi$   $\mathcal{B}$   $\Gamma$   $\Delta$   $s1$   $\tau$   $y$   $s2$   $b$ )
have  $\Theta; \Phi; \mathcal{B}; \Gamma[x::=v]_{\Gamma v}; \Delta[x::=v]_{\Delta v} \vdash_{wf} LET\ y : \tau[x::=v]_{\tau v} = (s1[x::=v]_{sv})\ IN\ (s2[x::=v]_{sv})$ 
: b
proof
  show  $\langle \Theta; \Phi; \mathcal{B}; \Gamma[x::=v]_{\Gamma v}; \Delta[x::=v]_{\Delta v} \vdash_{wf} s1[x::=v]_{sv} : b\text{-of}\ (\tau[x::=v]_{\tau v}) \rangle$  using wfS-let2I
b-of-subst by simp
  have  $\langle \Theta; \Phi; \mathcal{B}; ((y, b\text{-of}\ \tau, TRUE) \#_{\Gamma} \Gamma)[x::=v]_{\Gamma v}; \Delta[x::=v]_{\Delta v} \vdash_{wf} s2[x::=v]_{sv} : b \rangle$ 
    using wfS-let2I append-g.simps by metis
  thus  $\langle \Theta; \Phi; \mathcal{B}; (y, b\text{-of}\ \tau[x::=v]_{\tau v}, TRUE) \#_{\Gamma} \Gamma[x::=v]_{\Gamma v}; \Delta[x::=v]_{\Delta v} \vdash_{wf} s2[x::=v]_{sv} : b \rangle$ 

```

```

    using wfS-let2I subst-gv.simps append-g.simps using b-of-subst by simp
  show ( Θ ; B ; Γ[x::=v]Γv ⊢wf τ[x::=v]Γτv ) using wfS-let2I wf-subst1 by metis
  show (atom u # (Φ, Θ, B, Γ[x::=v]Γv, Δ[x::=v]Δv, s1[x::=v]sv, b, τ[x::=v]Γτv)
    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 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 ; B ; Γ[x::=v]Γv ⊢wf τ[x::=v]Γτv ) using wfS-varI wf-subst1 by auto
  have b-of (τ[x::=v]Γτv) = b-of τ using b-of-subst by auto
  thus ( Θ ; B ; Γ[x::=v]Γv ⊢wf v[x::=v]Γvv : b-of τ[x::=v]Γτv ) 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, B, Γ[x::=v]Γv, Δ[x::=v]Δv, τ[x::=v]Γτv, v[x::=v]Γvv, b)
    unfolding fresh-prodN apply(auto simp add: wfS-varI)
    using wfS-varI fresh-subst-gv * fresh-subst-dv by metis+
  show ( Θ ; Φ ; B ; Γ[x::=v]Γv ; (u, τ[x::=v]Γτv) #Δ Δ[x::=v]Δv ⊢wf s[x::=v]sv : b ) using
wfS-varI by auto
qed
next
case (wfS-assignI u τ Δ Θ B Γ Φ v)
show ?case proof(subst subst-sv.simps, rule wf-intros)
  show ((u, τ[x::=v]Γτv) ∈ setD Δ[x::=v]Δv) using subst-dv-iff wfS-assignI using subst-dv-fst-eq
    using subst-dv-member by auto
  show ( Θ ; B ; Γ[x::=v]Γv ⊢wf Δ[x::=v]Δv ) using wfS-assignI by auto
  show ( Θ ; B ; Γ[x::=v]Γv ⊢wf v[x::=v]Γvv : b-of τ[x::=v]Γτv ) using wfS-assignI b-of-subst wf-subst1
by auto
  show Θ ⊢wf Φ using wfS-assignI by auto
qed
next
case (wfS-matchI Θ B Γ v tid dclist Δ Φ cs b)
show ?case proof(subst subst-sv.simps, rule wf-intros)
  show ( Θ ; B ; Γ[x::=v]Γv ⊢wf v[x::=v]Γvv : 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 ; B ; Γ[x::=v]Γv ; Δ[x::=v]Δv ; tid ; dclist ⊢wf subst-branchlv cs x v' : b ) using
wfS-matchI by simp
  show Θ ; B ; Γ[x::=v]Γv ⊢wf Δ[x::=v]Δv using wfS-matchI by auto
  show Θ ⊢wf Φ using wfS-matchI by auto
qed
next
case (wfS-branchI Θ Φ B y τ Γ Δ s b tid dc)
have Θ ; Φ ; B ; Γ[x::=v]Γv ; Δ[x::=v]Δv ; tid ; dc ; τ ⊢wf dc y ⇒ (s[x::=v]sv) : b
proof
  have ( Θ ; Φ ; B ; ((y, b-of τ, TRUE) #Γ Γ)[x::=v]Γv ; Δ[x::=v]Δv ⊢wf s[x::=v]sv : b )
    using wfS-branchI append-g.simps by metis

```

```

thus  $\langle \Theta ; \Phi ; \mathcal{B} ; (y, b\text{-of } \tau, TRUE) \#_{\Gamma} \Gamma[x::=v]_{\Gamma v} ; \Delta[x::=v]_{\Delta v} \vdash_{wf} s[x::=v]_{sv} : b \rangle$ 
  using subst-gv.simps b-of-subst wfS-branchI by simp
show  $\langle atom\ y \# (\Phi, \Theta, \mathcal{B}, \Gamma[x::=v]_{\Gamma v}, \Delta[x::=v]_{\Delta v}, \Gamma[x::=v]_{\Gamma v}, \tau) \rangle$ 
  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  $\langle \Theta ; \mathcal{B} ; \Gamma[x::=v]_{\Gamma v} \vdash_{wf} \Delta[x::=v]_{\Delta v} \rangle$  using wfS-branchI by auto
qed
thus ?case using subst-branchv.simps wfS-branchI by auto

next
  case (wfS-finalI  $\Theta \Phi \mathcal{B} \Gamma \Delta tid\ dclist' cs\ b\ dclist$ )
  then show ?case using subst-branchlv.simps wf-intros by metis
next
  case (wfS-cons  $\Theta \Phi \mathcal{B} \Gamma \Delta tid\ dclist' cs\ b\ css\ dclist$ )
  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  $\Theta ; \mathcal{B} \vdash_{wf} \Gamma' @ (x, b, c0[z0::=V\text{-var } x]_{cv}) \#_{\Gamma} \Gamma$  and  $wfV \Theta \mathcal{B} \Gamma v\ b$ 
  shows  $\Theta ; \mathcal{B} \vdash_{wf} \Gamma'[x::=v]_{\Gamma v} @ \Gamma$ 
  using assms wf-subst subst-g-inside-simple by auto

lemma wfG-member-subst:
  assumes  $(x1, b1, c1) \in toSet (\Gamma' @ \Gamma)$  and  $wfG \Theta \mathcal{B} (\Gamma' @ ((x, b, c) \#_{\Gamma} \Gamma))$  and  $x \neq x1$ 
  shows  $\exists c1'. (x1, b1, c1') \in toSet ((\Gamma'[x::=v]_{\Gamma v}) @ \Gamma)$ 
proof –
  consider  $(lhs) (x1, b1, c1) \in toSet \Gamma' \mid (rhs) (x1, b1, c1) \in toSet \Gamma$  using append-g-toSetU assms
by auto
  thus ?thesis proof(cases)
    case lhs
    hence  $(x1, b1, c1[x::=v]_{cv}) \in toSet (\Gamma'[x::=v]_{\Gamma v})$  using wfG-inside-fresh[THEN subst-gv-member-iff[OF lhs]] assms by metis
    hence  $(x1, b1, c1[x::=v]_{cv}) \in toSet (\Gamma'[x::=v]_{\Gamma v} @ \Gamma)$  using append-g-toSetU by auto
    then show ?thesis by auto
  next
    case rhs
    hence  $(x1, b1, c1) \in toSet (\Gamma'[x::=v]_{\Gamma v} @ \Gamma)$  using append-g-toSetU by auto
    then show ?thesis by auto
  qed
qed

lemma wfG-member-subst2:
  assumes  $(x1, b1, c1) \in toSet (\Gamma' @ ((x, b, c) \#_{\Gamma} \Gamma))$  and  $wfG \Theta \mathcal{B} (\Gamma' @ ((x, b, c) \#_{\Gamma} \Gamma))$  and  $x \neq x1$ 
  shows  $\exists c1'. (x1, b1, c1') \in toSet ((\Gamma'[x::=v]_{\Gamma v}) @ \Gamma)$ 

```

proof –

consider $(lhs) (x1, b1, c1) \in toSet \Gamma' \mid (rhs) (x1, b1, c1) \in toSet \Gamma$ **using** *append-g-toSetU* *assms*
by *auto*
thus *?thesis* **proof**(*cases*)
 case *lhs*
 hence $(x1, b1, c1[x::=v]_{cv}) \in toSet (\Gamma'[x::=v]_{\Gamma v})$ **using** *wfG-inside-fresh*[*THEN subst-gv-member-iff*[*OF lhs*]] *assms* **by** *metis*
 hence $(x1, b1, c1[x::=v]_{cv}) \in toSet (\Gamma'[x::=v]_{\Gamma v} @ \Gamma)$ **using** *append-g-toSetU* **by** *auto*
 then show *?thesis* **by** *auto*
next
 case *rhs*
 hence $(x1, b1, c1) \in toSet (\Gamma'[x::=v]_{\Gamma v} @ \Gamma)$ **using** *append-g-toSetU* **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* $x \not\# \Gamma$

proof –

have $wfG \ P \ B \ ((x, b, c) \#_{\Gamma} \Gamma)$ **using** *wfG-suffix* *assms* **by** *auto*
thus *?thesis* **using** *wfG-elim*s **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 $\Gamma::\Gamma$ **and** $\Gamma'::\Gamma$ **and** $v::v$ **and** $e::e$ **and** $c::c$ **and** $\tau::\tau$ **and** $ts::(string*\tau)$ *list* **and** $\Delta::\Delta$ **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 $\Theta; B'; \Gamma \vdash_{wf} v : b' \implies \{|bv|\} = B' \implies \Theta; B \vdash_{wf} b \implies \Theta; B; \Gamma[bv::=b]_{\Gamma b} \vdash_{wf} v[bv::=b]_{vb} : b'[bv::=b]_{bb}$ **and**
 $\Theta; B'; \Gamma \vdash_{wf} c \implies \{|bv|\} = B' \implies \Theta; B \vdash_{wf} b \implies \Theta; B; \Gamma[bv::=b]_{\Gamma b} \vdash_{wf} c[bv::=b]_{cb}$ **and**
 $\Theta; B' \vdash_{wf} \Gamma \implies \{|bv|\} = B' \implies \Theta; B \vdash_{wf} b \implies \Theta; B \vdash_{wf} \Gamma[bv::=b]_{\Gamma b}$ **and**
 $\Theta; B'; \Gamma \vdash_{wf} \tau \implies \{|bv|\} = B' \implies \Theta; B \vdash_{wf} b \implies \Theta; B; \Gamma[bv::=b]_{\Gamma b} \vdash_{wf} \tau[bv::=b]_{\tau b}$ **and**

```

     $\Theta; \mathcal{B}; \Gamma \vdash_{wf} ts \Rightarrow \text{True}$  and
     $\vdash_{wf} \Theta \Rightarrow \text{True}$  and
     $\Theta; B' \vdash_{wf} b' \Rightarrow \{|bv|\} = B' \Rightarrow \Theta; B \vdash_{wf} b \Rightarrow \Theta; B \vdash_{wf} b'[bv::=b]_{bb}$  and
     $\Theta; B'; \Gamma \vdash_{wf} ce : b' \Rightarrow \{|bv|\} = B' \Rightarrow \Theta; B \vdash_{wf} b \Rightarrow \Theta; B; \Gamma[bv::=b]_{\Gamma b} \vdash_{wf}$ 
     $ce[bv::=b]_{ceb} : b'[bv::=b]_{bb}$  and
     $\Theta \vdash_{wf} td \Rightarrow \text{True}$ 
  proof(nominal-induct
     $b'$  and  $c$  and  $\Gamma$  and  $\tau$  and  $ts$  and  $\Theta$  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  $\Theta$   $\mathcal{B}$ )
  then show ?case using subst-bb.simps wf-intros wfX-wfY by metis
next
  case (wfB-boolI  $\Theta$   $\mathcal{B}$ )
  then show ?case using subst-bb.simps wf-intros wfX-wfY by metis
next
  case (wfB-unitI  $\Theta$   $\mathcal{B}$ )
  then show ?case using subst-bb.simps wf-intros wfX-wfY by metis
next
  case (wfB-bitvecI  $\Theta$   $\mathcal{B}$ )
  then show ?case using subst-bb.simps wf-intros wfX-wfY by metis
next
  case (wfB-pairI  $\Theta$   $\mathcal{B}$   $b1$   $b2$ )
  then show ?case using subst-bb.simps wf-intros wfX-wfY by metis
next
  case (wfB-consI  $\Theta$   $s$  dclist  $\mathcal{B}$ )
  then show ?case using subst-bb.simps Wellformed.wfB-consI by simp
next
  case (wfB-appI  $\Theta$   $ba$   $s$  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  $\Theta$   $\mathcal{B}1$   $\Gamma$   $b1$   $c$   $x$ )
  show ?case unfolding subst-vb.simps proof
    show  $\Theta; B \vdash_{wf} \Gamma[bv::=b]_{\Gamma b}$  using wfV-varI by auto
    show  $\text{Some}(b1[bv::=b]_{bb}, c[bv::=b]_{cb}) = \text{lookup } \Gamma[bv::=b]_{\Gamma b} x$  using subst-b-lookup wfV-varI by
    simp
  qed
next
  case (wfV-litI  $\Theta$   $\mathcal{B}$   $\Gamma$   $l$ )
  then show ?case using Wellformed.wfV-litI subst-b-base-for-lit by simp
next
  case (wfV-pairI  $\Theta$   $\mathcal{B}1$   $\Gamma$   $v1$   $b1$   $v2$   $b2$ )
  show ?case unfolding subst-vb.simps proof(subst subst-bb.simps, rule)
    show  $\Theta; B; \Gamma[bv::=b]_{\Gamma b} \vdash_{wf} v1[bv::=b]_{vb} : b1[bv::=b]_{bb}$  using wfV-pairI by simp
    show  $\Theta; B; \Gamma[bv::=b]_{\Gamma b} \vdash_{wf} v2[bv::=b]_{vb} : b2[bv::=b]_{bb}$  using wfV-pairI by simp
  qed
next
  case (wfV-consI  $s$  dclist  $\Theta$   $dc$   $x$   $b' c$   $\mathcal{B}' \Gamma v$ )
  show ?case unfolding subst-vb.simps proof(subst subst-bb.simps, rule Wellformed.wfV-consI)
    show  $1: AF\text{-typedef } s \text{ dclist} \in \text{set } \Theta$  using wfV-consI by auto
    show  $2:(dc, \{\!| x : b' \mid c \!\}) \in \text{set dclist}$  using wfV-consI by auto

```



```

have  $\Theta ; B ; \Gamma[bv ::= b]_{\Gamma b} \vdash_{wf} v[bv ::= b]_{vb} : b'[bv ::= b]_{bb}$  using wfV-conspI by auto
moreover hence  $\text{supp } b' = \{\}$  using 1 2 wfTh-lookup-supp-empty  $\tau.\text{supp } wfX\text{-}wfY$  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  $\Theta ; B ; \Gamma[bv ::= b]_{\Gamma b} \vdash_{wf} v[bv ::= b]_{vb} : b'$  using wfV-conspI by simp
qed
next

case (wfV-conspI s bva dclist  $\Theta$  dc x b' c  $\mathcal{B}'$  ba  $\Gamma$  v)
have  $\ast : \text{atom } bv \# b'$  using wfTh-poly-supp-b [of s bva dclist  $\Theta$  dc x b' c] fresh-def wfX-wfY (atom bva
 $\# bv$ )
by (metis insert-iff not-self-fresh singleton-insert-inj-eq' subsetI subset-antisym wfV-conspI wfV-conspI.hyps(4)
wfV-conspI.prem(2))
show ?case unfolding subst-vb.simps subst-bb.simps proof
show  $\langle AF\text{-typedef-poly } s \text{ bva dclist} \in \text{set } \Theta \rangle$  using wfV-conspI by auto
show  $\langle (dc, \{ x : b' \mid c \}) \in \text{set dclist} \rangle$  using wfV-conspI by auto

thus  $\langle \Theta ; B \vdash_{wf} ba[bv ::= b]_{bb} \rangle$  using wfV-conspI by metis
have  $\text{atom } bva \# \Gamma[bv ::= b]_{\Gamma b}$  using fresh-subst-if subst-b- $\Gamma$ -def wfV-conspI by metis
moreover have  $\text{atom } bva \# ba[bv ::= b]_{bb}$  using fresh-subst-if subst-b-b-def wfV-conspI by metis
moreover have  $\text{atom } bva \# v[bv ::= b]_{vb}$  using fresh-subst-if subst-b-v-def wfV-conspI by metis
ultimately show  $\langle \text{atom } bva \# (\Theta, B, \Gamma[bv ::= b]_{\Gamma b}, ba[bv ::= b]_{bb}, v[bv ::= b]_{vb}) \rangle$ 
unfolding fresh-prodN using wfV-conspI fresh-def supp-fset by auto
show  $\langle \Theta ; B ; \Gamma[bv ::= b]_{\Gamma b} \vdash_{wf} v[bv ::= b]_{vb} : b'[bva ::= ba[bv ::= b]_{bb}]_{bb} \rangle$ 
using wfV-conspI subst-bb-commute [of bv b' bva ba b]  $\ast$  wfV-conspI by metis
qed
next

case (wfTI z  $\Theta$   $\mathcal{B}'$   $\Gamma'$   $b' c$ )
show ?case proof(subst subst-tb.simps, rule Wellformed.wfTI)
show  $\text{atom } z \# (\Theta, B, \Gamma[bv ::= b]_{\Gamma b})$  using wfTI subst-g-b-x-fresh by simp
show  $\Theta ; B \vdash_{wf} b'[bv ::= b]_{bb}$  using wfTI by auto
show  $\Theta ; B ; (z, b'[bv ::= b]_{bb}, \text{TRUE}) \#_{\Gamma} \Gamma'[bv ::= b]_{\Gamma b} \vdash_{wf} c[bv ::= b]_{cb}$  using wfTI by simp
qed
next

case (wfC-eqI  $\Theta$   $\mathcal{B}'$   $\Gamma$  e1  $b' e2$ )
thus ?case using Wellformed.wfC-eqI subst-db.simps subst-cb.simps wfC-eqI by metis
next

case (wfG-nilI  $\Theta$   $\mathcal{B}'$ )
then show ?case using Wellformed.wfG-nilI subst-gb.simps by simp
next

case (wfG-cons1I  $c' \Theta$   $\mathcal{B}'$   $\Gamma' x b'$ )
show ?case proof(subst subst-gb.simps, rule Wellformed.wfG-cons1I)
show  $c'[bv ::= b]_{cb} \notin \{\text{TRUE}, \text{FALSE}\}$  using wfG-cons1I(1)
by(nominal-induct c' rule: c.strong-induct, auto+)
show  $\Theta ; B \vdash_{wf} \Gamma'[bv ::= b]_{\Gamma b}$  using wfG-cons1I by auto
show  $\text{atom } x \# \Gamma'[bv ::= b]_{\Gamma b}$  using wfG-cons1I subst-g-b-x-fresh by auto
show  $\Theta ; B ; (x, b'[bv ::= b]_{bb}, \text{TRUE}) \#_{\Gamma} \Gamma'[bv ::= b]_{\Gamma b} \vdash_{wf} c'[bv ::= b]_{cb}$  using wfG-cons1I by
auto
show  $\Theta ; B \vdash_{wf} b'[bv ::= b]_{bb}$  using wfG-cons1I by auto
qed
next

```

```

case (wfG-cons2I c'  $\Theta$  B'  $\Gamma'$  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  $\Theta$  ; B  $\vdash_{wf}$   $\Gamma'$ [bv::=b] $\Gamma_b$  using wfG-cons2I by auto
  show atom x  $\#$   $\Gamma'$ [bv::=b] $\Gamma_b$  using wfG-cons2I subst-g-b-x-fresh by auto
  show  $\Theta$  ; B  $\vdash_{wf}$  b'[bv::=b] $b_b$  using wfG-cons2I by auto
qed
next

case (wfCE-valI  $\Theta$  B  $\Gamma$  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  $\Theta$  B  $\Gamma$  v1 v2)
then show ?case using subst-bb.simps subst-ceb.simps wf-intros wfX-wfY
  by metis
next
case (wfCE-leqI  $\Theta$  B  $\Gamma$  v1 v2)
then show ?case using subst-bb.simps subst-ceb.simps wf-intros wfX-wfY
  by metis
next
case (wfCE-fstI  $\Theta$  B  $\Gamma$  v1 b1 b2)
then show ?case
  by (metis (no-types) subst-bb.simps(5) subst-ceb.simps(3) wfCE-fstI.hyps(2)
    wfCE-fstI.prem(1) wfCE-fstI.prem(2) Wellformed.wfCE-fstI)
next
case (wfCE-sndI  $\Theta$  B  $\Gamma$  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.prem(2) Wellformed.wfCE-sndI)
next
case (wfCE-concatI  $\Theta$  B  $\Gamma$  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.prem(1) wfCE-concatI.prem(2)

    Wellformed.wfCE-concatI by auto
qed
next
case (wfCE-lenI  $\Theta$  B  $\Gamma$  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  $\Gamma::\Gamma$  and  $\Gamma'::\Gamma$  and  $v::v$  and  $e::e$  and  $c::c$  and  $\tau::\tau$  and  $ts::(\text{string}*\tau)$  list and  $\Delta::\Delta$  and  $b::b$ 
  and  $ftq::\text{fun-ty}p-q$  and  $ft::\text{fun-ty}p$  and  $s::s$  and  $b'::b$  and  $ce::ce$  and  $td::\text{type-def}$ 

```

and $cs::branch-s$ **and** $css::branch-list$
shows $\Theta ; \Phi ; B' ; \Gamma ; \Delta \vdash_{wf} e : b' \implies \{|bv|\} = B' \implies \Theta ; B \vdash_{wf} b \implies \Theta ; \Phi ; B ;$
 $\Gamma[bv::=b]_{\Gamma b} ; \Delta[bv::=b]_{\Delta b} \vdash_{wf} e[bv::=b]_{eb} : b'[bv::=b]_{bb}$ **and**
 $\Theta ; \Phi ; \mathcal{B} ; \Gamma ; \Delta \vdash_{wf} s : b \implies True$ **and**
 $\Theta ; \Phi ; \mathcal{B} ; \Gamma ; \Delta ; tid ; dc ; t \vdash_{wf} cs : b \implies True$ **and**
 $\Theta ; \Phi ; \mathcal{B} ; \Gamma ; \Delta ; tid ; dclist \vdash_{wf} css : b \implies True$ **and**
 $\Theta \vdash_{wf} (\Phi::\Phi) \implies True$ **and**
 $\Theta ; B' ; \Gamma \vdash_{wf} \Delta \implies \{|bv|\} = B' \implies \Theta ; B \vdash_{wf} b \implies \Theta ; B ; \Gamma[bv::=b]_{\Gamma b} \vdash_{wf}$
 $\Delta[bv::=b]_{\Delta b}$ **and**
 $\Theta ; \Phi \vdash_{wf} ftq \implies True$ **and**
 $\Theta ; \Phi ; \mathcal{B} \vdash_{wf} ft \implies True$
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 ($wfE-valI \Theta' \Phi' \mathcal{B}' \Gamma' \Delta' v' b'$)
then show *?case unfolding subst-vb.simps subst-eb.simps using wf-b-subst1(1) Wellformed.wfE-valI*
by auto
next
case ($wfE-plusI \Theta \Phi \mathcal{B} \Gamma \Delta v1 v2$)
then show *?case unfolding subst-eb.simps*
using $wf-b-subst-lemmas wf-b-subst1(1)$ *Wellformed.wfE-plusI*
proof –
have $\forall b ba v g f ts. ((ts ; f ; g[bv::=ba]_{\Gamma b} \vdash_{wf} v[bv::=ba]_{vb} : b[bv::=ba]_{bb}) \vee \neg ts ; \mathcal{B} ; g \vdash_{wf} v :$
 $b) \vee \neg ts ; f \vdash_{wf} ba$
using $wfE-plusI.premis(1)$ $wf-b-subst1(1)$ **by force**
then show $\Theta ; \Phi ; B ; \Gamma[bv::=b]_{\Gamma b} ; \Delta[bv::=b]_{\Delta b} \vdash_{wf} [plus v1[bv::=b]_{vb} v2[bv::=b]_{vb}]^e :$
 $B-int[bv::=b]_{bb}$
by (*metis (full-types) wfE-plusI.hyps(1) wfE-plusI.hyps(4) wfE-plusI.hyps(5) wfE-plusI.hyps(6)*
 $wfE-plusI.premis(1) wfE-plusI.premis(2) wfE-wfS-wfCS-wfCSS-wfPhi-wfD-wfFTQ-wfFT.wfE-plusI wf-b-subst-lemmas(84)$)
qed
next
case ($wfE-leqI \Theta \Phi \mathcal{B} \Gamma \Delta v1 v2$)
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 ($wfE-fstI \Theta \Phi \mathcal{B} \Gamma \Delta v1 b1 b2$)
then show *?case unfolding subst-eb.simps using wf-b-subst-lemmas(84) wf-b-subst1(1) Well-*
formed.wfE-fstI
by (*metis wf-b-subst-lemmas(87)*)
next
case ($wfE-sndI \Theta \Phi \mathcal{B} \Gamma \Delta v1 b1 b2$)
then show *?case unfolding subst-eb.simps using wf-b-subst-lemmas(86) wf-b-subst1(1) Well-*
formed.wfE-sndI
by (*metis wf-b-subst-lemmas(87)*)
next
case ($wfE-concatI \Theta \Phi \mathcal{B} \Gamma \Delta v1 v2$)
then show *?case unfolding subst-eb.simps using wf-b-subst-lemmas(86) wf-b-subst1(1) Well-*

formed.wfE-concatI
by (*metis wf-b-subst-lemmas*(89))

next
case (*wfE-splitI* $\Theta \Phi \mathcal{B} \Gamma \Delta v1 v2$)
then show *?case unfolding subst-eb.simps using wf-b-subst-lemmas*(86) *wf-b-subst1*(1) *Well-formed.wfE-splitI*
by (*metis wf-b-subst-lemmas*(84) *wf-b-subst-lemmas*(87) *wf-b-subst-lemmas*(89))

next
case (*wfE-lenI* $\Theta \Phi \mathcal{B} \Gamma \Delta v1$)
then show *?case unfolding subst-eb.simps using wf-b-subst-lemmas*(86) *wf-b-subst1*(1) *Well-formed.wfE-lenI*
by (*metis wf-b-subst-lemmas*(84) *wf-b-subst-lemmas*(89))

next
case (*wfE-appI* $\Theta \Phi \mathcal{B}' \Gamma \Delta f x b' c \tau s v$)
hence *bf: atom bv # 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 $\Theta ; \Phi ; B ; \Gamma[bv::=b]_{\Gamma b} ; \Delta[bv::=b]_{\Delta b} \vdash_{wf} (AE\text{-}app\ f\ (v[bv::=b]_{vb})) : (b\text{-of}\ (\tau[bv::=b]_{\tau b}))$
proof
show $\Theta \vdash_{wf} \Phi$ **using** *wfE-appI by auto*
show $\Theta ; B ; \Gamma[bv::=b]_{\Gamma b} \vdash_{wf} \Delta[bv::=b]_{\Delta b}$ **using** *wfE-appI by simp*
have *atom bv # 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 Φf*
using *wfE-appI by simp*
show $\Theta ; B ; \Gamma[bv::=b]_{\Gamma b} \vdash_{wf} v[bv::=b]_{vb} : b'$ **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* $\Theta \Phi \mathcal{B} \Gamma \Delta b' bv1 v1 \tau 1 f x1 b1 c1 s1$)
then have **: atom bv # b1 using wfPhi-f-supp(1) wfE-appPI(7,11)*
by (*metis fresh-def fresh-finset singleton-iff subsetD fresh-def supp-at-base wfE-appPI.hyps(1)*)
thm *Wellformed.wfE-appPI*
have $\Theta ; \Phi ; B ; \Gamma[bv::=b]_{\Gamma b} ; \Delta[bv::=b]_{\Delta b} \vdash_{wf} AE\text{-}appP\ f\ b'[bv::=b]_{bb}\ (v1[bv::=b]_{vb}) : (b\text{-of}\ \tau 1)[bv1::=b'[bv::=b]_{bb}]_b$
proof
show $\langle \Theta \vdash_{wf} \Phi \rangle$ **using** *wfE-appPI by auto*
show $\langle \Theta ; B ; \Gamma[bv::=b]_{\Gamma b} \vdash_{wf} \Delta[bv::=b]_{\Delta b} \rangle$ **using** *wfE-appPI by auto*
show $\langle \Theta ; B \vdash_{wf} b'[bv::=b]_{bb} \rangle$ **using** *wfE-appPI wf-b-subst1 by auto*
have *atom bv1 # $\Gamma[bv::=b]_{\Gamma b}$ using fresh-subst-if subst-b- Γ -def wfE-appPI by metis*
moreover have *atom bv1 # $b'[bv::=b]_{bb}$ using fresh-subst-if subst-b-b-def wfE-appPI by metis*
moreover have *atom bv1 # $v1[bv::=b]_{vb}$ using fresh-subst-if subst-b-v-def wfE-appPI by metis*
moreover have *atom bv1 # $\Delta[bv::=b]_{\Delta b}$ using fresh-subst-if subst-b- Δ -def wfE-appPI by metis*
moreover have *atom bv1 # $(b\text{-of}\ \tau 1)[bv1::=b'[bv::=b]_{bb}]_{bb}$ using fresh-subst-if subst-b-b-def wfE-appPI by metis*
ultimately show *atom bv1 # $(\Phi, \Theta, B, \Gamma[bv::=b]_{\Gamma b}, \Delta[bv::=b]_{\Delta b}, b'[bv::=b]_{bb}, v1[bv::=b]_{vb}, (b\text{-of}\ \tau 1)[bv1::=b'[bv::=b]_{bb}]_b)$*

```

 $\tau 1)[bv1 ::= b'[bv ::= b]_{bb}]_b)$ 
  using wfE-appPI using fresh-def fresh-prodN subst-b-b-def by metis
  show  $\langle \text{Some } (AF\text{-fundef } f \ (AF\text{-fun-typ-some } bv1 \ (AF\text{-fun-typ } x1 \ b1 \ c1 \ \tau 1 \ s1))) = \text{lookup-fun } \Phi \ f \rangle$ 
using wfE-appPI by auto

  have  $\langle \Theta ; B ; \Gamma[bv ::= b]_{\Gamma b} \vdash_{wf} v1[bv ::= b]_{vb} : b1[bv1 ::= b]_b[bv ::= b]_{bb} \rangle$ 
    using wfE-appPI subst-b-b-def * wf-b-subst1 by metis
  thus  $\langle \Theta ; B ; \Gamma[bv ::= b]_{\Gamma b} \vdash_{wf} v1[bv ::= b]_{vb} : b1[bv1 ::= b'[bv ::= b]_{bb}]_b \rangle$ 
    using subst-bb-commute subst-b-b-def * by auto
qed
moreover have atom bv  $\#$  b-of  $\tau 1$  proof -
  have supp (b-of  $\tau 1$ )  $\subseteq \{ \text{atom } bv1 \}$  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-finset 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  $\Theta \Phi \mathcal{B}' \Gamma \Delta u \tau$ )

have  $\Theta ; \Phi ; B ; \text{subst-gb } \Gamma \text{ bv } b ; \text{subst-db } \Delta \text{ bv } b \vdash_{wf} (AE\text{-mvar } u)[bv ::= b]_{eb} : (b\text{-of } (\tau[bv ::= b]_{\tau b}))$ 

proof(subst subst-eb.simps, rule Wellformed.wfE-mvarI)
  show  $\Theta \vdash_{wf} \Phi$  using wfE-mvarI by simp
  show  $\Theta ; B ; \Gamma[bv ::= b]_{\Gamma b} \vdash_{wf} \Delta[bv ::= b]_{\Delta b}$  using wfE-mvarI by metis
  show  $(u, \tau[bv ::= b]_{\tau b}) \in \text{setD } \Delta[bv ::= b]_{\Delta b}$ 
    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  $\Theta \Phi \mathcal{B} \Gamma \Delta s1 s2 b$ )
then show ?case using subst-bb.simps wf-intros wfX-wfY by metis

next
case (wfD-emptyI  $\Theta \mathcal{B}' \Gamma$ )
then show ?case using subst-db.simps Wellformed.wfD-emptyI wf-b-subst1 by simp
next
case (wfD-cons  $\Theta \mathcal{B}' \Gamma' \Delta \tau u$ )
show ?case proof(subst subst-db.simps, rule Wellformed.wfD-cons )
  show  $\Theta ; B ; \Gamma'[bv ::= b]_{\Gamma b} \vdash_{wf} \Delta[bv ::= b]_{\Delta b}$  using wfD-cons by auto
  show  $\Theta ; B ; \Gamma'[bv ::= b]_{\Gamma b} \vdash_{wf} \tau[bv ::= b]_{\tau b}$  using wfD-cons wf-b-subst1 by auto
  show  $u \notin \text{fst } \text{'setD } \Delta[bv ::= b]_{\Delta b}$  using wfD-cons subst-b-lookup-d by metis
qed
next
case (wfS-assertI  $\Theta \Phi \mathcal{B} x c \Gamma \Delta s b$ )
show ?case by auto
qed(auto)

lemmas wf-b-subst = wf-b-subst1 wf-b-subst2

lemma wfT-subst-wfT:

```

fixes $\tau::\tau$ **and** $b'::b$ **and** $bv::bv$
assumes $\Theta ; \{|bv|\} ; (x, b, c) \#_{\Gamma} GNil \vdash_{wf} \tau$ **and** $\Theta ; B \vdash_{wf} b'$
shows $\Theta ; B ; (x, b[bv::=b]_{bb}, c[bv::=b]_{cb}) \#_{\Gamma} GNil \vdash_{wf} (\tau[bv::=b]_{\tau b})$
proof –
have $\Theta ; B ; ((x, b, c) \#_{\Gamma} GNil)[bv::=b]_{\Gamma b} \vdash_{wf} (\tau[bv::=b]_{\tau b})$
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 $\Gamma::\Gamma$ **and** $\Gamma':\Gamma$ **and** $v::v$ **and** $e::e$ **and** $c::c$ **and** $\tau::\tau$ **and** $ts::(\text{string}*\tau)$ **list** **and** $\Delta::\Delta$ **and** $b::b$
and $ftq::\text{fun-ty-p-q}$ **and** $ft::\text{fun-ty-p}$ **and** $ce::ce$ **and** $td::\text{type-def}$ **and** $s::s$
and $cs::\text{branch-s}$ **and** $css::\text{branch-list}$ **and** $\Theta::\Theta$
shows $\Theta ; \mathcal{B} ; \Gamma \vdash_{wf} v : b' \implies \Gamma = (x, b, c2) \#_{\Gamma} G \implies \Theta ; \mathcal{B} ; (x, b, c1) \#_{\Gamma} G \vdash_{wf} c2$
 $\implies \Theta ; \mathcal{B} ; (x, b, c1) \#_{\Gamma} G \vdash_{wf} v : b'$ **and**
 $\Theta ; \mathcal{B} ; \Gamma \vdash_{wf} c \implies \Gamma = (x, b, c2) \#_{\Gamma} G \implies \Theta ; \mathcal{B} ; (x, b, c1) \#_{\Gamma} G \vdash_{wf} c2 \implies$
 $\Theta ; \mathcal{B} ; (x, b, c1) \#_{\Gamma} G \vdash_{wf} c$ **and**
 $\Theta ; \mathcal{B} \vdash_{wf} \Gamma \implies \text{True}$ **and**
 $\Theta ; \mathcal{B} ; \Gamma \vdash_{wf} \tau \implies \text{True}$ **and**
 $\Theta ; \mathcal{B} ; \Gamma \vdash_{wf} ts \implies \text{True}$ **and**
 $\vdash_{wf} \Theta \implies \text{True}$ **and**
 $\Theta ; \mathcal{B} \vdash_{wf} b \implies \text{True}$ **and**
 $\Theta ; \mathcal{B} ; \Gamma \vdash_{wf} ce : b' \implies \Gamma = (x, b, c2) \#_{\Gamma} G \implies \Theta ; \mathcal{B} ; (x, b, c1) \#_{\Gamma} G \vdash_{wf} c2 \implies \Theta ;$
 $\mathcal{B} ; (x, b, c1) \#_{\Gamma} G \vdash_{wf} ce : b'$ **and**
 $\Theta \vdash_{wf} td \implies \text{True}$

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 **and** Γ_1
rule:wfV-wfC-wfG-wfT-wfTs-wfTh-wfB-wfCE-wfTD.strong-induct
case (*wfV-varI* $\Theta \mathcal{B} \Gamma b' c' x'$)
have $wbg : \Theta ; \mathcal{B} \vdash_{wf} (x, b, c1) \#_{\Gamma} G$ **using** *wfC-wf wfV-varI by simp*
show *?case proof(cases $x=x'$)*
case *True*
have $\text{Some } (b', c1) = \text{lookup } ((x, b, c1) \#_{\Gamma} G) x'$ **using** *lookup.simps wfV-varI using True by*
auto
then show *?thesis using Wellformed.wfV-varI wbg by simp*
next
case *False*
then have $\text{Some } (b', c') = \text{lookup } ((x, b, c1) \#_{\Gamma} G) x'$ **using** *lookup.simps wfV-varI*
by *simp*
then show *?thesis using Wellformed.wfV-varI wbg by simp*
qed
next
case (*wfV-conspI* $s bv dclist \Theta dc x1 b' c \mathcal{B} b1 \Gamma v$)
show *?case proof*
show $\langle AF\text{-typedef-poly } s bv dclist \in \text{set } \Theta \rangle$ **using** *wfV-conspI by auto*
show $\langle (dc, \{x1 : b' \mid c\}) \in \text{set } dclist \rangle$ **using** *wfV-conspI by auto*
show $\langle \Theta ; \mathcal{B} \vdash_{wf} b1 \rangle$ **using** *wfV-conspI by auto*
show $\langle \text{atom } bv \nmid (\Theta, \mathcal{B}, (x, b, c1) \#_{\Gamma} G, b1, v) \rangle$ **unfolding** *fresh-prodN fresh-GCons using*
wfV-conspI fresh-prodN fresh-GCons by simp

```

    show  $\langle \Theta; \mathcal{B}; (x, b, c1) \#_{\Gamma} G \vdash_{wf} v : b'[bv::=b1]_{bb} \rangle$  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* :: $\Theta \Rightarrow \mathcal{B} \Rightarrow \Gamma \Rightarrow \tau \Rightarrow \tau \Rightarrow \text{bool}$ (- ; - ; - \vdash - \lesssim - [50, 50, 50] 50) **where**

subtype-baseI: \llbracket
 $\text{atom } x \nmid (\Theta, \mathcal{B}, \Gamma, z, c, z', c') ;$
 $\Theta; \mathcal{B}; \Gamma \vdash_{wf} \{ z : b \mid c \};$
 $\Theta; \mathcal{B}; \Gamma \vdash_{wf} \{ z' : b \mid c' \};$
 $\Theta; \mathcal{B} ; (x, b, c[z ::= [x]^v]_v) \#_{\Gamma} \Gamma \models c'[z' ::= [x]^v]_v$
 $\rrbracket \Rightarrow$
 $\Theta; \mathcal{B}; \Gamma \vdash \{ z : b \mid c \} \lesssim \{ z' : b \mid c' \}$

equivariance *subtype*

nominal-inductive *subtype*

avoids *subtype-baseI*: x

proof(*goal-cases*)

case (1 $\Theta \mathcal{B} \Gamma z b c z' c' x$)

then show ?*case* **using** *fresh-star-def 1* **by force**

next

case (2 $\Theta \mathcal{B} \Gamma z b c z' c' x$)

then show ?*case* **by auto**

qed

inductive-cases *subtype-elim*:

$\Theta; \mathcal{B}; \Gamma \vdash \{ z : b \mid c \} \lesssim \{ z' : b \mid c' \}$

$\Theta; \mathcal{B}; \Gamma \vdash \tau_1 \lesssim \tau_2$

9.2 Literals

The type synthesised has the constraint that z equates to the literal

inductive *infer-l* :: $l \Rightarrow \tau \Rightarrow \text{bool}$ (\vdash - \Rightarrow - [50, 50] 50) **where**

infer-trueI: $\vdash L\text{-true} \Rightarrow \{ z : B\text{-bool} \mid [[z]^v]^{ce} == [[L\text{-true}]^v]^{ce} \}$

| *infer-falseI*: $\vdash L\text{-false} \Rightarrow \{ z : B\text{-bool} \mid [[z]^v]^{ce} == [[L\text{-false}]^v]^{ce} \}$

| *infer-natI*: $\vdash L\text{-num } n \Rightarrow \{ z : B\text{-int} \mid [[z]^v]^{ce} == [[L\text{-num } n]^v]^{ce} \}$

| *infer-unitI*: $\vdash L\text{-unit} \Rightarrow \{ z : B\text{-unit} \mid [[z]^v]^{ce} == [[L\text{-unit}]^v]^{ce} \}$
| *infer-bitvecI*: $\vdash L\text{-bitvec } bv \Rightarrow \{ z : B\text{-bitvec} \mid [[z]^v]^{ce} == [[L\text{-bitvec } bv]^v]^{ce} \}$

nominal-inductive *infer-l* .
equivariance *infer-l*

inductive-cases *infer-l-elim*[*elim*!]:

$\vdash L\text{-true} \Rightarrow \tau$
 $\vdash L\text{-false} \Rightarrow \tau$
 $\vdash L\text{-num } n \Rightarrow \tau$
 $\vdash L\text{-unit} \Rightarrow \tau$
 $\vdash L\text{-bitvec } x \Rightarrow \tau$
 $\vdash l \Rightarrow \tau$

lemma *infer-l-form2*[*simp*]:

shows $\exists z. \vdash l \Rightarrow (\{ z : \text{base-for-lit } l \mid [[z]^v]^{ce} == [[l]^v]^{ce} \})$

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* :: $\Theta \Rightarrow \mathcal{B} \Rightarrow \Gamma \Rightarrow v \Rightarrow \tau \Rightarrow \text{bool}$ (- ; - ; - \vdash - \Rightarrow - [50, 50, 50] 50) **where**

infer-v-varI: \llbracket

$\Theta; \mathcal{B} \vdash_{wf} \Gamma ;$
 $\text{Some } (b, c) = \text{lookup } \Gamma \ x ;$
 $\text{atom } z \ \sharp \ x ; \text{atom } z \ \sharp \ (\Theta, \mathcal{B}, \Gamma)$

$\rrbracket \Rightarrow$

$\Theta; \mathcal{B}; \Gamma \vdash [x]^v \Rightarrow \{ z : b \mid [[z]^v]^{ce} == [[x]^v]^{ce} \}$

| *infer-v-litI*: \llbracket

$\Theta; \mathcal{B} \vdash_{wf} \Gamma ;$
 $\vdash l \Rightarrow \tau$

$\rrbracket \Rightarrow$

$\Theta; \mathcal{B}; \Gamma \vdash [l]^v \Rightarrow \tau$

| *infer-v-pairI*: \llbracket

```

    atom z # (v1, v2); atom z # (Θ, B, Γ) ;
    Θ; B; Γ ⊢ (v1::v) ⇒ t1 ;
    Θ; B; Γ ⊢ (v2::v) ⇒ t2
  ] ⇒
    Θ; B; Γ ⊢ V-pair v1 v2 ⇒ (⊥ z : B-pair (b-of t1) (b-of t2) | [[z]v]ce == [[v1, v2]v]ce ⊥)

| infer-v-consI: [
  AF-typedef s dclist ∈ set Θ;
  (dc, tc) ∈ set dclist ;
  Θ; B; Γ ⊢ v ⇒ tv ;
  Θ; B; Γ ⊢ tv ≲ tc ;
  atom z # v ; atom z # (Θ, B, Γ)
] ⇒
  Θ; B; Γ ⊢ V-cons s dc v ⇒ (⊥ z : B-id s | [[z]v]ce == [ V-cons s dc v ]ce ⊥)

| infer-v-conspI: [
  AF-typedef-poly s bv dclist ∈ set Θ;
  (dc, tc) ∈ set dclist ;
  Θ; B; Γ ⊢ v ⇒ tv;
  Θ; B; Γ ⊢ tv ≲ tc[bv::=b]τb ;
  atom z # (Θ, B, Γ, v, b);
  atom bv # (Θ, B, Γ, v, b);
  Θ; B ⊢wf b
] ⇒
  Θ; B; Γ ⊢ V-consp s dc b v ⇒ (⊥ z : B-app s b | [[z]v]ce == (CE-val (V-consp s dc b v)) ⊥)

```

equivariance *infer-v*

nominal-inductive *infer-v*

avoids *infer-v-conspI*: bv **and** z | *infer-v-varI*: z | *infer-v-pairI*: z | *infer-v-consI*: z

proof(goal-cases)

case (1 Θ B Γ b c x z)

hence atom z # ⊥ z : b | [[z]^v]^{ce} == [[x]^v]^{ce} ⊥ **using** τ.fresh **by** simp

then show ?case **unfolding** fresh-star-def **using** 1 **by** simp

next

case (2 Θ B Γ b c x z)

then show ?case **by** auto

next

case (3 z v1 v2 Θ B Γ t1 t2)

hence atom z # ⊥ z : [b-of t1 , b-of t2]^b | [[z]^v]^{ce} == [[v1, v2]^v]^{ce} ⊥ **using** τ.fresh **by** simp

then show ?case **unfolding** fresh-star-def **using** 3 **by** simp

next

case (4 z v1 v2 Θ B Γ t1 t2)

then show ?case **by** auto

next

case (5 s dclist Θ dc tc B Γ v tv z)

hence atom z # ⊥ z : B-id s | [[z]^v]^{ce} == [V-cons s dc v]^{ce} ⊥ **using** τ.fresh b.fresh pure-fresh

by auto

moreover have atom z # V-cons s dc v **using** v.fresh 5 **using** v.fresh fresh-prodN pure-fresh **by** metis

then show ?case **unfolding** fresh-star-def **using** 5 **by** simp

next

case (6 s dclist Θ dc tc B Γ v tv z)

```

then show ?case by auto
next
case (7 s bv dclist  $\Theta$  dc tc  $\mathcal{B}$   $\Gamma$  v tv b z)
hence atom bv  $\#$  V-consp s dc b v using v.fresh fresh-prodN pure-fresh by metis
moreover then have atom bv  $\#$   $\{ z : B\text{-id } s \mid [ [ z ]^v ]^{ce} == [ V\text{-consp } s \text{ dc } b \text{ v} ]^{ce} \}$ 
  using  $\tau$ .fresh ce.fresh v.fresh by auto
moreover have atom z  $\#$  V-consp s dc b v using v.fresh fresh-prodN pure-fresh 7 by metis
moreover then have atom z  $\#$   $\{ z : B\text{-id } s \mid [ [ z ]^v ]^{ce} == [ V\text{-consp } s \text{ dc } b \text{ v} ]^{ce} \}$ 
  using  $\tau$ .fresh ce.fresh v.fresh by auto
ultimately show ?case using fresh-star-def 7 by force
next
case (8 s bv dclist  $\Theta$  dc tc  $\mathcal{B}$   $\Gamma$  v tv b z)
then show ?case by auto
qed

```

```

inductive-cases infer-v-elim[elim!]:
 $\Theta; \mathcal{B}; \Gamma \vdash V\text{-var } x \Rightarrow \tau$ 
 $\Theta; \mathcal{B}; \Gamma \vdash V\text{-lit } l \Rightarrow \tau$ 
 $\Theta; \mathcal{B}; \Gamma \vdash V\text{-pair } v1 \text{ } v2 \Rightarrow \tau$ 
 $\Theta; \mathcal{B}; \Gamma \vdash V\text{-cons } s \text{ dc } v \Rightarrow \tau$ 
 $\Theta; \mathcal{B}; \Gamma \vdash V\text{-pair } v1 \text{ } v2 \Rightarrow (\{ z : b \mid c \})$ 
 $\Theta; \mathcal{B}; \Gamma \vdash V\text{-pair } v1 \text{ } v2 \Rightarrow (\{ z : [ b1 \text{ , } b2 ]^b \mid [[z]^v]^{ce} == [[v1, v2]^v]^{ce} \})$ 
 $\Theta; \mathcal{B}; \Gamma \vdash V\text{-consp } s \text{ dc } b \text{ v} \Rightarrow \tau$ 

```

9.4 Introductions

```

inductive check-v ::  $\Theta \Rightarrow \mathcal{B} \Rightarrow \Gamma \Rightarrow v \Rightarrow \tau \Rightarrow \text{bool}$  (-; -; -  $\vdash$  -  $\Leftarrow$  - [50, 50, 50] 50) where
check-v-subtypeI:  $\llbracket \Theta; \mathcal{B}; \Gamma \vdash \tau1 \lesssim \tau2; \Theta; \mathcal{B}; \Gamma \vdash v \Rightarrow \tau1 \rrbracket \Longrightarrow \Theta; \mathcal{B}; \Gamma \vdash v \Leftarrow \tau2$ 
equivariance check-v
nominal-inductive check-v .

```

```

inductive-cases check-v-elim[elim!]:
 $\Theta; \mathcal{B}; \Gamma \vdash v \Leftarrow \tau$ 

```

9.5 Expressions

Type synthesis for expressions

```

inductive infer-e ::  $\Theta \Rightarrow \Phi \Rightarrow \mathcal{B} \Rightarrow \Gamma \Rightarrow \Delta \Rightarrow e \Rightarrow \tau \Rightarrow \text{bool}$  (-; -; -; -; -  $\vdash$  -  $\Rightarrow$  - [50, 50, 50, 50] 50) where

```

```

infer-e-valI:  $\llbracket$ 
  ( $\Theta; \mathcal{B}; \Gamma \vdash_{wf} \Delta$ ) ;
  ( $\Theta \vdash_{wf} (\Phi :: \Phi)$ ) ;
  ( $\Theta; \mathcal{B}; \Gamma \vdash v \Rightarrow \tau$ )  $\rrbracket \Longrightarrow$ 
   $\Theta; \Phi; \mathcal{B}; \Gamma; \Delta \vdash (AE\text{-val } v) \Rightarrow \tau$ 

| infer-e-plusI:  $\llbracket$ 
   $\Theta; \mathcal{B}; \Gamma \vdash_{wf} \Delta$  ;
   $\Theta \vdash_{wf} (\Phi :: \Phi)$  ;
   $\Theta; \mathcal{B}; \Gamma \vdash v1 \Rightarrow \{ z1 : B\text{-int} \mid c1 \}$  ;
   $\Theta; \mathcal{B}; \Gamma \vdash v2 \Rightarrow \{ z2 : B\text{-int} \mid c2 \}$  ;

```

$$\begin{array}{l}
\text{atom } z\beta \# (AE\text{-op Plus } v1 \ v2); \text{atom } z\beta \# \Gamma \parallel \Rightarrow \\
\Theta; \Phi; \mathcal{B}; \Gamma; \Delta \vdash AE\text{-op Plus } v1 \ v2 \Rightarrow \llbracket z\beta : B\text{-int} \mid [[z\beta]^v]^{ce} == (CE\text{-op Plus } [v1]^{ce} \ [v2]^{ce}) \rrbracket \\
\\
| \text{infer-e-leqI: } \llbracket \\
\Theta; \mathcal{B}; \Gamma \vdash_{wf} \Delta; \\
\Theta \vdash_{wf} (\Phi :: \Phi); \\
\Theta; \mathcal{B}; \Gamma \vdash v1 \Rightarrow \llbracket z1 : B\text{-int} \mid c1 \rrbracket; \\
\Theta; \mathcal{B}; \Gamma \vdash v2 \Rightarrow \llbracket z2 : B\text{-int} \mid c2 \rrbracket; \\
\text{atom } z\beta \# (AE\text{-op LEq } v1 \ v2); \text{atom } z\beta \# \Gamma \\
\parallel \Rightarrow \\
\Theta; \Phi; \mathcal{B}; \Gamma; \Delta \vdash AE\text{-op LEq } v1 \ v2 \Rightarrow \llbracket z\beta : B\text{-bool} \mid [[z\beta]^v]^{ce} == (CE\text{-op LEq } [v1]^{ce} \ [v2]^{ce}) \rrbracket \\
\\
| \text{infer-e-appI: } \llbracket \\
\Theta; \mathcal{B}; \Gamma \vdash_{wf} \Delta; \\
\Theta \vdash_{wf} (\Phi :: \Phi); \\
Some (AF\text{-fundef } f \ (AF\text{-fun-typ-none } (AF\text{-fun-typ } x \ b \ c \ \tau' \ s')))) = lookup\text{-fun } \Phi \ f; \\
\Theta; \mathcal{B}; \Gamma \vdash v \Leftarrow \llbracket x : b \mid c \rrbracket; \\
\text{atom } x \# (\Theta, \Phi, \mathcal{B}, \Gamma, \Delta, v, \tau); \\
\tau'[x::v]_v = \tau \\
\parallel \Rightarrow \\
\Theta; \Phi; \mathcal{B}; \Gamma; \Delta \vdash AE\text{-app } f \ v \Rightarrow \tau \\
\\
| \text{infer-e-appPI: } \llbracket \\
\Theta; \mathcal{B}; \Gamma \vdash_{wf} \Delta; \\
\Theta \vdash_{wf} (\Phi :: \Phi); \\
\Theta; \mathcal{B} \vdash_{wf} b'; \\
Some (AF\text{-fundef } f \ (AF\text{-fun-typ-some } bv \ (AF\text{-fun-typ } x \ b \ c \ \tau' \ s')))) = lookup\text{-fun } \Phi \ f; \\
\Theta; \mathcal{B}; \Gamma \vdash v \Leftarrow \llbracket x : b[bv::=b]_b \mid c[bv::=b]_b \rrbracket; \text{atom } x \# \Gamma; \\
(\tau'[bv::=b]_b[x::v]_v) = \tau; \\
\text{atom } bv \# (\Theta, \Phi, \mathcal{B}, \Gamma, \Delta, b', v, \tau) \\
\parallel \Rightarrow \\
\Theta; \Phi; \mathcal{B}; \Gamma; \Delta \vdash AE\text{-appP } f \ b' \ v \Rightarrow \tau \\
\\
| \text{infer-e-fstI: } \llbracket \\
\Theta; \mathcal{B}; \Gamma \vdash_{wf} \Delta; \\
\Theta \vdash_{wf} (\Phi :: \Phi); \\
\Theta; \mathcal{B}; \Gamma \vdash v \Rightarrow \llbracket z' : [b1, b2]^b \mid c \rrbracket; \\
\text{atom } z \# AE\text{-fst } v; \text{atom } z \# \Gamma \parallel \Rightarrow \\
\Theta; \Phi; \mathcal{B}; \Gamma; \Delta \vdash AE\text{-fst } v \Rightarrow \llbracket z : b1 \mid [[z]^v]^{ce} == ((CE\text{-fst } [v]^{ce})) \rrbracket \\
\\
| \text{infer-e-sndI: } \llbracket \\
\Theta; \mathcal{B}; \Gamma \vdash_{wf} \Delta; \\
\Theta \vdash_{wf} (\Phi :: \Phi); \\
\Theta; \mathcal{B}; \Gamma \vdash v \Rightarrow \llbracket z' : [b1, b2]^b \mid c \rrbracket; \\
\text{atom } z \# AE\text{-snd } v; \text{atom } z \# \Gamma \parallel \Rightarrow \\
\Theta; \Phi; \mathcal{B}; \Gamma; \Delta \vdash AE\text{-snd } v \Rightarrow \llbracket z : b2 \mid [[z]^v]^{ce} == ((CE\text{-snd } [v]^{ce})) \rrbracket \\
\\
| \text{infer-e-lenI: } \llbracket \\
\Theta; \mathcal{B}; \Gamma \vdash_{wf} \Delta; \\
\Theta \vdash_{wf} (\Phi :: \Phi); \\
\Theta; \mathcal{B}; \Gamma \vdash v \Rightarrow \llbracket z' : B\text{-bitvec} \mid c \rrbracket; \\
\text{atom } z \# AE\text{-len } v; \text{atom } z \# \Gamma \parallel \Rightarrow
\end{array}$$

$$\Theta; \Phi; \mathcal{B}; \Gamma; \Delta \vdash AE-len\ v \Rightarrow \llbracket z : B-int \mid [[z]^v]^{ce} == ((CE-len\ [v]^{ce})) \rrbracket$$

$$\begin{array}{l} | \text{infer-e-mvarI: } \llbracket \\ \quad \Theta; \mathcal{B} \vdash_{wf} \Gamma ; \\ \quad \Theta \vdash_{wf} (\Phi::\Phi) ; \\ \quad \Theta; \mathcal{B}; \Gamma \vdash_{wf} \Delta; \\ \quad (u, \tau) \in setD\ \Delta \rrbracket \Longrightarrow \\ \quad \Theta; \Phi; \mathcal{B}; \Gamma; \Delta \vdash AE-mvar\ u \Rightarrow \tau \end{array}$$

$$\begin{array}{l} | \text{infer-e-concatI: } \llbracket \\ \quad \Theta; \mathcal{B}; \Gamma \vdash_{wf} \Delta ; \\ \quad \Theta \vdash_{wf} (\Phi::\Phi) ; \\ \quad \Theta; \mathcal{B}; \Gamma \vdash v1 \Rightarrow \llbracket z1 : B-bitvec \mid c1 \rrbracket ; \\ \quad \Theta; \mathcal{B}; \Gamma \vdash v2 \Rightarrow \llbracket z2 : B-bitvec \mid c2 \rrbracket ; \\ \quad atom\ z3 \# (AE-concat\ v1\ v2); atom\ z3 \# \Gamma \rrbracket \Longrightarrow \\ \quad \Theta; \Phi; \mathcal{B}; \Gamma; \Delta \vdash AE-concat\ v1\ v2 \Rightarrow \llbracket z3 : B-bitvec \mid [[z3]^v]^{ce} == (CE-concat\ [v1]^{ce}\ [v2]^{ce}) \rrbracket \end{array}$$

$$\begin{array}{l} | \text{infer-e-splitI: } \llbracket \\ \quad \Theta; \mathcal{B}; \Gamma \vdash_{wf} \Delta ; \\ \quad \Theta \vdash_{wf} (\Phi::\Phi); \\ \quad infer-v\ \Theta\ \mathcal{B}\ \Gamma\ v1\ \llbracket z1 : B-bitvec \mid c1 \rrbracket ; \\ \quad check-v\ \Theta\ \mathcal{B}\ \Gamma\ v2\ \llbracket z2 : B-int \mid (CE-op\ LEq\ (CE-val\ (V-lit\ (L-num\ 0)))\ (CE-val\ (V-var\ z2))) == \\ (CE-val\ (V-lit\ L-true))\ AND \\ \quad \quad \quad (CE-op\ LEq\ (CE-val\ (V-var\ z2))\ (CE-len\ (CE-val\ (v1)))) == (CE-val \\ (V-lit\ L-true)) \rrbracket ; \\ \quad atom\ z1 \# (AE-split\ v1\ v2); atom\ z1 \# \Gamma; \\ \quad atom\ z2 \# (AE-split\ v1\ v2); atom\ z2 \# \Gamma; \\ \quad atom\ z3 \# (AE-split\ v1\ v2); atom\ z3 \# \Gamma \\ \rrbracket \Longrightarrow \\ \quad infer-e\ \Theta\ \Phi\ \mathcal{B}\ \Gamma\ \Delta\ (AE-split\ v1\ v2)\ \llbracket z3 : B-pair\ B-bitvec\ B-bitvec \mid \\ \quad \quad \quad ((CE-val\ v1) == (CE-concat\ (CE-fst\ (CE-val\ (V-var\ z3)))\ (CE-snd\ (CE-val\ (V-var \\ z3)))))) \\ \quad \quad \quad AND\ (((CE-len\ (CE-fst\ (CE-val\ (V-var\ z3)))) == (CE-val\ (v2))) \rrbracket \end{array}$$

equivariance *infer-e*

nominal-inductive *infer-e*

avoids *infer-e-appI: x | infer-e-appPI: bv | infer-e-splitI: z3 and z1 and z2*

proof(goal-cases)

case (1 $\Theta\ \mathcal{B}\ \Gamma\ \Delta\ \Phi\ f\ x\ b\ c\ \tau'\ s'\ v\ \tau$)

moreover hence $atom\ x \# [f\ v]^e$ **using** *fresh-prodN pure-fresh e.fresh by force*

ultimately show ?case **unfolding** *fresh-star-def* **using** *fresh-prodN e.fresh pure-fresh by simp*

next

case (2 $\Theta\ \mathcal{B}\ \Gamma\ \Delta\ \Phi\ f\ x\ b\ c\ \tau'\ s'\ v\ \tau$)

then show ?case **by** *auto*

next

case (3 $\Theta\ \mathcal{B}\ \Gamma\ \Delta\ \Phi\ b'\ f\ bv\ x\ b\ c\ \tau'\ s'\ v\ \tau$)

moreover hence $atom\ bv \# AE-appP\ f\ b'\ v$ **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 (4 $\Theta\ \mathcal{B}\ \Gamma\ \Delta\ \Phi\ b'\ f\ bv\ x\ b\ c\ \tau'\ s'\ v\ \tau$)

then show ?case by auto
 next
 case (5 $\Theta \mathcal{B} \Gamma \Delta \Phi v1 z1 c1 v2 z2 z3$)
 have atom $z3 \# \{ z3 : [B\text{-bitvec} , B\text{-bitvec}]^b \mid [v1]^{ce} == [\#1 [[z3]^v]^{ce}]^{ce} @@ \#2 [[z3]^v]^{ce}]^{ce} \text{ AND } [\#1 [[z3]^v]^{ce}]^{ce} == [v2]^{ce} \}$
 using $\tau.\text{fresh}$ by simp
 then show ?case unfolding fresh-star-def fresh-prod7 using wfG-fresh-x2 5 by auto
 next
 case (6 $\Theta \mathcal{B} \Gamma \Delta \Phi v1 z1 c1 v2 z2 z3$)
 then show ?case by auto
 qed

inductive-cases infer-e-elim_s[elim]:

$\Theta; \Phi; \mathcal{B}; \Gamma; \Delta \vdash (AE\text{-op Plus } v1 \ v2) \Rightarrow \{ z3 : B\text{-int} \mid [[z3]^v]^{ce} == (CE\text{-op Plus } [v1]^{ce} [v2]^{ce}) \}$
 $\Theta; \Phi; \mathcal{B}; \Gamma; \Delta \vdash (AE\text{-op LEq } v1 \ v2) \Rightarrow \{ z3 : B\text{-bool} \mid [[z3]^v]^{ce} == (CE\text{-op LEq } [v1]^{ce} [v2]^{ce}) \}$
 $\Theta; \Phi; \mathcal{B}; \Gamma; \Delta \vdash (AE\text{-op Plus } v1 \ v2) \Rightarrow \{ z3 : B\text{-int} \mid c \}$
 $\Theta; \Phi; \mathcal{B}; \Gamma; \Delta \vdash (AE\text{-op Plus } v1 \ v2) \Rightarrow \{ z3 : b \mid c \}$
 $\Theta; \Phi; \mathcal{B}; \Gamma; \Delta \vdash (AE\text{-op LEq } v1 \ v2) \Rightarrow \{ z3 : b \mid c \}$
 $\Theta; \Phi; \mathcal{B}; \Gamma; \Delta \vdash (AE\text{-app } f \ v) \Rightarrow \tau$
 $\Theta; \Phi; \mathcal{B}; \Gamma; \Delta \vdash (AE\text{-val } v) \Rightarrow \tau$
 $\Theta; \Phi; \mathcal{B}; \Gamma; \Delta \vdash (AE\text{-fst } v) \Rightarrow \tau$
 $\Theta; \Phi; \mathcal{B}; \Gamma; \Delta \vdash (AE\text{-snd } v) \Rightarrow \tau$
 $\Theta; \Phi; \mathcal{B}; \Gamma; \Delta \vdash (AE\text{-mvar } u) \Rightarrow \tau$
 $\Theta; \Phi; \mathcal{B}; \Gamma; \Delta \vdash (AE\text{-op Plus } v1 \ v2) \Rightarrow \tau$
 $\Theta; \Phi; \mathcal{B}; \Gamma; \Delta \vdash (AE\text{-op LEq } v1 \ v2) \Rightarrow \tau$
 $\Theta; \Phi; \mathcal{B}; \Gamma; \Delta \vdash (AE\text{-op LEq } v1 \ v2) \Rightarrow \{ z3 : B\text{-bool} \mid c \}$
 $\Theta; \Phi; \mathcal{B}; \Gamma; \Delta \vdash (AE\text{-app } f \ v) \Rightarrow \tau[x::=v]_{\tau v}$
 $\Theta; \Phi; \mathcal{B}; \Gamma; \Delta \vdash (AE\text{-op opp } v1 \ v2) \Rightarrow \tau$
 $\Theta; \Phi; \mathcal{B}; \Gamma; \Delta \vdash (AE\text{-len } v) \Rightarrow \tau$
 $\Theta; \Phi; \mathcal{B}; \Gamma; \Delta \vdash (AE\text{-len } v) \Rightarrow \{ z : B\text{-int} \mid [[z]^v]^{ce} == ((CE\text{-len } [v]^{ce})) \}$
 $\Theta; \Phi; \mathcal{B}; \Gamma; \Delta \vdash AE\text{-concat } v1 \ v2 \Rightarrow \tau$
 $\Theta; \Phi; \mathcal{B}; \Gamma; \Delta \vdash AE\text{-concat } v1 \ v2 \Rightarrow (\{ z : b \mid c \})$
 $\Theta; \Phi; \mathcal{B}; \Gamma; \Delta \vdash AE\text{-concat } v1 \ v2 \Rightarrow (\{ z : B\text{-bitvec} \mid [[z]^v]^{ce} == (CE\text{-concat } [v1]^{ce} [v2]^{ce}) \})$
 $\Theta; \Phi; \mathcal{B}; \Gamma; \Delta \vdash (AE\text{-appP } f \ b \ v) \Rightarrow \tau$
 $\Theta; \Phi; \mathcal{B}; \Gamma; \Delta \vdash AE\text{-split } v1 \ v2 \Rightarrow \tau$

nominal-termination (eqvt) by lexicographic-order

9.6 Statements

inductive check-s :: $\Theta \Rightarrow \Phi \Rightarrow \mathcal{B} \Rightarrow \Gamma \Rightarrow \Delta \Rightarrow s \Rightarrow \tau \Rightarrow \text{bool}$ (- ; - ; - ; - ; - \vdash - \Leftarrow - [50, 50, 50, 50, 50] 50) and

check-branch-s :: $\Theta \Rightarrow \Phi \Rightarrow \mathcal{B} \Rightarrow \Gamma \Rightarrow \Delta \Rightarrow \text{tyid} \Rightarrow \text{string} \Rightarrow \tau \Rightarrow v \Rightarrow \text{branch-s} \Rightarrow \tau \Rightarrow \text{bool}$ (- ; - ; - ; - ; - ; - ; - \vdash - \Leftarrow - [50, 50, 50, 50, 50] 50) and

check-branch-list :: $\Theta \Rightarrow \Phi \Rightarrow \mathcal{B} \Rightarrow \Gamma \Rightarrow \Delta \Rightarrow \text{tyid} \Rightarrow (\text{string} * \tau) \text{ list} \Rightarrow v \Rightarrow \text{branch-list} \Rightarrow \tau \Rightarrow \text{bool}$ (- ; - ; - ; - ; - ; - ; - \vdash - \Leftarrow - [50, 50, 50, 50, 50] 50) where

check-valI: \llbracket

$\Theta; \mathcal{B}; \Gamma \vdash_{wf} \Delta;$

$\Theta \vdash_{wf} \Phi;$

$\Theta; \mathcal{B}; \Gamma \vdash v \Rightarrow \tau';$

$\Theta; \mathcal{B}; \Gamma \vdash \tau' \lesssim \tau \rrbracket \implies$

$\Theta; \Phi; \mathcal{B}; \Gamma; \Delta \vdash (AS\text{-val } v) \Leftarrow \tau$

$$\begin{array}{l}
| \text{check-letI}: \llbracket \\
\quad \text{atom } x \# (\Theta, \Phi, \mathcal{B}, \Gamma, \Delta, e, \tau); \\
\quad \text{atom } z \# (x, \Theta, \Phi, \mathcal{B}, \Gamma, \Delta, e, \tau, s); \\
\quad \Theta; \Phi; \mathcal{B}; \Gamma; \Delta \vdash e \Rightarrow \llbracket z : b \mid c \rrbracket; \\
\quad \Theta; \Phi; \mathcal{B}; ((x, b, c[z::=V\text{-var } x]_v) \#_{\Gamma} \Gamma); \Delta \vdash s \Leftarrow \tau \\
\rrbracket \Rightarrow \\
\quad \Theta; \Phi; \mathcal{B}; \Gamma; \Delta \vdash (AS\text{-let } x \ e \ s) \Leftarrow \tau \\
\\
| \text{check-assertI}: \llbracket \\
\quad \text{atom } x \# (\Theta, \Phi, \mathcal{B}, \Gamma, \Delta, c, \tau, s); \\
\quad \Theta; \Phi; \mathcal{B}; ((x, B\text{-bool}, c) \#_{\Gamma} \Gamma); \Delta \vdash s \Leftarrow \tau; \\
\quad \Theta; \mathcal{B}; \Gamma \models c; \\
\quad \Theta; \mathcal{B}; \Gamma \vdash_{wf} \Delta \\
\rrbracket \Rightarrow \\
\quad \Theta; \Phi; \mathcal{B}; \Gamma; \Delta \vdash (AS\text{-assert } c \ s) \Leftarrow \tau \\
\\
| \text{check-branch-s-branchI}: \llbracket \\
\quad \Theta; \mathcal{B}; \Gamma \vdash_{wf} \Delta; \\
\quad \vdash_{wf} \Theta; \\
\quad \Theta; \mathcal{B}; \Gamma \vdash_{wf} \tau; \\
\quad \Theta; \{\mid\}; GNil \vdash_{wf} \text{const}; \\
\quad \text{atom } x \# (\Theta, \Phi, \mathcal{B}, \Gamma, \Delta, \text{tid}, \text{cons}, \text{const}, v, \tau); \\
\quad \Theta; \Phi; \mathcal{B}; ((x, b\text{-of } \text{const}, ([v]^{ce} == [V\text{-cons } \text{tid } \text{cons } [x]^v]^{ce}) \text{ AND } (c\text{-of } \text{const } x)) \#_{\Gamma} \Gamma); \Delta \vdash s \\
\Leftarrow \tau \\
\rrbracket \Rightarrow \\
\quad \Theta; \Phi; \mathcal{B}; \Gamma; \Delta; \text{tid}; \text{cons}; \text{const}; v \vdash (AS\text{-branch } \text{cons } x \ s) \Leftarrow \tau \\
\\
| \text{check-branch-list-consI}: \llbracket \\
\quad \Theta; \Phi; \mathcal{B}; \Gamma; \Delta; \text{tid}; \text{cons}; \text{const}; v \vdash \text{cs} \Leftarrow \tau; \\
\quad \Theta; \Phi; \mathcal{B}; \Gamma; \Delta; \text{tid}; \text{dclist}; v \vdash \text{css} \Leftarrow \tau \\
\rrbracket \Rightarrow \\
\quad \Theta; \Phi; \mathcal{B}; \Gamma; \Delta; \text{tid}; (\text{cons}, \text{const}) \# \text{dclist}; v \vdash AS\text{-cons } \text{cs } \text{css} \Leftarrow \tau \\
\\
| \text{check-branch-list-finalI}: \llbracket \\
\quad \Theta; \Phi; \mathcal{B}; \Gamma; \Delta; \text{tid}; \text{cons}; \text{const}; v \vdash \text{cs} \Leftarrow \tau \\
\rrbracket \Rightarrow \\
\quad \Theta; \Phi; \mathcal{B}; \Gamma; \Delta; \text{tid}; [(\text{cons}, \text{const})]; v \vdash AS\text{-final } \text{cs} \Leftarrow \tau \\
\\
| \text{check-ifI}: \llbracket \\
\quad \text{atom } z \# (\Theta, \Phi, \mathcal{B}, \Gamma, \Delta, v, s1, s2, \tau); \\
\quad (\Theta; \mathcal{B}; \Gamma \vdash v \Leftarrow (\llbracket z : B\text{-bool} \mid TRUE \rrbracket)); \\
\quad \Theta; \Phi; \mathcal{B}; \Gamma; \Delta \vdash s1 \Leftarrow (\llbracket z : b\text{-of } \tau \mid ([v]^{ce} == [[L\text{-true}]^v]^{ce}) \text{ IMP } (c\text{-of } \tau \ z) \rrbracket); \\
\quad \Theta; \Phi; \mathcal{B}; \Gamma; \Delta \vdash s2 \Leftarrow (\llbracket z : b\text{-of } \tau \mid ([v]^{ce} == [[L\text{-false}]^v]^{ce}) \text{ IMP } (c\text{-of } \tau \ z) \rrbracket) \\
\rrbracket \Rightarrow \\
\quad \Theta; \Phi; \mathcal{B}; \Gamma; \Delta \vdash IF \ v \ THEN \ s1 \ ELSE \ s2 \Leftarrow \tau \\
\\
| \text{check-let2I}: \llbracket \\
\quad \text{atom } x \# (\Theta, \Phi, \mathcal{B}, G, \Delta, t, s1, \tau); \\
\quad \Theta; \Phi; \mathcal{B}; G; \Delta \vdash s1 \Leftarrow t; \\
\quad \Theta; \Phi; \mathcal{B}; ((x, b\text{-of } t, c\text{-of } t \ x) \#_{\Gamma} G); \Delta \vdash s2 \Leftarrow \tau \\
\rrbracket \Rightarrow
\end{array}$$

$$\begin{array}{l}
\Theta; \Phi; \mathcal{B}; G; \Delta \vdash (LET\ x : t = s1\ IN\ s2) \Leftarrow \tau \\
\\
| \text{check-varI}: \llbracket \\
\quad atom\ u \# (\Theta, \Phi, \mathcal{B}, \Gamma, \Delta, \tau', v, \tau); \\
\quad \Theta; \mathcal{B}; \Gamma \vdash v \Leftarrow \tau'; \\
\quad \Theta; \Phi; \mathcal{B}; \Gamma; ((u, \tau') \#_{\Delta} \Delta) \vdash s \Leftarrow \tau \\
\rrbracket \Rightarrow \\
\quad \Theta; \Phi; \mathcal{B}; \Gamma; \Delta \vdash (VAR\ u : \tau' = v\ IN\ s) \Leftarrow \tau \\
\\
| \text{check-assignI}: \llbracket \\
\quad \Theta \vdash_{wf} \Phi; \\
\quad \Theta; \mathcal{B}; \Gamma \vdash_{wf} \Delta; \\
\quad (u, \tau) \in setD\ \Delta; \\
\quad \Theta; \mathcal{B}; \Gamma \vdash v \Leftarrow \tau; \\
\quad \Theta; \mathcal{B}; \Gamma \vdash (\{ z : B\text{-unit} \mid TRUE \}) \lesssim \tau' \\
\rrbracket \Rightarrow \\
\quad \Theta; \Phi; \mathcal{B}; \Gamma; \Delta \vdash (u ::= v) \Leftarrow \tau' \\
\\
| \text{check-whileI}: \llbracket \\
\quad \Theta; \Phi; \mathcal{B}; \Gamma; \Delta \vdash s1 \Leftarrow \{ z : B\text{-bool} \mid TRUE \}; \\
\quad \Theta; \Phi; \mathcal{B}; \Gamma; \Delta \vdash s2 \Leftarrow \{ z : B\text{-unit} \mid TRUE \}; \\
\quad \Theta; \mathcal{B}; \Gamma \vdash (\{ z : B\text{-unit} \mid TRUE \}) \lesssim \tau' \\
\rrbracket \Rightarrow \\
\quad \Theta; \Phi; \mathcal{B}; \Gamma; \Delta \vdash WHILE\ s1\ DO\ \{ s2 \} \Leftarrow \tau' \\
\\
| \text{check-seqI}: \llbracket \\
\quad \Theta; \Phi; \mathcal{B}; \Gamma; \Delta \vdash s1 \Leftarrow \{ z : B\text{-unit} \mid TRUE \}; \\
\quad \Theta; \Phi; \mathcal{B}; \Gamma; \Delta \vdash s2 \Leftarrow \tau \\
\rrbracket \Rightarrow \\
\quad \Theta; \Phi; \mathcal{B}; \Gamma; \Delta \vdash s1 ;; s2 \Leftarrow \tau \\
\\
| \text{check-caseI}: \llbracket \\
\quad \Theta; \Phi; \mathcal{B}; \Gamma; \Delta; tid; dclist; v \vdash cs \Leftarrow \tau; \\
\quad (AF\text{-typedef}\ tid\ dclist) \in set\ \Theta; \\
\quad \Theta; \mathcal{B}; \Gamma \vdash v \Leftarrow \{ z : B\text{-id}\ tid \mid TRUE \}; \\
\quad \vdash_{wf} \Theta \\
\rrbracket \Rightarrow \\
\quad \Theta; \Phi; \mathcal{B}; \Gamma; \Delta \vdash AS\text{-match}\ v\ cs \Leftarrow \tau
\end{array}$$

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 ($1\ x\ \Theta\ \Phi\ \mathcal{B}\ \Gamma\ \Delta\ e\ \tau\ z\ s\ b\ c$)

hence *atom* $x \# AS\text{-let}\ x\ e\ s$ **using** *s-branch-s-branch-list.fresh(2)* **by** *auto*

moreover have *atom* $z \# AS\text{-let}\ x\ e\ s$ **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 ($3\ x\ \Theta\ \Phi\ \mathcal{B}\ \Gamma\ \Delta\ c\ \tau\ s$)

hence $atom\ x \# AS\text{-}assert\ c\ s$ **using** $fresh\text{-}prodN\ s\text{-}branch\text{-}s\text{-}branch\text{-}list.fresh\ pure\text{-}fresh$ **by** $auto$
 then show $?case$ **using** $fresh\text{-}star\text{-}def\ 3$ **by** $force$
 next
 case $(5\ \Theta\ \mathcal{B}\ \Gamma\ \Delta\ \tau\ const\ x\ \Phi\ tid\ cons\ v\ s)$
 hence $atom\ x \# AS\text{-}branch\ cons\ x\ s$ **using** $fresh\text{-}prodN\ s\text{-}branch\text{-}s\text{-}branch\text{-}list.fresh\ pure\text{-}fresh$ **by** $auto$
 then show $?case$ **using** $fresh\text{-}star\text{-}def\ 5$ **by** $force$
 next
 case $(7\ z\ \Theta\ \Phi\ \mathcal{B}\ \Gamma\ \Delta\ v\ s1\ s2\ \tau)$
 hence $atom\ z \# AS\text{-}if\ v\ s1\ s2$ **using** $s\text{-}branch\text{-}s\text{-}branch\text{-}list.fresh$ **by** $auto$
 then show $?case$ **using** $7\ fresh\text{-}prodN\ fresh\text{-}star\text{-}def$ **by** $fastforce$
 next
 case $(9\ x\ \Theta\ \Phi\ \mathcal{B}\ G\ \Delta\ t\ s1\ \tau\ s2)$
 hence $atom\ x \# AS\text{-}let2\ x\ t\ s1\ s2$ **using** $s\text{-}branch\text{-}s\text{-}branch\text{-}list.fresh$ **by** $auto$
 thus $?case$ **using** $fresh\text{-}star\text{-}def\ 9$ **by** $force$
 next
 case $(11\ u\ \Theta\ \Phi\ \mathcal{B}\ \Gamma\ \Delta\ \tau'\ v\ \tau\ s)$
 hence $atom\ u \# AS\text{-}var\ u\ \tau'\ v\ s$ **using** $s\text{-}branch\text{-}s\text{-}branch\text{-}list.fresh$ **by** $auto$
 then show $?case$ **using** $fresh\text{-}star\text{-}def\ 11$ **by** $force$
 qed($auto+$)

inductive-cases $check\text{-}s\text{-}elims[elim!]$:

$\Theta; \Phi; \mathcal{B}; \Gamma; \Delta \vdash AS\text{-}val\ v \Leftarrow \tau$
 $\Theta; \Phi; \mathcal{B}; \Gamma; \Delta \vdash AS\text{-}let\ x\ e\ s \Leftarrow \tau$
 $\Theta; \Phi; \mathcal{B}; \Gamma; \Delta \vdash AS\text{-}if\ v\ s1\ s2 \Leftarrow \tau$
 $\Theta; \Phi; \mathcal{B}; \Gamma; \Delta \vdash AS\text{-}let2\ x\ t\ s1\ s2 \Leftarrow \tau$
 $\Theta; \Phi; \mathcal{B}; \Gamma; \Delta \vdash AS\text{-}while\ s1\ s2 \Leftarrow \tau$
 $\Theta; \Phi; \mathcal{B}; \Gamma; \Delta \vdash AS\text{-}var\ u\ t\ v\ s \Leftarrow \tau$
 $\Theta; \Phi; \mathcal{B}; \Gamma; \Delta \vdash AS\text{-}seq\ s1\ s2 \Leftarrow \tau$
 $\Theta; \Phi; \mathcal{B}; \Gamma; \Delta \vdash AS\text{-}assign\ u\ v \Leftarrow \tau$
 $\Theta; \Phi; \mathcal{B}; \Gamma; \Delta \vdash AS\text{-}match\ v\ cs \Leftarrow \tau$
 $\Theta; \Phi; \mathcal{B}; \Gamma; \Delta \vdash AS\text{-}assert\ c\ s \Leftarrow \tau$

inductive-cases $check\text{-}branch\text{-}s\text{-}elims[elim!]$:

$\Theta; \Phi; \mathcal{B}; \Gamma; \Delta; tid; dclist; v \vdash (AS\text{-}final\ cs) \Leftarrow \tau$
 $\Theta; \Phi; \mathcal{B}; \Gamma; \Delta; tid; dclist; v \vdash (AS\text{-}cons\ cs\ css) \Leftarrow \tau$
 $\Theta; \Phi; \mathcal{B}; \Gamma; \Delta; tid; cons; const; v \vdash (AS\text{-}branch\ dc\ x\ s) \Leftarrow \tau$

9.7 Programs

Type check function bodies

inductive $check\text{-}fun\text{-}typ :: \Theta \Rightarrow \Phi \Rightarrow \mathcal{B} \Rightarrow fun\text{-}typ \Rightarrow bool\ (-; -; - \vdash -)$ **where**

$check\text{-}fun\text{-}typI: \llbracket$
 $atom\ x \# (\Theta, \Phi, B, b);$
 $\Theta; \Phi; B; ((x, b, c) \#_{\Gamma} GNil); \llbracket_{\Delta} \vdash s \Leftarrow \tau$
 $\rrbracket \implies$
 $\Theta; \Phi; B \vdash (AF\text{-}fun\text{-}typ\ x\ b\ c\ \tau\ s)$

equivariance $check\text{-}fun\text{-}typ$

```

nominal-inductive check-funtyp
  avoids check-funtypI:  $x$ 
proof(goal-cases)
  case ( $1\ x\ \Theta\ \Phi\ B\ b\ c\ s\ \tau$  )
  hence  $atom\ x\ \sharp\ (AF\text{-}fun\text{-}typ\ x\ b\ c\ \tau\ s)$  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 ( $2\ \Theta\ \Phi\ x\ b\ c\ s\ \tau\ f$ )
  then show  $?case$  by auto
qed

```

```

inductive check-funtypq ::  $\Theta \Rightarrow \Phi \Rightarrow fun\text{-}typ\text{-}q \Rightarrow bool\ ( \ - ; - \vdash - )$  where
check-fundefq-simpleI:  $\llbracket$ 
   $\Theta; \Phi; \{\llbracket\} \vdash (AF\text{-}fun\text{-}typ\ x\ b\ c\ t\ s)$ 
 $\rrbracket \Rightarrow$ 
   $\Theta; \Phi \vdash ((AF\text{-}fun\text{-}typ\text{-}none\ (AF\text{-}fun\text{-}typ\ x\ b\ c\ t\ s)))$ 

```

```

|check-funtypq-polyI:  $\llbracket$ 
   $atom\ bv\ \sharp\ (\Theta, \Phi, (AF\text{-}fun\text{-}typ\ x\ b\ c\ t\ s));$ 
   $\Theta; \Phi; \{|bv|\} \vdash (AF\text{-}fun\text{-}typ\ x\ b\ c\ t\ s)$ 
 $\rrbracket \Rightarrow$ 
   $\Theta; \Phi \vdash (AF\text{-}fun\text{-}typ\text{-}some\ bv\ (AF\text{-}fun\text{-}typ\ x\ b\ c\ t\ s))$ 

```

```

equivariance check-funtypq
nominal-inductive check-funtypq
  avoids check-funtypq-polyI:  $bv$ 
proof(goal-cases)
  case ( $1\ bv\ \Theta\ \Phi\ x\ b\ c\ t\ s$  )
  hence  $atom\ bv\ \sharp\ (AF\text{-}fun\text{-}typ\text{-}some\ bv\ (AF\text{-}fun\text{-}typ\ x\ b\ c\ t\ s))$  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 ( $2\ bv\ \Theta\ \Phi\ ft$  )
  then show  $?case$  by auto
qed

```

```

inductive check-fundef ::  $\Theta \Rightarrow \Phi \Rightarrow fun\text{-}def \Rightarrow bool\ ( \ - ; - \vdash - )$  where
check-fundefI:  $\llbracket$ 
   $\Theta; \Phi \vdash ft$ 
 $\rrbracket \Rightarrow$ 
   $\Theta; \Phi \vdash (AF\text{-}fundef\ f\ ft)$ 

```

```

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-elim[[elim!]]:
  check-funtyp  $\Theta\ \Phi\ B\ ft$ 

```

```

inductive-cases check-funtypq-elim[[elim!]]:

```

```

check-funtypq  $\Theta \Phi$  (AF-fun-typ-none (AF-fun-typ  $x b c \tau s$ ))
check-funtypq  $\Theta \Phi$  (AF-fun-typ-some bv (AF-fun-typ  $x b c \tau s$ ))

```

```

inductive-cases check-fundef-elim[elim!]:
  check-fundef  $\Theta \Phi$  (AF-fundef  $f ftq$ )

```

```

declare[[ simproc add: alpha-lst]]

```

```

nominal-function  $\Delta$ -of :: var-def list  $\Rightarrow \Delta$  where
   $\Delta$ -of [] = DNil
|  $\Delta$ -of ((AV-def  $u t v$ )# $vs$ ) = ( $u, t$ ) # $\Delta$  ( $\Delta$ -of  $vs$ )
apply auto
using eqvt-def  $\Delta$ -of-graph-aux-def neq-Nil-conv old.prod.exhaust apply force
using eqvt-def  $\Delta$ -of-graph-aux-def neq-Nil-conv old.prod.exhaust
by (metis var-def.strong-exhaust)
nominal-termination (eqvt) by lexicographic-order

```

```

inductive check-prog ::  $p \Rightarrow \tau \Rightarrow \text{bool}$  ( $\vdash - \Leftarrow -$ ) where
  [
     $\Theta; \Phi; \{||\}; GNil ; \Delta$ -of  $\mathcal{G} \vdash s \Leftarrow \tau$ 
  ]  $\Rightarrow \vdash (AP$ -prog  $\Theta \Phi \mathcal{G} s) \Leftarrow \tau$ 

```

```

inductive-cases check-prog-elim[elim!]:
   $\vdash (AP$ -prog  $\Theta \Phi \mathcal{G} s) \Leftarrow \tau$ 

```

```

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 $\delta = (u*v) \text{ list}$

nominal-function $\text{update-d} :: \delta \Rightarrow u \Rightarrow v \Rightarrow \delta$ **where**

$\text{update-d } [] \text{ } - = []$
| $\text{update-d } ((u',v')\#\delta) \text{ } u \text{ } v = (\text{if } u = u' \text{ then } ((u,v)\#\delta) \text{ else } ((u',v')\# (\text{update-d } \delta \text{ } u \text{ } v)))$
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 $\text{find-branch} :: dc \Rightarrow \text{branch-list} \Rightarrow \text{branch-s} \Rightarrow \text{bool}$ **where**

$\text{find-branch-finalI: } dc' = dc \implies \text{find-branch } dc' (\text{AS-final } (\text{AS-branch } dc \text{ } x \text{ } s)) (\text{AS-branch } dc \text{ } x \text{ } s)$
| $\text{find-branch-branch-eqI: } dc' = dc \implies \text{find-branch } dc' (\text{AS-cons } (\text{AS-branch } dc \text{ } x \text{ } s) \text{ } css) (\text{AS-branch } dc \text{ } x \text{ } s)$
| $\text{find-branch-branch-neqI: } [dc \neq dc'; \text{find-branch } dc' \text{ } css \text{ } cs] \implies \text{find-branch } dc' (\text{AS-cons } (\text{AS-branch } dc \text{ } x \text{ } s) \text{ } css) \text{ } cs$

equivariance find-branch

nominal-inductive find-branch .

inductive-cases $\text{find-branch-elim}[elim!]$:

$\text{find-branch } dc (\text{AS-final } cs') \text{ } cs$
 $\text{find-branch } dc (\text{AS-cons } cs' \text{ } css) \text{ } cs$

nominal-function $\text{lookup-branch} :: dc \Rightarrow \text{branch-list} \Rightarrow \text{branch-s} \Rightarrow \text{option}$ **where**

$\text{lookup-branch } dc (\text{AS-final } (\text{AS-branch } dc' \text{ } x \text{ } s)) = (\text{if } dc = dc' \text{ then } (\text{Some } (\text{AS-branch } dc' \text{ } x \text{ } s)) \text{ else } \text{None})$
| $\text{lookup-branch } dc (\text{AS-cons } (\text{AS-branch } dc' \text{ } x \text{ } s) \text{ } css) = (\text{if } dc = dc' \text{ then } (\text{Some } (\text{AS-branch } dc' \text{ } x \text{ } s)) \text{ else } \text{lookup-branch } dc \text{ } css)$
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 :: $\Phi \Rightarrow \delta \Rightarrow s \Rightarrow \delta \Rightarrow s \Rightarrow \text{bool}$ (- \vdash < - , - $\rangle \longrightarrow$ < - , - \rangle [50, 50, 50] 50)
where

- | reduce-if-trueI: $\Phi \vdash \langle \delta, AS\text{-if } [L\text{-true}]^v s1\ s2 \rangle \longrightarrow \langle \delta, s1 \rangle$
- | reduce-if-falseI: $\Phi \vdash \langle \delta, AS\text{-if } [L\text{-false}]^v s1\ s2 \rangle \longrightarrow \langle \delta, s2 \rangle$
- | reduce-let-valI: $\Phi \vdash \langle \delta, AS\text{-let } x\ (AE\text{-val } v)\ s \rangle \longrightarrow \langle \delta, s[x::=v]_{sv} \rangle$
- | reduce-let-plusI: $\Phi \vdash \langle \delta, AS\text{-let } x\ (AE\text{-op } Plus\ ((V\text{-lit } (L\text{-num } n1)))\ ((V\text{-lit } (L\text{-num } n2))))\ s \rangle \longrightarrow$
 $\langle \delta, AS\text{-let } x\ (AE\text{-val } (V\text{-lit } (L\text{-num } ((n1)+(n2)))))\ s \rangle$
- | reduce-let-leqI: $b = (\text{if } (n1 \leq n2) \text{ then } L\text{-true} \text{ else } L\text{-false}) \implies$
 $\Phi \vdash \langle \delta, AS\text{-let } x\ ((AE\text{-op } LEq\ (V\text{-lit } (L\text{-num } n1))\ (V\text{-lit } (L\text{-num } n2))))\ s \rangle \longrightarrow$
 $\langle \delta, AS\text{-let } x\ (AE\text{-val } (V\text{-lit } b))\ s \rangle$
- | reduce-let-appI: $\text{Some } (AF\text{-fundef } f\ (AF\text{-fun-typ-none } (AF\text{-fun-typ } z\ b\ c\ \tau\ s'))) = \text{lookup-fun } \Phi\ f \implies$
 $\Phi \vdash \langle \delta, AS\text{-let } x\ ((AE\text{-app } f\ v))\ s \rangle \longrightarrow \langle \delta, AS\text{-let2 } x\ \tau[z::=v]_{\tau v}\ s'[z::=v]_{sv}\ s \rangle$
- | reduce-let-appPI: $\text{Some } (AF\text{-fundef } f\ (AF\text{-fun-typ-some } bv\ (AF\text{-fun-typ } z\ b\ c\ \tau\ s'))) = \text{lookup-fun } \Phi\ f \implies$
 $\Phi \vdash \langle \delta, AS\text{-let } x\ ((AE\text{-appP } f\ b'\ v))\ s \rangle \longrightarrow \langle \delta, AS\text{-let2 } x\ \tau[bv::=b]_{\tau b}[z::=v]_{\tau v}\ s'[bv::=b]_{sb}[z::=v]_{sv}\ s \rangle$
- | reduce-let-fstI: $\Phi \vdash \langle \delta, AS\text{-let } x\ (AE\text{-fst } (V\text{-pair } v1\ v2))\ s \rangle \longrightarrow \langle \delta, AS\text{-let } x\ (AE\text{-val } v1)\ s \rangle$
- | reduce-let-sndI: $\Phi \vdash \langle \delta, AS\text{-let } x\ (AE\text{-snd } (V\text{-pair } v1\ v2))\ s \rangle \longrightarrow \langle \delta, AS\text{-let } x\ (AE\text{-val } v2)\ s \rangle$
- | reduce-let-concatI: $\Phi \vdash \langle \delta, AS\text{-let } x\ (AE\text{-concat } (V\text{-lit } (L\text{-bitvec } v1))\ (V\text{-lit } (L\text{-bitvec } v2)))\ s \rangle \longrightarrow$
 $\langle \delta, AS\text{-let } x\ (AE\text{-val } (V\text{-lit } (L\text{-bitvec } (v1 @ v2))))\ s \rangle$
- | reduce-let-splitI: $\text{split } n\ v\ (v1, v2) \implies \Phi \vdash \langle \delta, AS\text{-let } x\ (AE\text{-split } (V\text{-lit } (L\text{-bitvec } v))\ (V\text{-lit } (L\text{-num } n)))\ s \rangle \longrightarrow$
 $\langle \delta, AS\text{-let } x\ (AE\text{-val } (V\text{-pair } (V\text{-lit } (L\text{-bitvec } v1))\ (V\text{-lit } (L\text{-bitvec } v2))))\ s \rangle$
- | reduce-let-lenI: $\Phi \vdash \langle \delta, AS\text{-let } x\ (AE\text{-len } (V\text{-lit } (L\text{-bitvec } v)))\ s \rangle \longrightarrow$
 $\langle \delta, AS\text{-let } x\ (AE\text{-val } (V\text{-lit } (L\text{-num } (\text{int } (\text{List.length } v)))))\ s \rangle$
- | reduce-let-mvar: $(u, v) \in \text{set } \delta \implies \Phi \vdash \langle \delta, AS\text{-let } x\ (AE\text{-mvar } u)\ s \rangle \longrightarrow \langle \delta, AS\text{-let } x\ (AE\text{-val } v)\ s \rangle$
- | reduce-assert1I: $\Phi \vdash \langle \delta, AS\text{-assert } c\ (AS\text{-val } v) \rangle \longrightarrow \langle \delta, AS\text{-val } v \rangle$
- | reduce-assert2I: $\Phi \vdash \langle \delta, s \rangle \longrightarrow \langle \delta', s' \rangle \implies \Phi \vdash \langle \delta, AS\text{-assert } c\ s \rangle \longrightarrow \langle \delta', AS\text{-assert } c\ s' \rangle$
- | reduce-varI: $\text{atom } u \nVdash \delta \implies \Phi \vdash \langle \delta, AS\text{-var } u\ \tau\ v\ s \rangle \longrightarrow \langle ((u, v) \# \delta), s \rangle$
- | reduce-assignI: $\Phi \vdash \langle \delta, AS\text{-assign } u\ v \rangle \longrightarrow \langle \text{update-d } \delta\ u\ v, AS\text{-val } (V\text{-lit } L\text{-unit}) \rangle$
- | reduce-seq1I: $\Phi \vdash \langle \delta, AS\text{-seq } (AS\text{-val } (V\text{-lit } L\text{-unit}))\ s \rangle \longrightarrow \langle \delta, s \rangle$
- | reduce-seq2I: $\llbracket s1 \neq AS\text{-val } v ; \Phi \vdash \langle \delta, s1 \rangle \longrightarrow \langle \delta', s1' \rangle \rrbracket \implies$
 $\Phi \vdash \langle \delta, AS\text{-seq } s1\ s2 \rangle \longrightarrow \langle \delta', AS\text{-seq } s1'\ s2 \rangle$
- | reduce-let2-valI: $\Phi \vdash \langle \delta, AS\text{-let2 } x\ t\ (AS\text{-val } v)\ s \rangle \longrightarrow \langle \delta, s[x::=v]_{sv} \rangle$
- | reduce-let2I: $\Phi \vdash \langle \delta, s1 \rangle \longrightarrow \langle \delta', s1' \rangle \implies \Phi \vdash \langle \delta, AS\text{-let2 } x\ t\ s1\ s2 \rangle \longrightarrow \langle \delta', AS\text{-let2 } x\ t\ s1'\ s2 \rangle$
- | reduce-caseI: $\llbracket \text{Some } (AS\text{-branch } dc\ x'\ s') = \text{lookup-branch } dc\ cs \rrbracket \implies \Phi \vdash \langle \delta, AS\text{-match } (V\text{-cons } tyid\ dc\ v)\ cs \rangle \longrightarrow \langle \delta, s'[x'::=v]_{sv} \rangle$
- | reduce-whileI: $\llbracket \text{atom } x \nVdash (s1, s2); \text{atom } z \nVdash x \rrbracket \implies$
 $\Phi \vdash \langle \delta, AS\text{-while } s1\ s2 \rangle \longrightarrow$
 $\langle \delta, AS\text{-let2 } x\ (\{ z : B\text{-bool} \mid TRUE \})\ s1\ (AS\text{-if } (V\text{-var } x)\ (AS\text{-seq } s2\ (AS\text{-while } s1\ s2))\ (AS\text{-val } (V\text{-lit } L\text{-unit}))) \rangle$

equivariance *reduce-stmt*
nominal-inductive *reduce-stmt* .

inductive-cases *reduce-stmt-elim*[*elim*!]:

$\Phi \vdash \langle \delta, AS\text{-if } (V\text{-lit } L\text{-true}) s1 s2 \rangle \longrightarrow \langle \delta, s1 \rangle$
 $\Phi \vdash \langle \delta, AS\text{-if } (V\text{-lit } L\text{-false}) s1 s2 \rangle \longrightarrow \langle \delta, s2 \rangle$
 $\Phi \vdash \langle \delta, AS\text{-let } x (AE\text{-val } v) s \rangle \longrightarrow \langle \delta, s^\wedge \rangle$
 $\Phi \vdash \langle \delta, AS\text{-let } x (AE\text{-op Plus } ((V\text{-lit } (L\text{-num } n1))) ((V\text{-lit } (L\text{-num } n2)))) s \rangle \longrightarrow$
 $\langle \delta, AS\text{-let } x (AE\text{-val } (V\text{-lit } (L\text{-num } ((n1)+(n2))))) s \rangle$
 $\Phi \vdash \langle \delta, AS\text{-let } x ((AE\text{-op LEq } (V\text{-lit } (L\text{-num } n1)) (V\text{-lit } (L\text{-num } n2)))) s \rangle \longrightarrow \langle \delta, AS\text{-let } x (AE\text{-val } (V\text{-lit } b)) s \rangle$
 $\Phi \vdash \langle \delta, AS\text{-let } x ((AE\text{-app } f v)) s \rangle \longrightarrow \langle \delta, AS\text{-let2 } x \tau (subst\text{-sv } s' x v) s \rangle$
 $\Phi \vdash \langle \delta, AS\text{-let } x ((AE\text{-len } v)) s \rangle \longrightarrow \langle \delta, AS\text{-let } x v' s \rangle$
 $\Phi \vdash \langle \delta, AS\text{-let } x ((AE\text{-concat } v1 v2)) s \rangle \longrightarrow \langle \delta, AS\text{-let } x v' s \rangle$
 $\Phi \vdash \langle \delta, AS\text{-seq } s1 s2 \rangle \longrightarrow \langle \delta', s' \rangle$
 $\Phi \vdash \langle \delta, AS\text{-let } x ((AE\text{-appP } f b v)) s \rangle \longrightarrow \langle \delta, AS\text{-let2 } x \tau (subst\text{-sv } s' z v) s \rangle$
 $\Phi \vdash \langle \delta, AS\text{-let } x ((AE\text{-split } v1 v2)) s \rangle \longrightarrow \langle \delta, AS\text{-let } x v' s \rangle$
 $\Phi \vdash \langle \delta, AS\text{-assert } c s \rangle \longrightarrow \langle \delta, s' \rangle$

inductive *reduce-stmt-many* :: $\Phi \Rightarrow \delta \Rightarrow s \Rightarrow \delta \Rightarrow s \Rightarrow bool$ $(- \vdash \langle -, - \rangle \longrightarrow^* \langle -, - \rangle [50, 50, 50])$ **where**

reduce-stmt-many-oneI: $\Phi \vdash \langle \delta, s \rangle \longrightarrow \langle \delta', s' \rangle \implies \Phi \vdash \langle \delta, s \rangle \longrightarrow^* \langle \delta', s' \rangle$
reduce-stmt-many-manyI: $\llbracket \Phi \vdash \langle \delta, s \rangle \longrightarrow \langle \delta', s' \rangle ; \Phi \vdash \langle \delta', s' \rangle \longrightarrow^* \langle \delta'', s'' \rangle \rrbracket \implies \Phi \vdash \langle \delta, s \rangle \longrightarrow^* \langle \delta'', s'' \rangle$

nominal-function *convert-fds* :: *fun-def list* \Rightarrow (*f*fun-def*) *list* **where**

convert-fds [] = []
| *convert-fds* ((*AF-fundef f* (*AF-fun-typ-none* (*AF-fun-typ x b c τ s*)))#*fs*) = ((*f, AF-fundef f* (*AF-fun-typ-none* (*AF-fun-typ x b c τ s*)))#*convert-fds fs*)
| *convert-fds* ((*AF-fundef f* (*AF-fun-typ-some bv* (*AF-fun-typ x b c τ s*)))#*fs*) = ((*f, AF-fundef f* (*AF-fun-typ-some bv* (*AF-fun-typ x b c τ s*)))#*convert-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-def*) *list* **where**

convert-tds [] = []
| *convert-tds* ((*AF-typedef s dclist*)#*fs*) = ((*s, AF-typedef s dclist*)#*convert-tds fs*)
| *convert-tds* ((*AF-typedef-poly s bv dclist*)#*fs*) = ((*s, AF-typedef-poly s bv dclist*)#*convert-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**

$\llbracket \text{reduce-stmt-many } \Phi \rrbracket s \delta (AS\text{-val } v) \rrbracket \implies \text{reduce-prog } (AP\text{-prog } \Theta \Phi \rrbracket s) v$

10.2 Reduction Typing

Checks that the store is consistent with Δ

inductive *delta-sim* :: $\Theta \Rightarrow \delta \Rightarrow \Delta \Rightarrow \text{bool} \ (- \vdash - \sim - \ [50,50] \ 50)$ **where**
delta-sim-nilI: $\Theta \vdash [] \sim []_\Delta$
| *delta-sim-consI*: $\llbracket \Theta \vdash \delta \sim \Delta ; \Theta ; \{\|\} ; GNil \vdash v \Leftarrow \tau ; u \notin \text{fst} \text{ ' set } \delta \rrbracket \Longrightarrow \Theta \vdash ((u,v)\#\delta) \sim ((u,\tau)\#\Delta)$

equivariance *delta-sim*
nominal-inductive *delta-sim* .

inductive-cases *delta-sim-elim*s[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 definitions are well-typed

inductive *config-type* :: $\Theta \Rightarrow \Phi \Rightarrow \Delta \Rightarrow \delta \Rightarrow s \Rightarrow \tau \Rightarrow \text{bool} \ (- ; - ; - \vdash \langle -, - \rangle \Leftarrow - \ [50, 50, 50] \ 50)$
where
config-typeI: $\llbracket \Theta ; \Phi ; \{\|\} ; GNil ; \Delta \vdash s \Leftarrow \tau ;$
 $(\forall \text{ fd} \in \text{set } \Phi. \Theta ; \Phi \vdash \text{fd}) ;$
 $\Theta \vdash \delta \sim \Delta \rrbracket$
 $\Longrightarrow \Theta ; \Phi ; \Delta \vdash \langle \delta , s \rangle \Leftarrow \tau$

equivariance *config-type*
nominal-inductive *config-type* .

inductive-cases *config-type-elim*s [elim!]:

$\Theta ; \Phi ; \Delta \vdash \langle \delta , s \rangle \Leftarrow \tau$

nominal-function *delta-of* :: *var-def list* $\Rightarrow \delta$ **where**

delta-of [] = []
| *delta-of* ((AV-def *u t v*)#*vs*) = (*u,v*) # (*delta-of vs*)
apply *auto*
using *eqvt-def delta-of-graph-aux-def neq-Nil-conv old.prod.exhaust* **apply** *force*
using *eqvt-def delta-of-graph-aux-def neq-Nil-conv old.prod.exhaust*
by (*metis var-def.strong-exhaust*)
nominal-termination (*eqvt*) **by** *lexicographic-order*

inductive *config-type-prog* :: *p* $\Rightarrow \tau \Rightarrow \text{bool} \ (\vdash \langle - \rangle \Leftarrow -)$ **where**

\llbracket
 $\Theta ; \Phi ; \Delta\text{-of } \mathcal{G} \vdash \langle \delta\text{-of } \mathcal{G} , s \rangle \Leftarrow \tau$
 $\rrbracket \Longrightarrow \vdash \langle AP\text{-prog } \Theta \Phi \mathcal{G} s \rangle \Leftarrow \tau$

inductive-cases *config-type-prog-elim*s [elim!]:

$\vdash \langle AP\text{-prog } \Theta \Phi \mathcal{G} s \rangle \Leftarrow \tau$

end
theory *SubstMethods*

imports *IVSubst WellformedL HOL-Eisbach.Eisbach-Tools*

begin

```

method fresh-subst-mth-aux uses add = (
  (match conclusion in atom z # ( $\Gamma::\Gamma$ )[ $x::=v$ ] $_{\Gamma v}$  for z x v  $\Gamma \Rightarrow \langle \text{auto simp add: fresh-subst-gv-if [of } \text{atom } z \text{ } \Gamma \text{ } v \text{ } x \text{ ] add} \rangle$ 
    | (match conclusion in atom z # ( $v'::v$ )[ $x::=v$ ] $_{vv}$  for z x v v'  $\Rightarrow \langle \text{auto simp add: } v.\text{fresh fresh-subst-v-if pure-fresh subst-v-v-def add} \rangle$  )
    | (match conclusion in atom z # ( $ce::ce$ )[ $x::=v$ ] $_{cev}$  for z x v ce  $\Rightarrow \langle \text{auto simp add: fresh-subst-v-if subst-v-ce-def add} \rangle$  )
    | (match conclusion in atom z # ( $\Delta::\Delta$ )[ $x::=v$ ] $_{\Delta v}$  for z x v  $\Delta \Rightarrow \langle \text{auto simp add: fresh-subst-v-if fresh-subst-dv-if add} \rangle$  )
    | (match conclusion in atom z #  $\Gamma'[x::=v]_{\Gamma v}$  @  $\Gamma$  for z x v  $\Gamma' \Gamma \Rightarrow \langle \text{metis add } \rangle$  )
    | (match conclusion in atom z # ( $\tau::\tau$ )[ $x::=v$ ] $_{\tau v}$  for z x v  $\tau \Rightarrow \langle \text{auto simp add: } v.\text{fresh fresh-subst-v-if pure-fresh subst-v-}\tau\text{-def add} \rangle$  )
    | (match conclusion in atom z # ( $\{\|\}$  :: bv fset) for z  $\Rightarrow \langle \text{auto simp add: fresh-empty-fset} \rangle$  )

  | (auto simp add: add x-fresh-b pure-fresh)
)

```

```

method fresh-mth uses add = (
  (unfold fresh-prodN, intro conjI)?,
  (fresh-subst-mth-aux add: add)+)

```

notepad

begin

```

fix  $\Gamma::\Gamma$  and  $z::x$  and  $x::x$  and  $v::v$  and  $\Theta::\Theta$  and  $v'::v$  and  $w::x$  and tyid::string and dc::string
and  $b::b$  and  $ce::ce$  and  $bv::bv$ 

```

```

assume as:atom z # ( $\Gamma, v', \Theta, v, w, ce$ )  $\wedge$  atom bv # ( $\Gamma, v', \Theta, v, w, ce, b$ )

```

```

have atom z #  $\Gamma[x::=v]_{\Gamma v}$ 
by (fresh-mth add: as)

```

```

hence atom z #  $v'[x::=v]_{vv}$ 
by (fresh-mth add: as)

```

```

hence atom z #  $\Gamma$ 
by (fresh-mth add: as)

```



```

hence atom z  $\#$   $\Theta$ 
  by (fresh-mth add: as)

hence atom z  $\#$   $(CE\text{-}val\ v == ce)[x::=v]_{cv}$ 
  using as by auto

hence atom bv  $\#$   $(CE\text{-}val\ v == ce)[x::=v]_{cv}$ 
  using as by auto

have atom z  $\#$   $(\Theta, \Gamma[x::=v]_{\Gamma v, v'}[x::=v]_{vv, w}, V\text{-}pair\ v\ v, V\text{-}consp\ tyid\ dc\ b\ v, (CE\text{-}val\ v == ce)[x::=v]_{cv})$ 
  by (fresh-mth add: as)

have atom bv  $\#$   $(\Theta, \Gamma[x::=v]_{\Gamma v, v'}[x::=v]_{vv, w}, V\text{-}pair\ v\ v, V\text{-}consp\ tyid\ dc\ b\ v)$ 
  by (fresh-mth add: as)

end

end

hide-const Syntax.dom

```

Chapter 11

Refinement Constraint Logic Lemmas

11.1 Lemmas

lemma *wfI-domi*:

assumes $\Theta ; \Gamma \vdash i$

shows $\text{fst } i \text{ toSet } \Gamma \subseteq \text{dom } i$

using *wfI-def toSet.simps assms* **by** *fastforce*

lemma *wfI-lookup*:

fixes $G::\Gamma$ **and** $b::b$

assumes $\text{Some } (b,c) = \text{lookup } G \ x \text{ and } P ; G \vdash i \text{ and } \text{Some } s = i \ x \text{ and } P ; B \vdash_{wf} G$

shows $P \vdash s : b$

proof –

have $(x,b,c) \in \text{toSet } G$ **using** *lookup.simps assms*

using *lookup-in-g* **by** *blast*

then obtain s' **where** $*:\text{Some } s' = i \ x \wedge \text{wfRCV } P \ s' \ b$ **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** $i = \text{restrict-map } i' \ (\text{fst } i \text{ toSet } \Gamma)$ **and** $\text{toSet } \Gamma \subseteq \text{toSet } \Gamma'$

shows $\Theta ; \Gamma \vdash i$

proof –

{ **fix** x

assume $x \in \text{toSet } \Gamma$

have *case* x **of** $(x, b, c) \Rightarrow \exists s. \text{Some } s = i \ x \wedge \Theta \vdash s : b$ **using** *assms wfI-def*

proof –

have *case* x **of** $(x, b, c) \Rightarrow \exists s. \text{Some } s = i' \ x \wedge \Theta \vdash s : b$

using $\langle x \in \text{toSet } \Gamma \rangle$ *assms wfI-def* **by** *auto*

then have $\exists s. \text{Some } s = i \ (\text{fst } x) \wedge \text{wfRCV } \Theta \ s \ (\text{fst } (\text{snd } x))$

by (*simp add: $\langle x \in \text{toSet } \Gamma \rangle$ 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 toSet.simps* **by** *simp*

lemma *wfI-replace-inside*:
assumes $wfI\ \Theta\ (\Gamma' @ (x, b, c) \#_{\Gamma} \Gamma)\ i$
shows $wfI\ \Theta\ (\Gamma' @ (x, b, c') \#_{\Gamma} \Gamma)\ i$
using *wfI-def toSet-splitP assms* **by** *simp*

11.2 Existence of evaluation

lemma *eval-l-base*:
 $\Theta \vdash \llbracket l \rrbracket : (base\text{-}for\text{-}lit\ l)$
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 $(dc, \llbracket x : b \mid c \rrbracket) \in set\ dclist$
obtains $bv1::bv$ **and** $dclist1\ x1\ b1\ c1$ **where** $AF\text{-typedef-poly}\ tyid\ bv\ dclist = AF\text{-typedef-poly}\ tyid\ bv1\ dclist1$
 $\wedge (dc, \llbracket x1 : b1 \mid c1 \rrbracket) \in set\ dclist1 \wedge atom\ bv1 \# tm$
proof –
obtain $bv1\ dclist1$ **where** $AF\text{-typedef-poly}\ tyid\ bv\ dclist = AF\text{-typedef-poly}\ tyid\ bv1\ dclist1 \wedge atom\ bv1 \# tm$
using *obtain-fresh-bv* **by** *metis*
moreover **hence** $\llbracket atom\ bv \rrbracket lst.\ dclist = \llbracket atom\ bv1 \rrbracket lst.\ dclist1$ **using** *type-def.eq-iff* **by** *metis*
moreover **then** **obtain** $x1\ b1\ c1$ **where** $(dc, \llbracket x1 : b1 \mid c1 \rrbracket) \in set\ dclist1$ **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 $(dc, \llbracket x : b \mid c \rrbracket) \in set\ dclist$ **and** $AF\text{-typedef-poly}\ tyid\ bv\ dclist \in set\ P$ **and** $\vdash_{wf} P$
obtains $bv1::bv$ **and** $dclist1\ x1\ b1\ c1$ **where** $AF\text{-typedef-poly}\ tyid\ bv\ dclist = AF\text{-typedef-poly}\ tyid\ bv1\ dclist1$
 $\wedge (dc, \llbracket x1 : b1 \mid c1 \rrbracket) \in set\ dclist1 \wedge atom\ bv1 \# tm \wedge b[bv::=b]_{bb} = b1[bv1::=b]_{bb}$
proof –
obtain $bv1\ dclist1\ x1\ b1\ c1$ **where** $*:AF\text{-typedef-poly}\ tyid\ bv\ dclist = AF\text{-typedef-poly}\ tyid\ bv1\ dclist1$
 $\wedge atom\ bv1 \# tm$
 $\wedge (dc, \llbracket x1 : b1 \mid c1 \rrbracket) \in set\ dclist1$ **using** *obtain-fresh-bv-dclist* *assms* **by** *metis*

hence $AF\text{-typedef-poly}\ tyid\ bv1\ dclist1 \in set\ P$ **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 $\Gamma::\Gamma$ and $v::v$ and $b::b$

assumes $P ; \Gamma \vdash i$ 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-elim rcl-val.supp pure-supp* by *metis*

next

case (*V-var x*)

then obtain *c* where $*:Some (b,c) = lookup \Gamma x$ using *wfV-elim* by *metis*

hence $x \in fst \text{ ' } toSet \Gamma$ 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 $*:P ; B ; \Gamma \vdash_{wf} v1 : b1 \wedge P ; B ; \Gamma \vdash_{wf} v2 : b2 \wedge b = B\text{-pair } b1 \ b2$ using *wfV-elim* by *metis*

then obtain *s1* and *s2* where $eval\text{-}v \ i \ v1 \ s1 \wedge wfRCV \ P \ s1 \ b1 \wedge eval\text{-}v \ i \ v2 \ s2 \wedge wfRCV \ P \ s2 \ b2$ 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 $(wfV \ P \ B \ \Gamma \ v \ b') \wedge i \llbracket v \rrbracket \sim s \wedge P \vdash s : b' \wedge b = B\text{-id } tyid \ \Delta$

$AF\text{-typedef } tyid \ dclist \in set \ P \wedge (dc, \llbracket x : b' \mid c \rrbracket) \in set \ dclist$ using *wfV-elim(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 $*(wfV \ P \ B \ \Gamma \ v \ b'a[bv::=b]_{bb}) \wedge b = B\text{-app } tyid \ b' \ \Delta$

$AF\text{-typedef-poly } tyid \ bv \ dclist \in set \ P \wedge (dc, \llbracket x : b'a \mid c \rrbracket) \in set \ dclist \wedge atom \ bv \ \# (P, B\text{-app } tyid \ b', B)$ 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 $\exists:AF\text{-typedef-poly } tyid \ bv \ dclist = AF\text{-typedef-poly } tyid \ bv1 \ dclist1$

$\wedge (dc, \llbracket x1 : b1 \mid c1 \rrbracket) \in set \ dclist1 \wedge atom \ bv1 \ \# (P, SConsp \ tyid \ dc \ b' \ s, B\text{-app } tyid \ b')$

```

     $\wedge b'a[bv::=b]_{bb} = b1[bv1::=b]_{bb}$ 
    using * obtain-fresh-bv-dclist-b-iff by blast

have i  $\llbracket V\text{-consp } tyid \text{ dc } b' v \rrbracket \sim SConsp \text{ tyid } dc \text{ } b' s$  using eval-v.intros by (simp add: **)

moreover have  $P \vdash SConsp \text{ tyid } dc \text{ } b' s : B\text{-app } tyid \text{ } b'$  proof
  show  $\langle AF\text{-typedef-poly } tyid \text{ } bv1 \text{ } dclist1 \in \text{set } P \rangle$  using 3 * by metis
  show  $\langle (dc, \{ x1 : b1 \mid c1 \}) \in \text{set } dclist1 \rangle$  using 3 by auto
  thus  $\langle atom \text{ } bv1 \nmid (P, SConsp \text{ tyid } dc \text{ } b' s, B\text{-app } tyid \text{ } b') \rangle$  using * 3 fresh-prodN by metis
  show  $\langle P \vdash s : b1[bv1::=b]_{bb} \rangle$  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  $i \llbracket v \rrbracket \sim s$  and  $i \llbracket v \rrbracket \sim s'$ 
  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  $s:i \llbracket v1 \rrbracket \sim s1 \wedge i \llbracket v2 \rrbracket \sim s2 \wedge s = SPair \text{ } s1 \text{ } s2$  using eval-v.elims
  V-pair by metis
  obtain s1' and s2' where  $s':i \llbracket v1 \rrbracket \sim s1' \wedge i \llbracket v2 \rrbracket \sim s2' \wedge s' = SPair \text{ } s1' \text{ } s2'$  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  $1:i \llbracket v1 \rrbracket \sim sv1 \wedge s = SCons \text{ tyid } dc \text{ } sv1$  using eval-v.elims V-cons by metis
  moreover obtain sv2 where  $2:i \llbracket v1 \rrbracket \sim sv2 \wedge s' = SCons \text{ tyid } dc \text{ } sv2$  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  $\Gamma::\Gamma$  and  $v::v$  and  $b::b$ 
  assumes  $P ; \Gamma \vdash i$  and  $P ; B ; \Gamma \vdash_{wf} v : b$  and  $i \llbracket v \rrbracket \sim s$ 
  shows  $P \vdash s : b$ 
  using eval-v-exist eval-v-uniqueness assms by metis

```

```

lemma eval-e-uniqueness:
  fixes e::ce
  assumes i [ e ] ~ s and i [ e ] ~ 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-elim by metis
next
  case (CE-op opp x1 x2)
  consider opp = Plus | 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-elim(2)[OF a1] eval-e-elim(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-elim(2)
    then show ?thesis using eval-v-uniqueness eval-e-elim(3)[OF a1] eval-e-elim(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-elim(6)[OF a1] eval-e-elim(6)[OF a2] CE-concat eval-e-concatI rcl-val.eq-iff

proof -
  assume  $\bigwedge P. (\bigwedge bv1\ bv2. [s' = SBitvec\ (bv1\ @\ bv2); i\ [x1]\ \sim\ SBitvec\ bv1\ ;\ i\ [x2]\ \sim\ SBitvec\ bv2] \implies P) \implies P$ 
  obtain bbs :: bit list and bbsa :: bit list where
    i [ x2 ] ~ SBitvec bbs  $\wedge$  i [ x1 ] ~ SBitvec bbsa  $\wedge$  SBitvec (bbsa @ bbs) = s'
  by (metis ( $\bigwedge P. (\bigwedge bv1\ bv2. [s' = SBitvec\ (bv1\ @\ bv2); i\ [x1]\ \sim\ SBitvec\ bv1\ ;\ i\ [x2]\ \sim\ SBitvec\ bv2] \implies P) \implies P$ )
    bv2 ]  $\implies P$ )  $\implies P$ )
  then have s' = s
  by (metis (no-types) ( $\bigwedge P. (\bigwedge bv1\ bv2. [s = SBitvec\ (bv1\ @\ bv2); i\ [x1]\ \sim\ SBitvec\ bv1\ ;\ i\ [x2]\ \sim\ SBitvec\ bv2] \implies P) \implies P$ )
    ( $\bigwedge s'\ s. [i\ [x1]\ \sim\ s\ ;\ i\ [x1]\ \sim\ s'] \implies s = s'$ )
    ( $\bigwedge s'\ s. [i\ [x2]\ \sim\ s\ ;\ i\ [x2]\ \sim\ s'] \implies s = s'$ )
    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-elim rcl-val.eq-iff)
next
  case (CE-snd x)
  then show ?case using eval-v-uniqueness by (meson eval-e-elim rcl-val.eq-iff)

```

next

case (CE-len x)
 then show ?case using eval-e-elim rcl-val.eq-iff
 proof –
 obtain bbs :: rcl-val \Rightarrow ce \Rightarrow (x \Rightarrow rcl-val option) \Rightarrow bit list where
 $\forall x0\ x1\ x2. (\exists v3. x0 = SNum\ (int\ (length\ v3)) \wedge x2\ \llbracket\ x1\ \rrbracket \sim SBitvec\ v3) = (x0 = SNum\ (int\ (length\ (bbs\ x0\ x1\ x2)))) \wedge x2\ \llbracket\ x1\ \rrbracket \sim SBitvec\ (bbs\ x0\ x1\ x2))$
 by moura
 then have $\forall f\ c\ r. \neg f\ \llbracket\ \llbracket\ c\ \rrbracket^{ce}\ \rrbracket \sim r \vee r = SNum\ (int\ (length\ (bbs\ r\ c\ f))) \wedge f\ \llbracket\ c\ \rrbracket \sim SBitvec\ (bbs\ r\ c\ f)$
 by (meson eval-e-elim(γ))
 then show ?thesis
 by (metis (no-types) CE-len.hyps CE-len.prem1 CE-len.prem2 rcl-val.eq-iff(1))
 qed

qed

lemma wfV-eval-bitvec:

fixes v::v
 assumes P ; B ; $\Gamma \vdash_{wf} v : B\text{-bitvec}$ and P ; $\Gamma \vdash i$
 shows $\exists bv. eval\text{-}v\ i\ v\ (SBitvec\ bv)$
 proof –
 obtain s where $i\ \llbracket\ v\ \rrbracket \sim s \wedge wfRCV\ P\ s\ B\text{-bitvec}$ using eval-v-exist assms by metis
 moreover then obtain bv where $s = SBitvec\ bv$ using wfRCV-elim(1)[of P s] by metis
 ultimately show ?thesis by metis
 qed

lemma wfV-eval-pair:

fixes v::v
 assumes P ; B ; $\Gamma \vdash_{wf} v : B\text{-pair}\ b1\ b2$ and P ; $\Gamma \vdash i$
 shows $\exists s1\ s2. eval\text{-}v\ i\ v\ (SPair\ s1\ s2)$
 proof –
 obtain s where $i\ \llbracket\ v\ \rrbracket \sim s \wedge wfRCV\ P\ s\ (B\text{-pair}\ b1\ b2)$ using eval-v-exist assms by metis
 moreover then obtain s1 and s2 where $s = SPair\ s1\ s2$ using wfRCV-elim(2)[of P s] by metis
 ultimately show ?thesis by metis
 qed

lemma wfV-eval-int:

fixes v::v
 assumes P ; B ; $\Gamma \vdash_{wf} v : B\text{-int}$ and P ; $\Gamma \vdash i$
 shows $\exists n. eval\text{-}v\ i\ v\ (SNum\ n)$
 proof –
 obtain s where $i\ \llbracket\ v\ \rrbracket \sim s \wedge wfRCV\ P\ s\ (B\text{-int})$ using eval-v-exist assms by metis
 moreover then obtain n where $s = SNum\ n$ using wfRCV-elim(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 $\Theta ; B ; \Gamma \vdash_{wf} v : b$ and wfI $\Theta\ \Gamma\ i$

```

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 ⊢ 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-elim1
  [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-elim
  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-elim 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.prem1 wfCE-elim(2) by blast
  next
    case 2
    hence eval-e i (CE-op LEq v1 v2) (SBool (n1 ≤ n2)) using eval-e-leqI * ** by simp
    moreover have wfRCV P (SBool (n1 ≤ n2)) B-bool using wfRCV.intros by auto
    ultimately show ?thesis using 2
    using CE-op.prem1 wfCE-elim 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-elim(6))
  thus ?case using eval-e-concatI wfRCV.intros(1) wfRCV-elim
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-elim(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-elim wfI-wfV-eval-v

```



```

    by (metis wfRCV-elim(2) rcl-val.eq-iff(4))
next
case (CE-snd v1)
thus ?case using eval-e-sndI wfRCV.intros wfCE-elim wfI-wfV-eval-v
    by (metis wfRCV-elim(2) rcl-val.eq-iff(4))
next
case (CE-len x)
thus ?case using eval-e-lenI wfRCV.intros wfCE-elim wfI-wfV-eval-v wfV-eval-bitvec
    by (metis wfRCV-elim(1))
qed

lemma eval-e-exist:
  fixes  $\Gamma::\Gamma$  and  $e::ce$ 
  assumes  $P ; \Gamma \vdash i$  and  $P ; B ; \Gamma \vdash_{wf} e : b$ 
  shows  $\exists s. i \llbracket e \rrbracket \sim s$ 
using assms proof(nominal-induct e arbitrary: b rule:ce.strong-induct)
  case (CE-val v)
  then show ?case using eval-v-exist wfCE-elim eval-e.intros by metis
next
  case (CE-op op v1 v2)

  show ?case proof(rule opp.exhaust)
    assume  $\langle op = Plus \rangle$ 
    hence  $P ; B ; \Gamma \vdash_{wf} v1 : B-int \wedge P ; B ; \Gamma \vdash_{wf} v2 : B-int \wedge b = B-int$  using wfCE-elim CE-op
  by metis
    then obtain n1 and n2 where eval-e i v1 (SNum n1)  $\wedge$  eval-e i v2 (SNum n2) using CE-op
  eval-v-exist wfV-eval-int
    by (metis wfI-wfCE-eval-e wfRCV-elim(3))
    then show  $\langle \exists a. eval-e i (CE-op op v1 v2) a \rangle$  using eval-e-plusI[of i v1 - v2]  $\langle op=Plus \rangle$  by auto
  next
    assume  $\langle op = LEq \rangle$ 
    hence  $P ; B ; \Gamma \vdash_{wf} v1 : B-int \wedge P ; B ; \Gamma \vdash_{wf} v2 : B-int \wedge b = B-bool$  using wfCE-elim
  CE-op by metis
    then obtain n1 and n2 where eval-e i v1 (SNum n1)  $\wedge$  eval-e i v2 (SNum n2) using CE-op
  eval-v-exist wfV-eval-int
    by (metis wfI-wfCE-eval-e wfRCV-elim(3))
    then show  $\langle \exists a. eval-e i (CE-op op v1 v2) a \rangle$  using eval-e-leqI[of i v1 - v2] eval-v-exist  $\langle op=LEq \rangle$ 
  CE-op by auto
  qed
next
  case (CE-concat v1 v2)
  then obtain bv1 and bv2 where eval-e i v1 (SBitvec bv1)  $\wedge$  eval-e i v2 (SBitvec bv2)
    using wfV-eval-bitvec wfCE-elim(6)
    by (meson eval-e-elim(6) wfI-wfCE-eval-e)
  then show ?case using eval-e.intros by metis
next
  case (CE-fst ce)
  then obtain b2 where  $b:P ; B ; \Gamma \vdash_{wf} ce : B-pair b b2$  using wfCE-elim by metis
  then obtain s where  $s:i \llbracket ce \rrbracket \sim s$  using CE-fst by auto
  then obtain s1 and s2 where  $s = (SPair s1 s2)$  using eval-e-elim(4) CE-fst wfI-wfCE-eval-e[of
  P B  $\Gamma$  ce B-pair b b2 i,OF b] wfRCV-elim(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 ;  $\Gamma \vdash_{wf} ce : B\text{-pair } b1\ b$  using wfCE-elim by metis
then obtain s where s:i [ ce ]  $\sim$  s using CE-snd by auto
then obtain s1 and s2 where s = (SPair s1 s2)
  using eval-e-elim(5) CE-snd wfI-wfCE-eval-e[of P B  $\Gamma$  ce B-pair b1 b i, OF b] wfRCV-elim(2)[of
P s b b1]
  eval-e-uniqueness
  by (metis wfRCV-elim(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-elim eval-e-uniqueness
  by (metis eval-e-elim(6) wfCE-concatI wfI-wfCE-eval-e)
then show ?case using eval-e.intros by metis
qed

```

```

lemma eval-c-exist:
  fixes  $\Gamma::\Gamma$  and c::c
  assumes P ;  $\Gamma \vdash i$  and P ; B ;  $\Gamma \vdash_{wf} c$ 
  shows  $\exists s. i [ c ] \sim s$ 
using assms proof(nominal-induct c rule: c.strong-induct)
case C-true
  then show ?case using eval-c.intros wfC-elim by metis
next
case C-false
  then show ?case using eval-c.intros wfC-elim by metis
next
case (C-conj c1 c2)
  then show ?case using eval-c.intros wfC-elim by metis
next
case (C-disj x1 x2)
  then show ?case using eval-c.intros wfC-elim by metis
next
case (C-not x)
  then show ?case using eval-c.intros wfC-elim by metis
next
case (C-imp x1 x2)
  then show ?case using eval-c.intros eval-e-exist wfC-elim by metis
next
case (C-eq x1 x2)
  then show ?case using eval-c.intros eval-e-exist wfC-elim by metis
qed

```

```

lemma eval-c-uniqueness:
  fixes c::c
  assumes i [ c ]  $\sim$  s and i [ c ]  $\sim$  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-elim 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) ∈ toSet 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-elim1 eval-v-litI)
next
  case (V-var y)
  then obtain s where *: Some s = i y ∧ s=s' using eval-v-elim1 by metis
  moreover have x ≠ y using ⟨atom x # V-var y⟩ v.sup 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.sup 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-elim1 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.sup by simp
  moreover obtain s1 where *: eval-v i v1 s1 ∧ s' = SCons tyid dc s1 using eval-v-elim1 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-elim V-consp
by metis
ultimately have eval-v ((i ( x ↦ s))) v1 s1 using V-consp by blast
thus ?case using eval-v-consI * 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-elim
  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-elim(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-elim 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-elim subst-vv.simps by metis

```

hence $(i (x \mapsto s)) \llbracket v1 \rrbracket \sim s1 \wedge (i (x \mapsto s)) \llbracket v2 \rrbracket \sim s2$ **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-elim subst-vv.simps* **by** *metis*
 thus $?case$ **using** *eval-v-consI V-cons*
 by (*metis eval-v-elim 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-elim subst-vv.simps* **by** *metis*
 hence $i (x \mapsto s) \llbracket v1 \rrbracket \sim s1$ **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 $i \llbracket e' \rrbracket \sim s'$ and $e' = (e[y::=v]_{cev})$ and $i \llbracket v \rrbracket \sim s$
 shows $(i (y \mapsto s)) \llbracket e \rrbracket \sim s'$
 using *assms* **proof**(*induct arbitrary: e rule: eval-e.induct*)
 case (*eval-e-valI i v1 sv*)
 then obtain $v1'$ **where** $*:e = CE-val\ v1' \wedge v1 = v1'[y::=v]_{vv}$
 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** $*:e = CE-op\ Plus\ v1'\ v2' \wedge v1 = v1'[y::=v]_{cev} \wedge v2 = v2'[y::=v]_{cev}$
 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** $*:e = CE-op\ LEq\ v1'\ v2' \wedge v1 = v1'[y::=v]_{cev} \wedge v2 = v2'[y::=v]_{cev}$
 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** $*:e = CE-fst\ v1' \wedge v1 = v1'[y::=v]_{cev}$
 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 $\text{eval-e } (i \ (y \mapsto s)) \ v1' \ (SPair \ s1 \ s2)$ using $\text{eval-e-fstI} \ * \ \text{by} \ \text{metis}$
 then show $?case$ using $RCLogic.\text{eval-e-fstI} \ * \ \text{by} \ \text{meson}$
 next
 case $(\text{eval-e-sndI } i \ v1 \ s1 \ s2)$
 then obtain $v1'$ and $v2'$ where $*:e = CE\text{-snd } v1' \wedge v1 = v1' [y::=v]_{cev}$
 using $\text{assms} \ \text{by} \ (\text{nominal-induct } e \ \text{rule:ce.strong-induct, simp+})$
 hence $\text{eval-e } i \ (v1' [y::=v]_{cev}) \ (SPair \ s1 \ s2)$ using $\text{eval-e-sndI} \ \text{by} \ \text{simp}$
 hence $\text{eval-e } (i \ (y \mapsto s)) \ v1' \ (SPair \ s1 \ s2)$ using $\text{subst-v-eval-v } \text{eval-e-sndI} \ * \ \text{by} \ \text{blast}$
 then show $?case$ using $RCLogic.\text{eval-e-sndI} \ * \ \text{by} \ \text{meson}$
 next
 case $(\text{eval-e-concatI } i \ v1 \ bv1 \ v2 \ bv2)$
 then obtain $v1'$ and $v2'$ where $*:e = CE\text{-concat } v1' \ v2' \wedge v1 = v1' [y::=v]_{cev} \wedge v2 = v2' [y::=v]_{cev}$
 using $\text{assms} \ \text{by} \ (\text{nominal-induct } e \ \text{rule:ce.strong-induct, simp+})$
 hence $\text{eval-e } i \ (v1' [y::=v]_{cev}) \ (SBitvec \ bv1) \wedge \text{eval-e } i \ (v2' [y::=v]_{cev}) \ (SBitvec \ bv2)$ using eval-e-concatI
 by simp
 moreover hence $\text{eval-e } (i \ (y \mapsto s)) \ v1' \ (SBitvec \ bv1) \wedge \text{eval-e } (i \ (y \mapsto s)) \ v2' \ (SBitvec \ bv2)$
 using $\text{subst-v-eval-v } \text{eval-e-concatI} \ * \ \text{by} \ \text{blast}$
 ultimately show $?case$ using $RCLogic.\text{eval-e-concatI} \ * \ \text{eval-v-uniqueness} \ \text{by} \ (\text{metis } \text{eval-e-concatI}.\text{hyps}(1))$
 next
 case $(\text{eval-e-lenI } i \ v1 \ bv)$
 then obtain $v1'$ where $*:e = CE\text{-len } v1' \wedge v1 = v1' [y::=v]_{cev}$
 using $\text{assms} \ \text{by} \ (\text{nominal-induct } e \ \text{rule:ce.strong-induct, simp+})$
 hence $\text{eval-e } i \ (v1' [y::=v]_{cev}) \ (SBitvec \ bv)$ using $\text{eval-e-lenI} \ \text{by} \ \text{simp}$
 hence $\text{eval-e } (i \ (y \mapsto s)) \ v1' \ (SBitvec \ bv)$ using $\text{subst-v-eval-v } \text{eval-e-lenI} \ * \ \text{by} \ \text{blast}$
 then show $?case$ using $RCLogic.\text{eval-e-lenI} \ * \ \text{by} \ \text{meson}$
 qed

lemma $\text{subst-c-eval-v}[\text{simp}]$:
 fixes $v::v$ and $c::c$
 assumes $i \llbracket v \rrbracket \sim s$ and $i \llbracket c[x::=v]_{cv} \rrbracket \sim s1$ and
 $(i \ (x \mapsto s)) \llbracket c \rrbracket \sim s2$
 shows $s1 = s2$
 using $\text{assms} \ \text{proof} \ (\text{nominal-induct } c \ \text{arbitrary: } s1 \ s2 \ \text{rule: } c.\text{strong-induct})$
 case $C\text{-true}$
 hence $s1 = \text{True} \wedge s2 = \text{True}$ using $\text{eval-c-elim} \ \text{subst-cv.simps} \ \text{by} \ \text{auto}$
 then show $?case$ by auto
 next
 case $C\text{-false}$
 hence $s1 = \text{False} \wedge s2 = \text{False}$ using $\text{eval-c-elim} \ \text{subst-cv.simps} \ \text{by} \ \text{metis}$
 then show $?case$ by auto
 next
 case $(C\text{-conj } c1 \ c2)$
 hence $*:\text{eval-c } i \ (c1[x::=v]_{cv} \ \text{AND} \ c2[x::=v]_{cv}) \ s1$ using $\text{subst-cv.simps} \ \text{by} \ \text{auto}$
 then obtain $s11$ and $s12$ where $(s1 = (s11 \wedge s12)) \wedge \text{eval-c } i \ c1[x::=v]_{cv} \ s11 \wedge \text{eval-c } i \ c2[x::=v]_{cv} \ s12$ using
 $\text{eval-c-elim}(3) \ \text{by} \ \text{metis}$
 moreover obtain $s21$ and $s22$ where $\text{eval-c } (i \ (x \mapsto s)) \ c1 \ s21 \wedge \text{eval-c } (i \ (x \mapsto s)) \ c2 \ s22 \wedge$
 $(s2 = (s21 \wedge s22))$ using
 $\text{eval-c-elim}(3) \ C\text{-conj} \ \text{by} \ \text{metis}$
 ultimately show $?case$ using $C\text{-conj} \ \text{by} \ (\text{meson } \text{eval-c-elim})$
 next
 case $(C\text{-disj } c1 \ c2)$

```

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) #Γ Γ)
shows wfI Θ ((x, b, c) #Γ Γ) (i(x ↦ s))
proof(subst wfI-def,rule)
  fix xa
  assume as:xa ∈ toSet ((x, b, c) #Γ Γ)

  then obtain x1::x and b1::b and c1::c where xa: xa = (x1,b1,c1) using toSet.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 $(x1, b1, c1) \in toSet \Gamma$ using *xa as* by auto
 then obtain *sa* where *Some sa* = $i \ x1 \wedge wfRCV \ \Theta \ sa \ b1$ using *assms wfI-def as xa* by auto
 hence *Some sa* = $(i(x \mapsto s)) \ x1 \wedge wfRCV \ \Theta \ sa \ b1$ using *False* by auto
 then show ?thesis by auto
 qed

thus case *xa* of $(xa, ba, ca) \Rightarrow \exists sa. \text{Some } sa = (i(x \mapsto s)) \ x1 \wedge wfRCV \ \Theta \ sa \ ba$ using *xa* by auto
 qed

lemma *wfI-upd-full*:
 fixes $v::v$
 assumes $wfI \ \Theta \ G \ i$ and $G = ((\Gamma'[x::=v]_{\Gamma v}) @ \Gamma)$ and $wfRCV \ \Theta \ s \ b$ and $wfG \ \Theta \ B \ (\Gamma' @ ((x, b, c) \#_{\Gamma} \Gamma))$
 and $\Theta ; B ; \Gamma \vdash_{wf} v : b$
 shows $wfI \ \Theta \ (\Gamma' @ ((x, b, c) \#_{\Gamma} \Gamma)) \ (i(x \mapsto s))$
proof(*subst wfI-def, rule*)
 fix *xa*
 assume *as:xa* $\in toSet (\Gamma' @ ((x, b, c) \#_{\Gamma} \Gamma))$

then obtain $x1::x$ and $b1::b$ and $c1::c$ where *xa*: $xa = (x1, b1, c1)$ using *toSet.simps*
 using *prod-cases3* by blast

have $\exists sa. \text{Some } sa = (i(x \mapsto s)) \ x1 \wedge wfRCV \ \Theta \ sa \ b1$
proof(*cases x=x1*)
 case True
 hence $b=b1$ using *as xa wfG-unique-full assms* by metis
 hence *Some s* = $(i(x \mapsto s)) \ x1 \wedge wfRCV \ \Theta \ s \ b1$ using *assms True* by simp
 then show ?thesis by auto
 next
 case False
 hence $(x1, b1, c1) \in toSet (\Gamma' @ \Gamma)$ using *as xa* by auto
 then obtain $c1'$ where $(x1, b1, c1') \in toSet (\Gamma'[x::=v]_{\Gamma v} @ \Gamma)$ using *xa as wfG-member-subst assms*
False by metis
 then obtain *sa* where *Some sa* = $i \ x1 \wedge wfRCV \ \Theta \ sa \ b1$ using *assms wfI-def as xa* by blast
 hence *Some sa* = $(i(x \mapsto s)) \ x1 \wedge wfRCV \ \Theta \ sa \ b1$ using *False* by auto
 then show ?thesis by auto
 qed

thus case *xa* of $(xa, ba, ca) \Rightarrow \exists sa. \text{Some } sa = (i(x \mapsto s)) \ x1 \wedge wfRCV \ \Theta \ sa \ ba$ using *xa* by auto
 qed

lemma *subst-c-satis[simp]*:
 fixes $v::v$
 assumes $i \llbracket v \rrbracket \sim s$ and $wfC \ \Theta \ B \ ((x, b, c') \#_{\Gamma} \Gamma) \ c$ and $wfI \ \Theta \ \Gamma \ i$ and $\Theta ; B ; \Gamma \vdash_{wf} v : b$
 shows $i \models (c[x::=v]_{cv}) \longleftrightarrow (i \ (x \mapsto s)) \models c$
proof –
 have $wfI \ \Theta \ ((x, b, c') \#_{\Gamma} \Gamma) \ (i(x \mapsto s))$ using *wfI-upd assms wfC-wf eval-v-base* by blast
 then obtain $s1$ where $s1:eval-c \ (i(x \mapsto s)) \ c \ s1$ using *eval-c-exist[of \Theta ((x, b, c') \#_{\Gamma} \Gamma) (i \ (x \mapsto s)) B c]* assms by auto

```

  have  $\Theta ; B ; \Gamma \vdash_{wf} c[x::=v]_{cv}$  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  $\Theta\ \Gamma\ i\ B\ c[x::=v]_{cv}$  ] 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

```

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  $i \llbracket v \rrbracket \sim s$  and  $wfC\ \Theta\ B\ (\Gamma'@((x,b,c')\#_{\Gamma}\Gamma))\ c$  and  $wfI\ \Theta\ ((\Gamma'[x::=v]_{\Gamma v})@_{\Gamma})\ i$  and  $\Theta$ 
  ;  $B ; \Gamma \vdash_{wf} v : b$ 
  shows  $i \models (c[x::=v]_{cv}) \longleftrightarrow (i (x \mapsto s)) \models c$ 
proof -
  have  $wfI\ \Theta\ (\Gamma'@((x, b, c')\#_{\Gamma}\Gamma))\ (i(x \mapsto s))$  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  $\Theta\ (\Gamma'@((x,b,c')\#_{\Gamma}\Gamma))\ (i(x \mapsto s))\ B\ c$  ] assms by auto

```

```

  have  $\Theta ; B ; ((\Gamma'[x::=v]_{\Gamma v})@_{\Gamma}) \vdash_{wf} c[x::=v]_{cv}$  using wbc-subst assms by auto

  then obtain s2 where  $s2:eval-c\ i\ c[x::=v]_{cv}\ s2$  using eval-c-exist[of  $\Theta\ ((\Gamma'[x::=v]_{\Gamma v})@_{\Gamma})\ i\ B\ c[x::=v]_{cv}$  ] 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  $G::\Gamma$ 
  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  $i = i' \mid 'd$  and  $\text{supp } c \subseteq \text{atom } 'd \cup \text{supp } B$  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  $\text{atom } x \in \text{atom } 'd$  using x-not-in-b-set[of x B] assms v.sup(2) supp-at-base
  proof –
    show ?thesis
    by (metis UnE V-var.prems(2) ( $\text{atom } x \notin \text{supp } B$ ) singletonI subset-iff supp-at-base v.sup(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) ( $i = i' \mid 'd$ ) (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.sup
  by (metis assms le-sup-iff)
next
  case (V-cons dc v1)
  then show ?case using eval-v-elims(4) eval-v-consI v.sup
  by (metis assms le-sup-iff)
next
  case (V-consp tyid dc b1 v1)

  then obtain sv1 where  $*:i \llbracket v1 \rrbracket \sim sv1 \wedge s = S\text{Consp } tyid \ dc \ b1 \ sv1$  using eval-v-elims by metis
  hence  $i' \llbracket v1 \rrbracket \sim sv1$  using V-consp by auto
  then show ?case using  $* \text{ eval-v-conspI } v.\text{supp } \text{eval-v.simp}s \text{ assms le-sup-iff}$  by metis
qed

```

```

lemma eval-v-restrict:
  fixes  $c::v$  and  $B::bv$  fset
  assumes  $i = i' \mid 'd$  and  $\text{supp } c \subseteq \text{atom } 'd \cup \text{supp } B$  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  $x \in \text{atom } 'd$  using  $x\text{-not-in-b-set}[of\ x\ B]$  assms  $v.\text{supp}(2)$   $\text{supp-at-base}$ 
proof -
  show ?thesis
    by (metis  $UnE$   $V\text{-var.prem}(2)$   $\langle \text{atom } x \notin \text{supp } B \rangle$   $\text{singletonI}$   $\text{subset-iff}$   $\text{supp-at-base}$   $v.\text{supp}(2)$ )
qed
moreover have  $\text{Some } s = i' x$  using assms  $\text{eval-v-elim}(2)$ 
  using  $V\text{-var.prem}(3)$  by blast
hence  $\text{Some } s = i x$  using assms  $\text{insert-subset}$   $\text{restrict-in}$ 
proof -
  show ?thesis
    by (metis (no-types)  $\langle i = i' \mid 'd \rangle$   $\langle \text{Some } s = i' x \rangle$   $\text{atom-eq-iff}$   $\text{calculation}$   $\text{imageE}$   $\text{restrict-in}$ )
qed
thus ?case using  $\text{eval-v.eval-v-varI}$  by simp
next
case ( $V\text{-pair } v1\ v2$ )
then show ?case using  $\text{eval-v-elim}(3)$   $\text{eval-v-pairI}$   $v.\text{supp}$ 
  by (metis assms  $\text{le-sup-iff}$ )
next
case ( $V\text{-cons } dc\ v1$ )
then show ?case using  $\text{eval-v-elim}(4)$   $\text{eval-v-consI}$   $v.\text{supp}$ 
  by (metis assms  $\text{le-sup-iff}$ )
next
case ( $V\text{-consp } tyid\ dc\ b1\ v1$ )

then obtain  $sv1$  where  $*:i' \llbracket v1 \rrbracket \sim sv1 \wedge s = SConsp\ tyid\ dc\ b1\ sv1$  using  $\text{eval-v-elim}$  by metis
hence  $i \llbracket v1 \rrbracket \sim sv1$  using  $V\text{-consp}$  by auto
then show ?case using  $*$   $\text{eval-v-conspI}$   $v.\text{supp}$   $\text{eval-v.simps}$  assms  $\text{le-sup-iff}$  by metis
qed

lemma  $\text{eval-e-weakening}$ :
  fixes  $e::ce$  and  $B::bv\ fset$ 
  assumes  $i \llbracket e \rrbracket \sim s$  and  $i = i' \mid 'd$  and  $\text{supp } e \subseteq \text{atom } 'd \cup \text{supp } B$ 
  shows  $i' \llbracket e \rrbracket \sim s$ 
using assms proof(induct rule:  $\text{eval-e.induct}$ )
  case ( $\text{eval-e-valI } i\ v\ sv$ )
  then show ?case using  $ce.\text{supp}$   $\text{eval-e.intros}$ 
    using  $\text{eval-v-weakening}$  by auto
next
  case ( $\text{eval-e-plusI } i\ v1\ n1\ v2\ n2$ )
  then show ?case using  $ce.\text{supp}$   $\text{eval-e.intros}$ 
    using  $\text{eval-v-weakening}$  by auto
next
  case ( $\text{eval-e-leqI } i\ v1\ n1\ v2\ n2$ )
  then show ?case using  $ce.\text{supp}$   $\text{eval-e.intros}$ 
    using  $\text{eval-v-weakening}$  by auto
next
  case ( $\text{eval-e-fstI } i\ v\ v1\ v2$ )
  then show ?case using  $ce.\text{supp}$   $\text{eval-e.intros}$ 
    using  $\text{eval-v-weakening}$  by metis
next
  case ( $\text{eval-e-sndI } i\ v\ v1\ v2$ )
  then show ?case using  $ce.\text{supp}$   $\text{eval-e.intros}$ 

```

```

    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'  $\llbracket e \rrbracket \sim s$  and  $i = i' \mid d$  and  $\text{supp } e \subseteq \text{atom } d \cup \text{supp } B$ 
  shows  $i \llbracket e \rrbracket \sim s$ 
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  $\llbracket c \rrbracket \sim s$  and  $i = i' \mid d$  and  $\text{supp } c \subseteq \text{atom } d \cup \text{supp } B$ 
  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-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 $\text{supp } c \subseteq \text{atom } ' d \cup \text{supp } B$
 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 $\text{supp } c \subseteq \text{atom } ' d \cup \text{supp } B$ 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-i-restrict:
 fixes $c::c$ and $B::bv$ fset
 assumes $i = i' \mid ' d$ and $\text{supp } c \subseteq \text{atom } ' d \cup \text{supp } B$ 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 $\Gamma'::\Gamma$ and $\Gamma::\Gamma$
 assumes $\text{toSet } \Gamma \subseteq \text{toSet } \Gamma'$ 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 toSet.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 $\Gamma'::\Gamma$ and $\Gamma::\Gamma$
 assumes $i \models \Gamma$ and $i' = i \mid ' d$ and $\text{atom-dom } \Gamma \subseteq \text{atom } ' d$ 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 $atom\text{-}dom\ G \subseteq atom\ 'd$ **using** $GCons$ **by** *simp*

ultimately show *?thesis* **using** $GCons\ wfG\text{-}cons2$ **by** *blast*

qed

moreover have $i' \models c$ **proof** –

have $i \models c$ **using** $GCons$ **by** *auto*

moreover have $\Theta ; B ; (x, b, TRUE) \#_{\Gamma} G \vdash_{wf} c$ **using** $wfG\text{-}wfC\ GCons$ **by** *simp*

moreover hence $supp\ c \subseteq atom\ 'd \cup supp\ B$ **using** $wfC\text{-}supp\ GCons\ atom\text{-}dom\text{-}eq$ **by** *blast*

ultimately show *?thesis* **using** $is\text{-}satis\text{-}i\text{-}restrict[of\ i'\ i\ d\ c]\ GCons$ **by** *simp*

qed

ultimately show *?case* **by** *auto*

qed

lemma *is-satis-g-restrict*:

fixes $\Gamma'::\Gamma$ and $\Gamma::\Gamma$

assumes $toSet\ \Gamma \subseteq toSet\ \Gamma'$ and $i' \models \Gamma'$ and $i = i' \mid' (fst\ 'toSet\ \Gamma)$ and $\Theta ; B \vdash_{wf} \Gamma$

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 $G::\Gamma$

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 $*:atom\ x \# G' \wedge atom\ x \# c'$

using *fresh-def fresh-GCons GCons* **by** *force*

moreover hence $is\text{-}satis\ ((i\ (x \mapsto s)))\ c'$

using *is-satis-c-i-upd GCons is-satis-g.simps* **by** *auto*

moreover have $is\text{-}satis\text{-}g\ (i(x \mapsto s))\ G'$ **using** $GCons\ *$ **by** *fastforce*

ultimately show *?case*

using $GCons\ is\text{-}satis\text{-}g.simps(2)$ **by** *metis*

qed

lemma *valid-weakening*:

assumes $\Theta ; B ; \Gamma \models c$ and $toSet\ \Gamma \subseteq toSet\ \Gamma'$ and $wfG\ \Theta\ B\ \Gamma'$

shows $\Theta ; B ; \Gamma' \models c$
proof –
have $wfC \ \Theta \ B \ \Gamma \ c$ **using** *assms valid.simps* **by** *auto*
hence $sp: \text{supp } c \subseteq \text{atom } (fst \ 'toSet \ \Gamma) \cup \text{supp } B$ **using** *wfX-wfY wfG-elim*
using *atom-dom.simps dom.simps wf-supp(2)* **by** *metis*
have $wfg: wfG \ \Theta \ B \ \Gamma$ **using** *assms valid.simps wfC-wf* **by** *auto*

moreover have $a1: (\forall i. wfI \ \Theta \ \Gamma' \ i \wedge i \models \Gamma' \longrightarrow i \models c)$ **proof**(*rule allI, rule impI*)
fix i
assume $as: wfI \ \Theta \ \Gamma' \ i \wedge i \models \Gamma'$
hence $as1: fst \ 'toSet \ \Gamma \subseteq dom \ i$ **using** *assms wfI-domi* **by** *blast*
obtain i' **where** $idash: i' = restrict\text{-}map \ i \ (fst \ 'toSet \ \Gamma)$ **by** *blast*
hence $as2: dom \ i' = (fst \ 'toSet \ \Gamma)$ **using** *dom-restrict as1* **by** *auto*

have $id2: \Theta ; \Gamma \vdash i' \wedge i' \models \Gamma$ **proof** –
have $wfI \ \Theta \ \Gamma \ i'$ **using** $as2$ *wfI-restrict-weakening[of $\Theta \ \Gamma' \ i \ i' \ \Gamma$]* *as* *assms*
using $idash$ **by** *blast*
moreover have $i' \models \Gamma$ **using** *is-satis-g-restrict[OF assms(2)] wfg as idash* **by** *auto*
ultimately show *?thesis* **using** $idash$ **by** *auto*
qed
hence $i' \models c$ **using** *assms valid.simps* **by** *auto*
thus $i \models c$ **using** *assms valid.simps is-satis-i-weakening idash sp* **by** *blast*
qed
moreover have $wfC \ \Theta \ B \ \Gamma' \ c$ **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 $G::\Gamma$
assumes $i \models (G'@G)$
shows $i \models G$
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 $i \models (xbc \#_{\Gamma} (x2 \ @ \ G))$ **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** $\Gamma::\Gamma$ **and** $c0::c$ **and** $c0'::c$
assumes $wfG \ \Theta \ B \ (\Gamma'@((x,b0,c0')\#_{\Gamma}\Gamma))$ **and**
 $\Theta ; B ; \Gamma'@((x,b0,c0)\#_{\Gamma}\Gamma) \models c0'$
shows $wfG \ \Theta \ B \ (\Gamma'@((x,b0,c0)\#_{\Gamma}\Gamma))$
proof –
have $wfG \ \Theta \ B \ (\Gamma'@((x,b0,c0)\#_{\Gamma}\Gamma))$ **using** *valid.simps wfC-wf assms* **by** *auto*
thus *?thesis* **using** *wfG-replace-inside-full assms* **by** *auto*
qed

lemma *valid-trans*:

assumes $\Theta ; \mathcal{B} ; \Gamma \models c0[z ::= v]_v$ **and** $\Theta ; \mathcal{B} ; (z, b, c0) \#_{\Gamma} \Gamma \models c1$ **and** $\text{atom } z \# \Gamma$ **and** $\text{wfV } \Theta \mathcal{B}$
 $\Gamma \vdash v \ b$

shows $\Theta ; \mathcal{B} ; \Gamma \models c1[z ::= v]_v$

proof –

have $\ast : \text{wfC } \Theta \mathcal{B} ((z, b, c0) \#_{\Gamma} \Gamma) \ c1$ **using** *valid.simps* **assms** **by** *auto*

hence $\text{wfC } \Theta \mathcal{B} \Gamma (c1[z ::= v]_v)$ **using** *wf-subst1(2)[OF *, of GNil]* **assms** *subst-gv.simps* *subst-v-c-def*
by *force*

moreover **have** $\forall i. \text{wfI } \Theta \Gamma \ i \wedge \text{is-satis-g } i \Gamma \longrightarrow \text{is-satis } i (c1[z ::= v]_v)$

proof(*rule, rule*)

fix i

assume $\text{as: } \text{wfI } \Theta \Gamma \ i \wedge \text{is-satis-g } i \Gamma$

then obtain sv **where** $sv: \text{eval-v } i \ v \ sv \wedge \text{wfRCV } \Theta \ sv \ b$ **using** *eval-v-exist* **assms** **by** *metis*

hence $\text{is-satis } i (c0[z ::= v]_v)$ **using** *assms* *valid.simps* **as** **by** *metis*

hence $\text{is-satis } (i(z \mapsto sv)) \ c0$ **using** *subst-c-satis* sv **as** *assms* *valid.simps* *wfC-wf* *wfG-elim2*
subst-v-c-def **by** *metis*

moreover **have** $\text{is-satis-g } (i(z \mapsto sv)) \ \Gamma$

using *is-satis-g-i-upd* **assms** **by** (*simp* *add: as*)

ultimately **have** $\text{is-satis-g } (i(z \mapsto sv)) ((z, b, c0) \#_{\Gamma} \Gamma)$

using *is-satis-g.simps* **by** *simp*

moreover **have** $\text{wfI } \Theta ((z, b, c0) \#_{\Gamma} \Gamma) (i(z \mapsto sv))$ **using** *as* *wfI-upd* sv **assms** *valid.simps* *wfC-wf*
by *metis*

ultimately **have** $\text{is-satis } (i(z \mapsto sv)) \ c1$ **using** *assms* *valid.simps* **by** *auto*

thus $\text{is-satis } i (c1[z ::= v]_v)$ **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-full*:

assumes $\Theta ; \mathcal{B} ; ((x, b, c1[z1 ::= V\text{-var } x]_v) \#_{\Gamma} \Gamma) \models c2[z2 ::= V\text{-var } x]_v$ **and**

$\Theta ; \mathcal{B} ; ((x, b, c2[z2 ::= V\text{-var } x]_v) \#_{\Gamma} \Gamma) \models c3[z3 ::= V\text{-var } x]_v$

shows $\Theta ; \mathcal{B} ; ((x, b, c1[z1 ::= V\text{-var } x]_v) \#_{\Gamma} \Gamma) \models c3[z3 ::= V\text{-var } x]_v$

unfolding *valid.simps* **proof**

show $\Theta ; \mathcal{B} ; (x, b, c1[z1 ::= V\text{-var } x]_v) \#_{\Gamma} \Gamma \vdash_{wf} c3[z3 ::= V\text{-var } x]_v$ **using** *wf-trans* *valid.simps*
assms **by** *metis*

show $\forall i. (\text{wfI } \Theta ((x, b, c1[z1 ::= V\text{-var } x]_v) \#_{\Gamma} \Gamma) \ i \wedge (\text{is-satis-g } i ((x, b, c1[z1 ::= V\text{-var } x]_v) \#_{\Gamma} \Gamma)) \longrightarrow (\text{is-satis } i (c3[z3 ::= V\text{-var } x]_v)))$

proof(*rule, rule*)

fix i

assume $\text{as: } \Theta ; (x, b, c1[z1 ::= V\text{-var } x]_v) \#_{\Gamma} \Gamma \vdash i \wedge i \models (x, b, c1[z1 ::= V\text{-var } x]_v) \#_{\Gamma} \Gamma$

have $i \models c2[z2 ::= V\text{-var } x]_v$ **using** *is-satis-g.simps* **as** *assms* **by** *simp*

moreover **have** $i \models \Gamma$ **using** *is-satis-g.simps* **as** **by** *simp*

ultimately **show** $i \models c3[z3 ::= V\text{-var } x]_v$ **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  $\text{atom } x \# c$  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  $\text{atom } x \# c$  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  $G::\Gamma$ 
  assumes  $i' \models G$  and  $\text{atom } x \# G$  and  $i = i' (x \mapsto s)$ 
  shows  $i \models G$ 
  using assms proof(induct G rule:  $\Gamma$ -induct)
  case GNil
  then show ?case by auto
next
  case (GCons x' b' c'  $\Gamma'$ )
  hence  $\text{atom } x \# c'$  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'  $\Gamma'$ ] 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* :: $\Theta \Rightarrow rcl\text{-}val \Rightarrow b \Rightarrow bv \Rightarrow b \Rightarrow rcl\text{-}val \Rightarrow bool$ ($- \vdash - \sim - [- ::= -] \setminus - [50,50]$) **where**

boxed-b-BVar1I: $\llbracket bv = bv' ; wfRCV P s b \rrbracket \Longrightarrow boxed\text{-}b P s (B\text{-}var bv') bv b (Sut s)$
boxed-b-BVar2I: $\llbracket bv \neq bv' ; wfRCV P s (B\text{-}var bv') \rrbracket \Longrightarrow boxed\text{-}b P s (B\text{-}var bv') bv b s$
boxed-b-BIntI: $wfRCV P s B\text{-}int \Longrightarrow boxed\text{-}b P s B\text{-}int - - s$
boxed-b-BBoolI: $wfRCV P s B\text{-}bool \Longrightarrow boxed\text{-}b P s B\text{-}bool - - s$
boxed-b-BUnitI: $wfRCV P s B\text{-}unit \Longrightarrow boxed\text{-}b P s B\text{-}unit - - s$
boxed-b-BPairI: $\llbracket boxed\text{-}b P s1 b1 bv b s1' ; boxed\text{-}b P s2 b2 bv b s2' \rrbracket \Longrightarrow boxed\text{-}b P (SPair s1 s2) (B\text{-}pair b1 b2) bv b (SPair s1' s2')$

| *boxed-b-BConsI*:
 $AF\text{-}typedef\ tyid\ dclist \in set\ P;$
 $(dc, \llbracket x : b \mid c \rrbracket) \in set\ dclist ;$
 $boxed\text{-}b P s1 b bv b' s1'$
 $\llbracket \Longrightarrow$
 $boxed\text{-}b P (SCons tyid dc s1) (B\text{-}id tyid) bv b' (SCons tyid dc s1')$

| *boxed-b-BConspI*: $AF\text{-}typedef\text{-}poly\ tyid\ bva\ dclist \in set\ P;$
 $atom\ bva \# (b1, bv, b', s1, s1');$
 $(dc, \llbracket x : b \mid c \rrbracket) \in set\ dclist ;$
 $boxed\text{-}b P s1 (b[bva::=b1]_{bb}) bv b' s1'$
 $\llbracket \Longrightarrow$
 $boxed\text{-}b P (SConsp tyid dc b1[bv::=b]_{bb} s1) (B\text{-}app tyid b1) bv b' (SConsp tyid dc b1 s1')$

| *boxed-b-Bbitvec*: $wfRCV P s B\text{-}bitvec \Longrightarrow boxed\text{-}b P s B\text{-}bitvec bv b s$

equivariance *boxed-b*

nominal-inductive *boxed-b* .

inductive-cases *boxed-b-elim*:

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-supb-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 $P \vdash SCons\ tyid\ dc\ s1 : (B-id\ tyid)$ **using** *wfRCV.intros* *rcl-val.supp* *subst-bb.simps* *boxed-b-BConsI*
by *metis*
moreover **have** $P \vdash SCons\ tyid\ dc\ s1' : B-id\ tyid$ **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** $\ast: AF\text{-typedef-poly}\ tyid\ bva\ dclist = AF\text{-typedef-poly}\ tyid\ bva2\ dclist2$
 \wedge
 $atom\ bva2 \nmid (bv, (P, SConsp\ tyid\ dc\ b1[bv ::= b']_{bb}\ s1, B-app\ tyid\ b1[bv ::= b']_{bb}))$
using *obtain-fresh-bv* **by** *metis*

then obtain $x2$ **and** $b2$ **and** $c2$ **where** $\ast: (dc, \{x2 : b2 \mid c2\}) \in set\ dclist2$
using *boxed-b-BConspI* *td-lookup-eq-iff* *type-def.eq-iff* **by** *metis*

have $P \vdash SConsp\ tyid\ dc\ b1[bv ::= b']_{bb}\ s1 : (B-app\ tyid\ b1[bv ::= b']_{bb})$ **proof**
show $1: \langle AF\text{-typedef-poly}\ tyid\ bva2\ dclist2 \in set\ P \rangle$ **using** *boxed-b-BConspI* **by** *auto*
show $2: \langle (dc, \{x2 : b2 \mid c2\}) \in set\ dclist2 \rangle$ **using** *boxed-b-BConspI* **using** \ast **by** *simp*

hence $atom\ bv \nmid b2$ **proof** –
have *supp* $b2 \subseteq \{atom\ bva2\}$ **using** *wfTh-poly-supb-b* $1\ 2$ *boxed-b-BConspI* **by** *auto*
moreover **have** $bv \neq bva2$ **using** \ast *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* $\ast\ 2$ *boxed-b-BConspI*
by *metis*
ultimately show $\langle P \vdash s1 : b2[bva2 ::= b1[bv ::= b']_{bb}]_{bb} \rangle$ **using** *boxed-b-BConspI* *subst-b-b-def* *subst-bb-commute* **by** *auto*
show $atom\ bva2 \nmid (P, SConsp\ tyid\ dc\ b1[bv ::= b']_{bb}\ s1, B-app\ tyid\ b1[bv ::= b']_{bb})$ **using** \ast *fresh-Pair*
by *metis*
qed

moreover **have** $P \vdash SConsp\ tyid\ dc\ b1\ s1' : B-app\ tyid\ b1$ **proof**
show $\langle AF\text{-typedef-poly}\ tyid\ bva\ dclist \in set\ P \rangle$ **using** *boxed-b-BConspI* **by** *auto*
show $\langle (dc, \{x : b \mid c\}) \in set\ dclist \rangle$ **using** *boxed-b-BConspI* **by** *auto*

```

  show  $\langle P \vdash s1' : b[bva::=b1]_{bb} \rangle$  using boxed-b-BConspI by auto
  have atom bva  $\sharp P$  using boxed-b-BConspI wfTh-fresh by metis
  thus atom bva  $\sharp (P, SConsp tyid dc b1 s1', B\text{-app tyid } b1)$  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 ::  $\Theta \Rightarrow \Gamma \Rightarrow b \Rightarrow bv \Rightarrow \text{valuation} \Rightarrow \text{valuation} \Rightarrow \text{bool}$  ( $- ; - ; - , - \vdash - \approx -$  [50,50])
where
  boxed-i-GNil:  $\Theta ; GNil ; b , bv \vdash i \approx i$ 
  | boxed-i-GConsI:  $\llbracket \text{Some } s = i x ; \text{boxed-b } \Theta s b bv b' s' ; \Theta ; \Gamma ; b' , bv \vdash i \approx i' \rrbracket \implies \Theta ;$ 
 $((x,b,c)\#_{\Gamma}\Gamma) ; b' , bv \vdash i \approx (i'(x \mapsto s'))$ 
equivariance boxed-i
nominal-inductive boxed-i .

```

```

inductive-cases boxed-i-elim:
   $\Theta ; GNil ; b , bv \vdash i \approx i'$ 
   $\Theta ; ((x,b,c)\#_{\Gamma}\Gamma) ; b' , bv \vdash i \approx i'$ 

```

```

lemma wfRCV-poly-elim:
  fixes  $tm::'a::fs$  and  $b::b$ 
  assumes  $T \vdash SConsp tyid dc bdc s : b$ 
  obtains  $bva \text{ dclist } x1 b1 c1$  where  $b = B\text{-app tyid } bdc \wedge$ 
 $AF\text{-typedef-poly tyid } bva \text{ dclist} \in \text{set } T \wedge (dc, \llbracket x1 : b1 \mid c1 \rrbracket) \in \text{set dclist} \wedge T \vdash s : b1[bva::=bdc]_{bb}$ 
 $\wedge \text{atom } bva \sharp tm$ 
using assms proof(nominal-induct SConsp tyid dc bdc s b avoiding: tm rule:wfRCV.strong-induct)
  case (wfRCV-BConsPI bv dclist  $\Theta x b c$ )
  then show ?case by simp
qed

```

```

lemma boxed-b-ex:
  assumes  $wfRCV T s b[bv::=b]_{bb}$  and  $wfTh T$ 
  shows  $\exists s'. \text{boxed-b } T s b bv b' s'$ 
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-elim(9)[OF SBitvec(1)] by metis
  show ?case proof (cases  $b = B\text{-var } bv$ )
  case True
  moreover have  $T \vdash SBitvec x : B\text{-bitvec}$  using wfRCV.intros by simp
  moreover hence  $b' = B\text{-bitvec}$  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-elim(10)[OF SNum(1)] by metis
show ?case proof (cases b = B-var bv)
case True
moreover have T ⊢ 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-elim(11)[OF SBool(1)] by metis
show ?case proof (cases b = B-var bv)
case True
moreover have T ⊢ 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 ∧ wfRCV T s1 b1 ∧ wfRCV T s2 b2
using wfRCV-elim(12) by metis
show ?case proof (cases b = B-var bv)
case True
moreover have T ⊢ 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' ∧ b1 = b1'[bv::=b]_bb ∧ 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-elim(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  $local.*\ SCons.premis\ by\ auto$ 
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  $**: AF-typedef\ tyid\ dclist \in set\ T \wedge (dc, \llbracket x : b2 \mid c \rrbracket) \in set\ dclist$ 
 $\wedge wfRCV\ T\ s1\ b2$  using  $wfRCV.elims(13) *$   $SCons$  by  $metis$ 
  then have  $atom\ bv \# b2$  using  $\langle wfTh\ T \rangle\ wfTh-lookup-supply-empty[of\ tyid\ dclist\ T\ dc\ \llbracket x : b2 \mid c \rrbracket]$ 
 $\tau.fresh\ fresh-def$  by  $auto$ 
  then have  $b2 = b2[ bv ::= b ]_{bb}$  using  $forget-subst\ subst-b-b-def$  by  $metis$ 
  then obtain  $s1'$  where  $s1:T \vdash s1 \sim b2[ bv ::= b' ] \setminus s1'$  using  $SCons **$  by  $metis$ 

  have  $T \vdash SCons\ tyid\ dc\ s1 \sim (B-id\ tyid)\ [ bv ::= b' ] \setminus SCons\ tyid\ dc\ s1'$  proof  $(rule\ boxed-b-BConsI)$ 
  show  $AF-typedef\ tyid\ dclist \in set\ T$  using  $**$  by  $auto$ 
  show  $(dc, \llbracket x : b2 \mid c \rrbracket) \in set\ dclist$  using  $**$  by  $auto$ 
  show  $T \vdash s1 \sim b2[ bv ::= b' ] \setminus s1'$  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  $**: b[bv ::= b]_{bb} = B-app\ typid\ bdc \wedge$ 
 $AF-typedef-poly\ typid\ bva\ dclist \in set\ T \wedge (dc, \llbracket x1 : b1 \mid c1 \rrbracket) \in set\ dclist \wedge T \vdash s : b1[bva ::= bdc]_{bb}$ 
 $\wedge atom\ bva \# bv$ 
  using  $wfRCV-poly-elimis[OF\ SConsp(2)]$  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  $T \vdash SConsp\ typid\ dc\ bdc\ s : B-app\ typid\ bdc$  using  $wfRCV.intros$ 
  using  $local.*\ SConsp.premis(1)$  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  $bdc: b = B-app\ typid\ bdc' \wedge bdc = bdc'[bv ::= b]_{bb}$  using  $*\ subst-bb-inject(8)[OF\$ 
 $*$  by  $metis$ 

  have  $atom\ bv \# b1$  proof  $-$ 
  have  $supp\ b1 \subseteq \{ atom\ bva \}$  using  $wfTh-poly-supply-b **\ SConsp$  by  $metis$ 
  moreover have  $bv \neq bva$  using  $**$  by  $auto$ 
  ultimately show  $?thesis$  using  $fresh-def$  by  $force$ 
  qed
  have  $T \vdash s : b1[bva ::= bdc]_{bb}$  using  $**$  by  $auto$ 
  moreover have  $b1[bva ::= bdc]_{bb}[bv ::= b]_{bb} = b1[bva ::= bdc]_{bb}$  using  $bdc\ subst-bb-commute\ \langle atom\ bv \# b1 \rangle$  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  $T \vdash SConsp\ typid\ dc\ bdc'[bv ::= b]_{bb}\ s \sim (B\text{-}app\ typid\ bdc')\ [bv ::= b'] \setminus SConsp\ typid\ dc\ bdc'\ s'$ 
proof –

obtain  $bva3$  and  $dclist3$  where  $3:AF\text{-}typedef\text{-}poly\ typid\ bva3\ dclist3 = AF\text{-}typedef\text{-}poly\ typid\ bva\ dclist \wedge$ 
   $atom\ bva3 \# (bdc', bv, b', s, s')$  using obtain-fresh-bv by metis
then obtain  $x3\ b3\ c3$  where  $4:(dc, \{x3 : b3 \mid c3\}) \in set\ dclist3$ 
  using boxed-b-BConspI td-lookup-eq-iff type-def.eq-iff
  by (metis **)

show ?thesis proof
  show  $\langle AF\text{-}typedef\text{-}poly\ typid\ bva3\ dclist3 \in set\ T \rangle$  using  $3$  ** by metis
  show  $\langle atom\ bva3 \# (bdc', bv, b', s, s') \rangle$  using  $3$  by metis
  show  $4:\langle (dc, \{x3 : b3 \mid c3\}) \in set\ dclist3 \rangle$  using  $4$  by auto
  have  $b3[bva3 ::= bdc]_{bb} = b1[bva ::= bdc]_{bb}$  proof(rule wfTh-typedef-poly-b-eq-iff)
    show  $\langle AF\text{-}typedef\text{-}poly\ typid\ bva3\ dclist3 \in set\ T \rangle$  using  $3$  ** by metis
    show  $\langle (dc, \{x3 : b3 \mid c3\}) \in set\ dclist3 \rangle$  using  $4$  by auto
    show  $\langle AF\text{-}typedef\text{-}poly\ typid\ bva\ dclist \in set\ T \rangle$  using ** by auto
    show  $\langle (dc, \{x1 : b1 \mid c1\}) \in set\ dclist \rangle$  using ** by auto
    qed(simp add: ** SConsp)
    thus  $\langle T \vdash s \sim b3[bva3 ::= bdc]_{bb}\ [bv ::= b'] \setminus s' \rangle$  using  $s'$  by auto
  qed
qed
then show ?thesis using bdc by auto

qed
next
case SUnit
  have  $*:b[bv ::= b]_{bb} = B\text{-}unit$  using wfRCV-elimS SUnit by metis
show ?case proof (cases b = B-var bv)
  case True
    moreover have  $T \vdash SUnit : B\text{-}unit$  using wfRCV.intros by simp
    moreover hence  $b' = B\text{-}unit$  using True SUnit subst-bb.simps * by simp
    ultimately show ?thesis using boxed-b.intros wfRCV.intros by metis
  next
  case False
    hence  $b = B\text{-}unit$  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  $*:b[bv ::= b]_{bb} = B\text{-}var\ bv'$  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 $wfI\ T\ \Gamma[bv::=b]_{\Gamma b}\ i$ and $wfTh\ T$
 shows $\exists i'.\ T\ ;\ \Gamma\ ;\ b\ ,\ bv \vdash i \approx i'$
 using *assms* **proof**(*induct* Γ *arbitrary: i* *rule:Γ-induct*)
 case *GNil*
 then show *?case* using *boxed-i-GNilI* by *metis*
 next
 case (*GCons* $x'\ b'\ c'\ \Gamma'$)
 then obtain s where $1:Some\ s = i\ x' \wedge wfRCV\ T\ s\ b'[bv::=b]_{bb}$ 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 $3:boxed-i\ T\ \Gamma'\ b\ bv\ i\ i'$ using *GCons wfI-def subst-gb.simps* by *force*
 have *boxed-i* $T\ ((x',\ b',\ c')\ \#_{\Gamma}\ \Gamma')\ b\ bv\ i\ (i'(x' \mapsto s'))$ **proof**
 show $Some\ s = i\ x'$ using 1 by *auto*
 show *boxed-b* $T\ s\ b'\ bv\ b\ s'$ using 2 by *auto*
 show $T\ ;\ \Gamma'\ ;\ b\ ,\ bv \vdash i \approx i'$ using 3 by *auto*
 qed
 thus *?case* by *auto*
 qed

lemma boxed-b-eq:

assumes *boxed-b* $\Theta\ s1\ b\ bv\ b'\ s1'$ and $\vdash_{wf}\ \Theta$
 shows $wfTh\ \Theta \implies boxed-b\ \Theta\ s2\ b\ bv\ b'\ s2' \implies (s1 = s2) = (s1' = s2')$
 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-elim1(1) rcl-val.eq-iff* by *metis*
 next
 case (*boxed-b-BVar2I* $bv\ bv'\ P\ s\ b$)
 then show *?case* using *boxed-b-elim1(1)* by *metis*
 next
 case (*boxed-b-BIntI* $P\ s\ uv\ uv$)
 hence $s2 = s2'$ using *boxed-b-elim1* by *metis*
 then show *?case* by *auto*
 next
 case (*boxed-b-BBoolI* $P\ s\ uw\ ux$)
 hence $s2 = s2'$ using *boxed-b-elim1* by *metis*
 then show *?case* by *auto*
 next
 case (*boxed-b-BUnitI* $P\ s\ uy\ uz$)
 hence $s2 = s2'$ using *boxed-b-elim1* 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-elim1(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 $*:s2 = SCons\ tyid\ dc2\ s22 \wedge s2' = SCons\ tyid\ dc2$

```

s22'  $\wedge$  boxed-b P s22 b2 bv b' s22'
   $\wedge$  AF-typedef tyid dclist2  $\in$  set P  $\wedge$  (dc2,  $\{ x2 : b2 \mid c2 \}$ )  $\in$  set dclist2 using boxed-b-elim(6)[OF
boxed-b-BConsI(6)] by metis
show ?case proof(cases dc = dc2)
  case True
  hence b = b2 using wfTh-ctor-unique  $\tau$ .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-elim by metis
  then show ?case by auto
next
  case (boxed-b-BConsI tyid bva dclist P b1 bv b' s1 s1' dc x b c)
  thm boxed-b-elim(8)[OF boxed-b-BConsI(7)]
  obtain bva2 s22 s22' dclist2 dc2 x2 b2 c2 where *:
    s2 = SConsp tyid dc2 b1 [bv::=b]bb s22  $\wedge$ 
    s2' = SConsp tyid dc2 b1 s22'  $\wedge$ 
    boxed-b P s22 b2 [bva2::=b1]bb bv b' s22'  $\wedge$ 
    AF-typedef-poly tyid bva2 dclist2  $\in$  set P  $\wedge$  (dc2,  $\{ x2 : b2 \mid c2 \}$ )  $\in$  set dclist2 using boxed-b-elim(8)[OF
boxed-b-BConsI(7)] by metis
  show ?case proof(cases dc = dc2)
    case True
    hence AF-typedef-poly tyid bva2 dclist2  $\in$  set P  $\wedge$  (dc,  $\{ x2 : b2 \mid c2 \}$ )  $\in$  set dclist2 using * by
    auto
    hence b[bva2::=b1]bb = b2[bva2::=b1]bb using wfTh-typedef-poly-b-eq-iff[OF boxed-b-BConsI(1)
boxed-b-BConsI(3)] * 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
qed

```

lemma bs-boxed-var:

```

assumes boxed-i  $\Theta$   $\Gamma$  b' bv i i'
shows Some (b,c) = lookup  $\Gamma$  x  $\implies$  Some s = i x  $\implies$  Some s' = i' x  $\implies$  boxed-b  $\Theta$  s b bv b' s'
using asms proof(induct rule: boxed-i.inducts)
  case (boxed-i-GNil T i)
  then show ?case using lookup.simps by auto
next
  case (boxed-i-GConsI s i x1  $\Theta$  b1 bv b' s'  $\Gamma$  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 $\llbracket l \rrbracket = s$
shows $\text{boxed-b } \Theta s (\text{base-for-lit } l) \text{ bv } b' s$
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 $\text{boxed-i } \Theta \Gamma b' \text{ bv } i i'$ **and** $i \llbracket v[bv::=b]_{vb} \rrbracket \sim s$ **and** $i' \llbracket v \rrbracket \sim s'$ **and** $\text{wfV } \Theta B \Gamma v b$
and $\text{wfI } \Theta \Gamma i'$
shows $\text{boxed-b } \Theta s b \text{ bv } b' s'$
using *assms* **proof**(*nominal-induct v arbitrary: s s' b rule:v.strong-induct*)
case (*V-lit l*)
hence $\llbracket l \rrbracket = s \wedge \llbracket l \rrbracket = s'$ **using** *eval-v-elim*s **by** *auto*
moreover **have** $b = \text{base-for-lit } l$ **using** *wfV-elim*s(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 $\text{Some } s = i x \wedge \text{Some } s' = i' x$ **using** *eval-v-elim*s *subst-vb.simps* **by** *metis*
moreover **obtain** $c1$ **where** $\text{bc:Some } (b, c1) = \text{lookup } \Gamma x$ **using** *wfV-elim*s *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** $b:b=B\text{-pair } b1 b2$ **using** *wfV-elim*s *subst-vb.simps* **by** *metis*
obtain $s1$ **and** $s2$ **where** $s: \text{eval-v } i (v1[bv::=b]_{vb}) s1 \wedge \text{eval-v } i (v2[bv::=b]_{vb}) s2 \wedge s = \text{SPair } s1 s2$
using *eval-v-elim*s *V-pair subst-vb.simps* **by** *metis*
obtain $s1'$ **and** $s2'$ **where** $s': \text{eval-v } i' v1 s1' \wedge \text{eval-v } i' v2 s2' \wedge s' = \text{SPair } s1' s2'$ **using** *eval-v-elim*s *V-pair* **by** *metis*
thm *boxed-b-BPairI*
have $\text{boxed-b } \Theta (\text{SPair } s1 s2) (B\text{-pair } b1 b2) \text{ bv } b' (\text{SPair } s1' s2')$ **proof**(*rule boxed-b-BPairI*)
show $\text{boxed-b } \Theta s1 b1 \text{ bv } b' s1'$ **using** *V-pair eval-v-elim*s *wfV-elim*s $b s s' b.\text{eq-iff}$ **by** *metis*
show $\text{boxed-b } \Theta s2 b2 \text{ bv } b' s2'$ **using** *V-pair eval-v-elim*s *wfV-elim*s $b s s' b.\text{eq-iff}$ **by** *metis*
qed
then show *?case using s s' b* **by** *auto*
next
case (*V-cons tyid dc v1*)

obtain $\text{dclist } x b1 c$ **where** $*$: $b = B\text{-id } \text{tyid} \wedge \text{AF-typedef } \text{tyid } \text{dclist} \in \text{set } \Theta \wedge (\text{dc}, \llbracket x : b1 \mid c \rrbracket) \in \text{set } \text{dclist} \wedge \Theta ; B ; \Gamma \vdash_{\text{wf}} v1 : b1$
using *wfV-elim*s(4)[*OF V-cons*(5)] *V-cons* **by** *metis*
obtain $s2$ **where** $s2: s = \text{SCons } \text{tyid } \text{dc } s2 \wedge i \llbracket (v1[bv::=b]_{vb}) \rrbracket \sim s2$ **using** *eval-v-elim*s *V-cons* *subst-vb.simps* **by** *metis*
obtain $s2'$ **where** $s2': s' = \text{SCons } \text{tyid } \text{dc } s2' \wedge i' \llbracket v1 \rrbracket \sim s2'$ **using** *eval-v-elim*s *V-cons* **by** *metis*
have $\text{sp: supp } \llbracket x : b1 \mid c \rrbracket = \{\}$ **using** *wfTh-lookup-supp-empty* $*$ *wfX-wfY* **by** *metis*

```

have boxed-b  $\Theta$  (SCons tyid dc s2) (B-id tyid) bv b' (SCons tyid dc s2')
proof(rule boxed-b-BConsI)
  show 1:AF-typedef tyid dclist  $\in$  set  $\Theta$  using * by auto
  show 2:(dc,  $\{ x : b1 \mid c \}$ )  $\in$  set dclist using * by auto
  have bv:atom bv  $\#$  b1 using sp  $\tau$ .fresh fresh-def by auto
  show  $\Theta \vdash s2 \sim b1 [bv ::= b'] \setminus s2'$  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  $\wedge$  AF-typedef-poly tyid bv2 dclist  $\in$  set  $\Theta \wedge$ 
  (dc,  $\{ x2 : b2 \mid c2 \}$ )  $\in$  set dclist  $\wedge$   $\Theta ; B ; \Gamma \vdash_{wf} v1 : b2[bv2 ::= b1]_{bb}$ 
  using wf-strong-elim(1)[OF V-consp (5)] by metis

obtain s2 where s2: s = SConsp tyid dc b1[bv ::= b]_{bb} s2  $\wedge$  i  $\llbracket (v1[bv ::= b]_{vb}) \rrbracket \sim s2$ 
  using eval-v-elim V-consp subst-vb.simps by metis

obtain s2' where s2': s' = SConsp tyid dc b1 s2'  $\wedge$  i'  $\llbracket v1 \rrbracket \sim s2'$ 
  using eval-v-elim V-consp by metis

thm obtain-fresh-bv-dclist-b-iff

have  $\vdash_{wf} \Theta$  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  $\wedge$ 
  (dc,  $\{ x3 : b3 \mid c3 \}$ )  $\in$  set dclist3  $\wedge$  atom bv3  $\#$  (b1, bv, b', s2, s2')  $\wedge$  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  $\Theta$  (SConsp tyid dc b1[bv ::= b]_{bb} s2) (B-app tyid b1) bv b' (SConsp tyid dc b1 s2')
proof(rule boxed-b-BConsI[of tyid bv3 dclist3  $\Theta$ , where x=x3 and b=b3 and c=c3])
  show 1:AF-typedef-poly tyid bv3 dclist3  $\in$  set  $\Theta$  using ** by auto
  show 2:(dc,  $\{ x3 : b3 \mid c3 \}$ )  $\in$  set dclist3 using ** by auto
  show atom bv3  $\#$  (b1, bv, b', s2, s2') using ** by auto
  show  $\Theta \vdash s2 \sim b3[bv3 ::= b1]_{bb} [bv ::= b'] \setminus s2'$  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'  $\llbracket e \rrbracket \sim s'$  and i  $\llbracket e[bv ::= b]_{ceb} \rrbracket \sim s$  and wfCE  $\Theta B \Gamma e b$  and boxed-i  $\Theta \Gamma b' bv i i'$ 
  and wfI  $\Theta \Gamma i'$ 
  shows boxed-b  $\Theta s b bv b' s'$ 
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-elim wfCE-elim subst-ceb.simps by metis

```

next

case (CE-op opp v1 v2)

have 1:wfCE $\Theta B \Gamma v1$ (B-int) using wfCE-elim CE-op by metis

have 2:wfCE $\Theta B \Gamma v2$ (B-int) using wfCE-elim 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-elim Plus by metis

obtain n1 and n2 where n:s = SNum (n1 + n2) $\wedge i \llbracket v1[bv::=b]_{ceb} \rrbracket \sim SNum\ n1 \wedge i \llbracket v2[bv::=b]_{ceb} \rrbracket \sim SNum\ n2$ using eval-e-elim CE-op subst-ceb.simps Plus by metis

obtain n1' and n2' where n':s' = SNum (n1' + n2') $\wedge i' \llbracket v1 \rrbracket \sim SNum\ n1' \wedge i' \llbracket v2 \rrbracket \sim SNum\ n2'$ using eval-e-elim 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-elim(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-elim by metis

obtain n1 and n2 where n:s = SBool (n1 \leq n2) $\wedge i \llbracket v1[bv::=b]_{ceb} \rrbracket \sim SNum\ n1 \wedge i \llbracket v2[bv::=b]_{ceb} \rrbracket \sim SNum\ n2$ using eval-e-elim subst-ceb.simps CE-op LEq by metis

obtain n1' and n2' where n':s' = SBool (n1' \leq n2') $\wedge i' \llbracket v1 \rrbracket \sim SNum\ n1' \wedge i' \llbracket v2 \rrbracket \sim SNum\ n2'$ using eval-e-elim 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-elim(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) $\wedge (i \llbracket v1[bv::=b]_{ceb} \rrbracket \sim SBitvec\ bv1) \wedge i \llbracket v2[bv::=b]_{ceb} \rrbracket \sim SBitvec\ bv2$

using eval-e-elim(6) subst-ceb.simps CE-concat.prem(2) eval-e-elim(6) subst-ceb.simps(6) by metis

obtain bv1' and bv2' where s' : s' = SBitvec (bv1' @ bv2') $\wedge i' \llbracket v1 \rrbracket \sim SBitvec\ bv1' \wedge i' \llbracket v2 \rrbracket \sim SBitvec\ bv2'$

using eval-e-elim(6) CE-concat by metis

then show ?case using boxed-i-eval-v-boxed-b wfCE-elim s s' CE-concat

by (metis CE-concat.premis(3) assms assms(5) wfRCV-BBitvec boxed-b-Bbitvec boxed-b-elimis(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-elimis subst-ceb.simps by
 metis
 obtain s2' where 2:i' [[ce]] ~ SPair s' s2' using CE-fst eval-e-elimis by metis
 obtain b2 where 3:wfCE Θ B Γ ce (B-pair b b2) using wfCE-elimis(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-elimis(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-elimis subst-ceb.simps by
 metis
 obtain s1' where 2:i' [[v]] ~ SPair s1' s' using CE-snd eval-e-elimis by metis
 obtain b1 where 3:wfCE Θ B Γ v (B-pair b1 b) using wfCE-elimis(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-elimis(5) by force
 next
 case (CE-len v)
 obtain s1 where s: i [[v[bv::=b]_{ceb}]] ~ SBitvec s1 using CE-len eval-e-elimis subst-ceb.simps by
 metis
 obtain s1' where s': i' [[v]] ~ SBitvec s1' using CE-len eval-e-elimis by metis

 have Θ ; B ; $\Gamma \vdash_{wf} v : B\text{-bitvec} \wedge b = B\text{-int}$ using wfCE-elimis 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-elimis(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 ; $\Gamma[bv::=b]_{\Gamma b}$
 $\vdash i$
 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-elimis subst-cb.simps by metis
 next
 case C-false
 then show ?case using eval-c-elimis 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 \wedge eval-c i (c2[bv::=b]_{cb}) s2 \wedge s = (s1 \wedge s2)
 using C-conj eval-c-elimis(3) subst-cb.simps(3) by metis
 obtain s1' and s2' where 2:eval-c i' c1 s1' \wedge eval-c i' c2 s2' \wedge s' = (s1' \wedge s2') using C-conj
 eval-c-elimis(3) by metis

then show ?case using 1 2 wfC-elim 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-elim(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-elim(4) by metis
 then show ?case using 1 2 wfC-elim 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-elim(6)
 subst-cb.simps(7) by metis
 obtain s1':bool where 2: (i' [c] ~ s1') ∧ (s' = (¬ s1')) using C-not eval-c-elim(6) by metis
 then show ?case using 1 2 wfC-elim 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-elim(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-elim(5) by metis
 then show ?case using 1 2 wfC-elim 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-elim 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-elim(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-elim(7) by metis
 have ⊢_{wf} Θ using C-eq wfX-wfY by metis
 moreover have Θ ; Γ[bv::=b]_{Γb} ⊢ i using C-eq by auto
 ultimately show ?case using boxed-b-eq[of Θ s1 be bv b s1' s2'] 1 2 boxed-i-eval-ce-boxed-b C-eq
 wfC-elim 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]_{Γb} 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 asms by auto
 moreover obtain s where i' [c] ~ s using eval-c-exist asms by metis
 ultimately show ?thesis using eval-c-eq-bs-boxed asms 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]_{Γb} i and Θ ; Γ ⊢ i' and wfC
 Θ {||} Γ[bv::=b]_{Γb} (c[bv::=b]_{cb})
 and (i' ⊢ 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 $\Theta::\Theta$ and $B::B$

assumes $boxed-i\ \Theta\ \Gamma\ b\ bv\ i\ i'$ and $wfI\ \Theta\ \Gamma[bv::=b]_{\Gamma b}\ i$ and $\vdash_{wf}\ \Theta$ and $wfG\ \Theta\ B\ \Gamma$

shows $\Theta ; \Gamma \vdash i'$

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 $as : (x2, b2, c2) \in toSet\ ((x1, b1, c1) \#_{\Gamma} G)$

then consider (*hd*) $(x2, b2, c2) = (x1, b1, c1) \mid (tail)\ (x2, b2, c2) \in toSet\ G$ using *toSet.simps* by

auto

hence $\exists s. Some\ s = (i'(x1 \mapsto s'))\ x2 \wedge wfRCV\ T\ s\ b2$ **proof**(*cases*)

case *hd*

hence $b1=b2$ by *auto*

moreover have $(x2, b2[bv::=b]_{bb}, c2[bv::=b]_{cb}) \in toSet\ ((x1, b1, c1) \#_{\Gamma} G)[bv::=b]_{\Gamma b}$ 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 $ss :: b \Rightarrow x \Rightarrow (x \Rightarrow rcl-val\ option) \Rightarrow type-def\ list \Rightarrow rcl-val$ where

$\forall x1a\ x2a\ x3\ x4. (\exists v5. Some\ v5 = x3\ x2a \wedge wfRCV\ x4\ v5\ x1a) = (Some\ (ss\ x1a\ x2a\ x3\ x4) =$
 $x3\ x2a \wedge wfRCV\ x4\ (ss\ x1a\ x2a\ x3\ x4)\ x1a)$

by *moura*

then have $f1: Some\ (ss\ b2[bv::=b]_{bb}\ x1\ i\ T) = i\ x1 \wedge wfRCV\ T\ (ss\ b2[bv::=b]_{bb}\ x1\ i\ T)$
 $b2[bv::=b]_{bb}$

using *boxed-i-GConsI.prem1* *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* *toSet.simps*(2) *wfG-cons2* *wfI-def*)

then show $?thesis$ using *wfI-def*[*of* $T\ G\ i'$] *tail*

using *boxed-i-GConsI.prem3* *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::Γ
  assumes boxed-i  $\Theta$  G1 b bv i i' and wfI  $\Theta$  G1[bv::=b]Γb i and wfI  $\Theta$  G1 i' and G1 = (G2@G)
and wfG  $\Theta$  B G1
  and (i  $\models$  G[bv::=b]Γb)
  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  $\Theta$  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'  $\models$  Γ' using GCons append-g.simps by simp
  qed
qed

lemma is-satis-g-bs-boxed:
  fixes G::Γ
  assumes boxed-i  $\Theta$  G b bv i i' and wfI  $\Theta$  G[bv::=b]Γb i and wfI  $\Theta$  G i' and wfG  $\Theta$  B G
  and (i  $\models$  G[bv::=b]Γb)
  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  $\Theta$  ; {||}  $\vdash_{wf}$  b and B = {|bv|} and  $\Theta$  ; {|bv|} ; Γ  $\models$  c
  shows  $\Theta$  ; {||} ; Γ[bv::=b]Γb  $\models$  c[bv::=b]cb
proof(rule validI)
  show *:  $\Theta$  ; {||} ; Γ[bv::=b]Γb  $\vdash_{wf}$  c[bv::=b]cb using assms valid.simps wf-b-subst subst-gb.simps
  by metis
  show  $\forall i. (wfI \Theta \Gamma[bv::=b]_{\Gamma b} i \wedge i \models \Gamma[bv::=b]_{\Gamma b}) \longrightarrow i \models c[bv::=b]_{cb}$ 
  proof(rule, rule)
    fix i
    assume *: wfI  $\Theta$  Γ[bv::=b]Γb i  $\wedge$  i  $\models$  Γ[bv::=b]Γb

    obtain i' where idash: boxed-i  $\Theta$  Γ b bv i i' using boxed-i-ex wfX-wfY assms * by fastforce

    have wfc:  $\Theta$  ; {|bv|} ; Γ  $\vdash_{wf}$  c using valid.simps assms by simp
    have wfg:  $\Theta$  ; {|bv|}  $\vdash_{wf}$  Γ using valid.simps wfX-wfY assms by metis
  end
end

```

hence $wfI: wfI \Theta \Gamma i'$ **using** $idash * bs\text{-}boxed\text{-}wfi\text{-}aux \text{ subst-gb.simps } wfX\text{-}wfY$ **by** $metis$
 moreover **have** $i' \models \Gamma$ **proof** ($rule \text{ is-satis-g-bs-boxed}[OF \text{ idash}] \text{ wfX-wfY}(2)[OF \text{ wfc}]$)
 show $wfI \Theta \Gamma[bv::=b]_{\Gamma b} i$ **using** $\text{subst-gb.simps} *$ **by** simp
 show $wfI \Theta \Gamma i'$ **using** wfI **by** $auto$
 show $\Theta ; B \vdash_{wf} \Gamma$ **using** $wfg \text{ assms}$ **by** $auto$
 show $i \models \Gamma[bv::=b]_{\Gamma b}$ **using** $\text{subst-gb.simps} *$ **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**($rule \text{ is-satis-bs-boxed-rev}$)
 show $\Theta ; \Gamma ; b, bv \vdash i \approx i'$ **using** $idash$ **by** $auto$
 show $\Theta ; B ; \Gamma \vdash_{wf} c$ **using** $wfc \text{ assms}$ **by** $auto$
 show $\Theta ; \Gamma[bv::=b]_{\Gamma b} \vdash i$ **using** $\text{subst-gb.simps} *$ **by** simp
 show $\Theta ; \Gamma \vdash i'$ **using** wfI **by** $auto$
 show $\Theta ; \{\|\} ; \Gamma[bv::=b]_{\Gamma b} \vdash_{wf} c[bv::=b]_{cb}$ **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 $i \models (CE\text{-val} (V\text{-var } x) == CE\text{-val} (V\text{-lit} (L\text{-num} (int (length v)))))$ (**is** $\text{is-satis } i ?c1$)
shows $i \models (CE\text{-val} (V\text{-var } x) == CE\text{-len} [V\text{-lit} (L\text{-bitvec } v)]^{ce})$
proof –
 have $*:eval\text{-}c \ i \ ?c1 \ \text{True}$ **using** $\text{assms is-satis.simps}$ **by** blast
 then have $eval\text{-}e \ i \ (CE\text{-val} (V\text{-lit} (L\text{-num} (int (length v))))) (SNum (int (length v)))$
 using $eval\text{-}e\text{-}elims(1) \ eval\text{-}v\text{-}elims \ eval\text{-}l.simps$ **by** ($metis \ eval\text{-}e.intros(1) \ eval\text{-}v\text{-}litI$)
 hence $eval\text{-}e \ i \ (CE\text{-val} (V\text{-var } x)) (SNum (int (length v)))$ **using** $eval\text{-}c\text{-}elims(7)[OF *]$
 by ($metis \ eval\text{-}e\text{-}elims(1) \ eval\text{-}v\text{-}elims(1)$)
 moreover have $eval\text{-}e \ i \ (CE\text{-len} [V\text{-lit} (L\text{-bitvec } v)]^{ce}) (SNum (int (length v)))$
 using $eval\text{-}e\text{-}elims(7) \ eval\text{-}v\text{-}elims \ eval\text{-}l.simps$ **by** ($metis \ eval\text{-}e.intros \ eval\text{-}v\text{-}litI$)
 ultimately show $?thesis$ **using** $eval\text{-}c.intros \ \text{is-satis.simps}$ **by** fastforce
 qed

lemma is-satis-plus-imp :

assumes $i \models (CE\text{-val} (V\text{-var } x) == CE\text{-val} (V\text{-lit} (L\text{-num} (n1+n2))))$ (**is** $\text{is-satis } i ?c1$)
shows $i \models (CE\text{-val} (V\text{-var } x) == CE\text{-op Plus} ([V\text{-lit} (L\text{-num } n1)]^{ce}) ([V\text{-lit} (L\text{-num } n2)]^{ce}))$
proof –
 have $*:eval\text{-}c \ i \ ?c1 \ \text{True}$ **using** $\text{assms is-satis.simps}$ **by** blast
 then have $eval\text{-}e \ i \ (CE\text{-val} (V\text{-lit} (L\text{-num} (n1+n2)))) (SNum (n1+n2))$
 using $eval\text{-}e\text{-}elims(1) \ eval\text{-}v\text{-}elims \ eval\text{-}l.simps$ **by** ($metis \ eval\text{-}e.intros(1) \ eval\text{-}v\text{-}litI$)
 hence $eval\text{-}e \ i \ (CE\text{-val} (V\text{-var } x)) (SNum (n1+n2))$ **using** $eval\text{-}c\text{-}elims(7)[OF *]$
 by ($metis \ eval\text{-}e\text{-}elims(1) \ eval\text{-}v\text{-}elims(1)$)
 moreover have $eval\text{-}e \ i \ (CE\text{-op Plus} ([V\text{-lit} (L\text{-num } n1)]^{ce}) ([V\text{-lit} (L\text{-num } n2)]^{ce})) (SNum (n1+n2))$
 using $eval\text{-}e\text{-}elims(7) \ eval\text{-}v\text{-}elims \ eval\text{-}l.simps$ **by** ($metis \ eval\text{-}e.intros \ eval\text{-}v\text{-}litI$)
 ultimately show $?thesis$ **using** $eval\text{-}c.intros \ \text{is-satis.simps}$ **by** fastforce
 qed

lemma *is-satis-leq-imp*:

assumes $i \models (CE\text{-val } (V\text{-var } x) == CE\text{-val } (V\text{-lit } (if (n1 \leq n2) \text{ then } L\text{-true else } L\text{-false})))$ (**is** *is-satis* i ?*c1*)

shows $i \models (CE\text{-val } (V\text{-var } x) == CE\text{-op } LEq [(V\text{-lit } (L\text{-num } n1))]^{ce} [(V\text{-lit } (L\text{-num } n2))]^{ce})$

proof –

have *:*eval-c* i ?*c1* *True* **using** *assms is-satis.simps* **by** *blast*

then have *eval-e* i ($CE\text{-val } (V\text{-lit } ((if (n1 \leq n2) \text{ then } L\text{-true else } L\text{-false})))$) (*SBool* ($n1 \leq n2$))

using *eval-e-elim*(1) *eval-v-elim* *eval-l.simps*

by (*metis* (*full-types*) *eval-e.intros*(1) *eval-v-litI*)

hence *eval-e* i ($CE\text{-val } (V\text{-var } x)$) (*SBool* ($n1 \leq n2$)) **using** *eval-c-elim*(7)[*OF* *]

by (*metis* *eval-e-elim*(1) *eval-v-elim*(1))

moreover have *eval-e* i ($CE\text{-op } LEq [(V\text{-lit } (L\text{-num } n1))]^{ce} [(V\text{-lit } (L\text{-num } n2))]^{ce}$) (*SBool* ($n1 \leq n2$))

using *eval-e-elim*(3) *eval-v-elim* *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 $\forall i s1 s2. wfG P \mathcal{B} GNil \wedge wfI P GNil i \wedge eval\text{-e } i e1 s1 \wedge eval\text{-e } i e2 s2 \longrightarrow s1 = s2$

and $wfCE P \mathcal{B} GNil e1 b$ **and** $wfCE P \mathcal{B} GNil e2 b$

shows $P ; \mathcal{B} ; (x, b, CE\text{-val } (V\text{-var } x) == e1) \#_{\Gamma} GNil \models CE\text{-val } (V\text{-var } x) == e2$

unfolding *valid.simps*

proof(*intro conjI*)

show $\langle P ; \mathcal{B} ; (x, b, [[x]^v]^{ce} == e1) \#_{\Gamma} GNil \vdash_{wf} [[x]^v]^{ce} == e2 \rangle$

using *assms wf-intros wfX-wfY b.eq-iff fresh-GNil wfC-e-eq2 wfV-elim* **by** *meson*

show $\langle \forall i. ((P ; (x, b, [[x]^v]^{ce} == e1) \#_{\Gamma} GNil \vdash i) \wedge (i \models (x, b, [[x]^v]^{ce} == e1) \#_{\Gamma} GNil)) \longrightarrow$

$(i \models [[x]^v]^{ce} == e2) \rangle$ **proof**(*rule+*)

fix i

assume $as:P ; (x, b, [[x]^v]^{ce} == e1) \#_{\Gamma} GNil \vdash i \wedge i \models (x, b, [[x]^v]^{ce} == e1) \#_{\Gamma} GNil$

have *: $P ; GNil \vdash i$ **using** *wfI-def* **by** *auto*

then obtain $s1$ **where** $s1:eval\text{-e } i e1 s1$ **using** *assms eval-e-exist* **by** *metis*

obtain $s2$ **where** $s2:eval\text{-e } i e2 s2$ **using** *assms eval-e-exist* * **by** *metis*

moreover have $i x = \text{Some } s1$ **proof** –

have $i \models [[x]^v]^{ce} == e1$ **using** *as is-satis-g.simps* **by** *auto*

thus ?*thesis* **using** $s1$

by (*metis* *eval-c-elim*(7) *eval-e-elim*(1) *eval-e-uniqueness* *eval-v-elim*(2) *is-satis.cases*)

qed

moreover have $s1 = s2$ **using** $s1 s2$ * *assms wfG-nilI wfX-wfY* **by** *metis*

ultimately show $i \ll [[x]^v]^{ce} == e2 \gg \sim \text{True}$

using *eval-c.intros eval-e.intros eval-v.intros*

proof –

have $i \ll e2 \gg \sim s1$

by (*metis* $\langle s1 = s2 \rangle s2$)

then show ?*thesis*

by (*metis* (*full-types*) $\langle i x = \text{Some } s1 \rangle$ *eval-c-eqI eval-e-valI eval-v-varI*)

qed

qed

qed

lemma valid-len:

assumes $\vdash_{wf} \Theta$

shows $\Theta ; \mathcal{B} ; (x, B\text{-int}, [[x]^v]^{ce} == [[L\text{-num} (int (length v))]^v]^{ce}) \#_{\Gamma} GNil \models [[x]^v]^{ce} == CE\text{-len} [[L\text{-bitvec } v]^v]^{ce} \text{ (is } \Theta ; \mathcal{B} ; ?G \models ?c \text{)}$

proof –

have $*:\Theta \vdash_{wf} ([::\Phi) \wedge \Theta ; \mathcal{B} ; GNil \vdash_{wf} []_{\Delta}$ **using** *assms wfG-nilI wfD-emptyI wfPhi-emptyI* **by** *auto*

moreover **hence** $\Theta ; \mathcal{B} ; GNil \vdash_{wf} CE\text{-val} (V\text{-lit} (L\text{-num} (int (length v)))) : B\text{-int}$
using *wfCE-valI * wfV-litI base-for-lit.simps*
by (*metis wfE-valI wfX-wfY*)

moreover **have** $\Theta ; \mathcal{B} ; GNil \vdash_{wf} CE\text{-len} [(V\text{-lit} (L\text{-bitvec } v))]^{ce} : B\text{-int}$
using *wfE-valI * wfV-litI base-for-lit.simps wfE-valI wfX-wfY wfCE-valI*
by (*metis wfCE-lenI*)

moreover **have** $atom\ x \# GNil$ **by** *auto*

ultimately **have** $\Theta ; \mathcal{B} ; ?G \vdash_{wf} ?c$ **using** *wfC-e-eq2 assms* **by** *simp*

moreover **have** $(\forall i. wfI\ \Theta\ ?G\ i \wedge is\text{-satis-}g\ i\ ?G \longrightarrow is\text{-satis}\ i\ ?c)$ **using** *is-satis-len-imp* **by** *auto*

ultimately **show** *?thesis* **using** *valid.simps* **by** *auto*

qed

lemma valid-bop:

assumes *wfG* $\Theta\ \mathcal{B}\ \Gamma$ **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 $(opp = Plus \longrightarrow b = B\text{-int}) \wedge (opp = LEq \longrightarrow b = B\text{-bool})$ **and**

$atom\ x \# \Gamma$

shows $\Theta ; \mathcal{B} ; (x, b, (CE\text{-val} (V\text{-var } x) == CE\text{-val} (V\text{-lit} (ll))) \#_{\Gamma} \Gamma$
 $\models (CE\text{-val} (V\text{-var } x) == CE\text{-op } opp\ ([V\text{-lit} (L\text{-num } n1)]^{ce})\ ([V\text{-lit} (L\text{-num } n2)]^{ce}))$
 $\text{ (is } \Theta ; \mathcal{B} ; ?G \models ?c \text{)}$

proof –

have *wfC* $\Theta\ \mathcal{B}\ ?G\ ?c$ **proof**(*rule wfC-e-eq2*)

show $\Theta ; \mathcal{B} ; \Gamma \vdash_{wf} CE\text{-val} (V\text{-lit } ll) : b$ **using** *wfCE-valI wfV-litI assms base-for-lit.simps* **by** *metis*

show $\Theta ; \mathcal{B} ; \Gamma \vdash_{wf} CE\text{-op } opp\ ([V\text{-lit} (L\text{-num } n1)]^{ce})\ ([V\text{-lit} (L\text{-num } n2)]^{ce}) : b$

using *wfCE-plusI wfCE-leqI wfV-litI wfCE-valI base-for-lit.simps assms* **by** *metis*

show $\vdash_{wf} \Theta$ **using** *assms wfX-wfY* **by** *auto*

show $atom\ x \# \Gamma$ **using** *assms* **by** *auto*

qed

moreover **have** $\forall i. wfI\ \Theta\ ?G\ i \wedge is\text{-satis-}g\ i\ ?G \longrightarrow is\text{-satis}\ i\ ?c$ **proof**(*rule allI , rule impI*)

fix *i*

assume $wfI\ \Theta\ ?G\ i \wedge is\text{-satis-}g\ i\ ?G$

hence $is\text{-satis}\ i\ ((CE\text{-val} (V\text{-var } x) == CE\text{-val} (V\text{-lit} (ll)))$ **by** *auto*

thus $is\text{-satis}\ i\ ((CE\text{-val} (V\text{-var } x) == CE\text{-op } opp\ ([V\text{-lit} (L\text{-num } n1)]^{ce})\ ([V\text{-lit} (L\text{-num } n2)]^{ce})))$

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 $wfTh \ \Theta$ and $wfV \ \Theta \ \mathcal{B} \ GNil \ (V\text{-pair } v_1 \ v_2) \ (B\text{-pair } b_1 \ b_2)$
 shows $\Theta ; \mathcal{B} ; (x, b_1, [[x]^v]^{ce} == [v_1]^{ce}) \#_{\Gamma} GNil \models [[x]^v]^{ce} == [\#1[[v_1, v_2]^v]^{ce}]^{ce}$
 proof(*rule valid-eq-e*)
 show $\langle \forall i \ s1 \ s2. (\Theta ; \mathcal{B} \vdash_{wf} GNil) \wedge (\Theta ; GNil \vdash i) \wedge (i \llbracket [v_1]^{ce} \rrbracket \sim s1) \wedge (i \llbracket [\#1[[v_1, v_2]^v]^{ce}]^{ce} \rrbracket \sim s2) \longrightarrow s1 = s2 \rangle$
 proof(*rule+*)
 fix $i \ s1 \ s2$
 assume $as:\Theta ; \mathcal{B} \vdash_{wf} GNil \wedge \Theta ; GNil \vdash i \wedge (i \llbracket [v_1]^{ce} \rrbracket \sim s1) \wedge (i \llbracket [\#1[[v_1, v_2]^v]^{ce}]^{ce} \rrbracket \sim s2)$
 then obtain $s2'$ where $*:i \llbracket [v_1, v_2]^v \rrbracket \sim SPair \ s2 \ s2'$
 using *eval-e-elim*(4)[*of* $i \llbracket [v_1, v_2]^v]^{ce} \ s2$] *eval-e-elim*
 by *meson*
 then have $i \llbracket [v_1] \rrbracket \sim s2$ using *eval-v-elim*(3)[*OF* $*$] by *auto*
 then show $s1 = s2$ using *eval-v-uniqueness* as
 using *eval-e-uniqueness* *eval-e-valI* by *blast*
 qed

show $\langle \Theta ; \mathcal{B} ; GNil \vdash_{wf} [v_1]^{ce} : b_1 \rangle$ using *assms*
 by (*metis* *b.eq-iff*(4) *wfV-elim*(3) *wfV-wfCE*)
 show $\langle \Theta ; \mathcal{B} ; GNil \vdash_{wf} [\#1[[v_1, v_2]^v]^{ce}]^{ce} : b_1 \rangle$ 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 $wfTh \ \Theta$ and $wfV \ \Theta \ \mathcal{B} \ GNil \ (V\text{-pair } v_1 \ v_2) \ (B\text{-pair } b_1 \ b_2)$
 shows $\Theta ; \mathcal{B} ; (x, b_2, [[x]^v]^{ce} == [v_2]^{ce}) \#_{\Gamma} GNil \models [[x]^v]^{ce} == [\#2[[v_1, v_2]^v]^{ce}]^{ce}$
 proof(*rule valid-eq-e*)
 show $\langle \forall i \ s1 \ s2. (\Theta ; \mathcal{B} \vdash_{wf} GNil) \wedge (\Theta ; GNil \vdash i) \wedge (i \llbracket [v_2]^{ce} \rrbracket \sim s1) \wedge (i \llbracket [\#2[[v_1, v_2]^v]^{ce}]^{ce} \rrbracket \sim s2) \longrightarrow s1 = s2 \rangle$
 proof(*rule+*)
 fix $i \ s1 \ s2$
 assume $as:\Theta ; \mathcal{B} \vdash_{wf} GNil \wedge \Theta ; GNil \vdash i \wedge (i \llbracket [v_2]^{ce} \rrbracket \sim s1) \wedge (i \llbracket [\#2[[v_1, v_2]^v]^{ce}]^{ce} \rrbracket \sim s2)$
 then obtain $s2'$ where $*:i \llbracket [v_1, v_2]^v \rrbracket \sim SPair \ s2' \ s2$
 using *eval-e-elim*(4)[*of* $i \llbracket [v_1, v_2]^v]^{ce} \ s2$] *eval-e-elim*
 by *meson*
 then have $i \llbracket [v_2] \rrbracket \sim s2$ using *eval-v-elim*(3)[*OF* $*$] by *auto*
 then show $s1 = s2$ using *eval-v-uniqueness* as
 using *eval-e-uniqueness* *eval-e-valI* by *blast*
 qed

show $\langle \Theta ; \mathcal{B} ; GNil \vdash_{wf} [v_2]^{ce} : b_2 \rangle$ using *assms*
 by (*metis* *b.eq-iff* *wfV-elim* *wfV-wfCE*)
 show $\langle \Theta ; \mathcal{B} ; GNil \vdash_{wf} [\#2[[v_1, v_2]^v]^{ce}]^{ce} : b_2 \rangle$ using *assms* using *wfCE-sndI* *wfCE-valI* by *blast*

qed

lemma valid-concat:

fixes $v1::bit\ list$ **and** $v2::bit\ list$

assumes $\vdash_{wf} \Pi$

shows $\Pi ; \mathcal{B} ; (x, B-bitvec, (CE-val (V-var x) == CE-val (V-lit (L-bitvec (v1 @ v2)))) \#_{\Gamma} GNil \models$
 $(CE-val (V-var x) == CE-concat ([V-lit (L-bitvec v1)]^{ce}) ([V-lit (L-bitvec v2)]^{ce}))$

proof(rule valid-eq-e)

show $\langle \forall i\ s1\ s2. ((\Pi ; \mathcal{B} \vdash_{wf} GNil) \wedge (\Pi ; GNil \vdash i) \wedge$
 $(i \llbracket [[L-bitvec (v1 @ v2)]^v]^{ce} \rrbracket \sim s1) \wedge (i \llbracket [[[L-bitvec v1]^v]^{ce} @ [[L-bitvec v2]^v]^{ce}]^{ce} \rrbracket \sim s2) \rangle \longrightarrow$
 $s1 = s2 \rangle$

proof(rule+)

fix $i\ s1\ s2$

assume $as: (\Pi ; \mathcal{B} \vdash_{wf} GNil) \wedge (\Pi ; GNil \vdash i) \wedge (i \llbracket [[L-bitvec (v1 @ v2)]^v]^{ce} \rrbracket \sim s1) \wedge$
 $(i \llbracket [[[L-bitvec v1]^v]^{ce} @ [[L-bitvec v2]^v]^{ce}]^{ce} \rrbracket \sim s2)$

hence $*$: $i \llbracket [[[L-bitvec v1]^v]^{ce} @ [[L-bitvec v2]^v]^{ce}]^{ce} \rrbracket \sim s2$ **by** *auto*

obtain $bv1\ bv2$ **where** $s2:s2 = SBitvec (bv1 @ bv2) \wedge i \llbracket [L-bitvec v1]^v \rrbracket \sim SBitvec\ bv1 \wedge (i \llbracket [L-bitvec\ v2]^v \rrbracket \sim SBitvec\ bv2)$

using *eval-e-elim*(6)[OF *] *eval-e-elim*(1) **by** *metis*

hence $v1 = bv1 \wedge v2 = bv2$ **using** *eval-v-elim*(1) *eval-l.simp*(5) **by** *force*

moreover then have $s1 = SBitvec (bv1 @ bv2)$ **using** $s2$ **using** *eval-v-elim*(1) *eval-l.simp*(5)

by (*metis as eval-e-elim*(1))

then show $s1 = s2$ **using** $s2$ **by** *auto*

qed

show $\langle \Pi ; \mathcal{B} ; GNil \vdash_{wf} [[L-bitvec (v1 @ v2)]^v]^{ce} : B-bitvec \rangle$

by (*metis assms base-for-lit.simp*(5) *wfG-nilI wfV-litI wfV-wfCE*)

show $\langle \Pi ; \mathcal{B} ; GNil \vdash_{wf} [[[L-bitvec v1]^v]^{ce} @ [[L-bitvec v2]^v]^{ce}]^{ce} : B-bitvec \rangle$

by (*metis assms base-for-lit.simp*(5) *wfCE-concatI wfG-nilI wfV-litI wfCE-valI*)

qed

lemma valid-ce-eq:

fixes $ce::ce$

assumes $\Theta ; \mathcal{B} ; \Gamma \vdash_{wf} ce : b$

shows $\langle \Theta ; \mathcal{B} ; \Gamma \models ce == ce \rangle$

unfolding *valid.simp* **proof**

show $\langle \Theta ; \mathcal{B} ; \Gamma \vdash_{wf} ce == ce \rangle$ **using** *assms wfC-eqI* **by** *auto*

show $\langle \forall i. \Theta ; \Gamma \vdash i \wedge i \models \Gamma \longrightarrow i \models ce == ce \rangle$ **proof**(rule+)

fix i

assume $\Theta ; \Gamma \vdash i \wedge i \models \Gamma$

then obtain s **where** $i \llbracket ce \rrbracket \sim s$ **using** *assms eval-e-exist* **by** *metis*

then show $i \llbracket ce == ce \rrbracket \sim True$ **using** *eval-c-eqI* **by** *metis*

qed

qed

lemma valid-eq-imp:

fixes $c1::c$ **and** $c2::c$

assumes $\Theta ; \mathcal{B} ; (x, b, c2) \#_{\Gamma} \Gamma \vdash_{wf} c1\ IMP\ c2$

```

shows  $\Theta ; \mathcal{B} ; (x, b, c2) \#_{\Gamma} \Gamma \models c1 \text{ IMP } c2$ 
proof –
  have  $\forall i. (\Theta ; (x, b, c2) \#_{\Gamma} \Gamma \vdash i \wedge i \models (x, b, c2) \#_{\Gamma} \Gamma) \longrightarrow i \models (c1 \text{ IMP } c2)$ 
  proof(rule,rule)
    fix  $i$ 
    assume  $as:\Theta ; (x, b, c2) \#_{\Gamma} \Gamma \vdash i \wedge i \models (x, b, c2) \#_{\Gamma} \Gamma$ 

    have  $\Theta ; \mathcal{B} ; (x, b, c2) \#_{\Gamma} \Gamma \vdash_{wf} c1$  using wfC-elim assms by metis

    then obtain  $sc$  where  $i \llbracket c1 \rrbracket \sim sc$  using eval-c-exist assms as by metis
    moreover have  $i \llbracket c2 \rrbracket \sim True$  using as is-satis-g.simps is-satis.simps by auto

    ultimately have  $i \llbracket c1 \text{ IMP } c2 \rrbracket \sim True$  using eval-c-impI by metis

    thus  $i \models c1 \text{ IMP } c2$  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  $\vdash_{wf} \Theta$ 
  shows  $\Theta ; \{\llbracket \rrbracket\} ; (x, B\text{-int}, (C\text{-eq} (CE\text{-val} (V\text{-var } x)) (CE\text{-val} (V\text{-lit} (L\text{-num } n)))) \#_{\Gamma} GNil \models$ 
 $(C\text{-eq} (CE\text{-op } LEq (CE\text{-val} (V\text{-var } x)) (CE\text{-val} (V\text{-lit} (L\text{-num } m)))) \llbracket L\text{-true}$ 
 $\rrbracket^v ]^{ce}) \text{ AND}$ 
 $(C\text{-eq} (CE\text{-op } LEq (CE\text{-val} (V\text{-lit} (L\text{-num } 0))) (CE\text{-val} (V\text{-var } x))) \llbracket L\text{-true} \rrbracket^v$ 
 $\rrbracket^{ce})$ 
  (is  $\Theta ; \{\llbracket \rrbracket\} ; ?G \models ?c1 \text{ AND } ?c2$ )
proof(rule validI)
  have  $wfg: \Theta ; \{\llbracket \rrbracket\} \vdash_{wf} (x, B\text{-int}, \llbracket x \rrbracket^v ]^{ce} == \llbracket L\text{-num } n \rrbracket^v ]^{ce}) \#_{\Gamma} GNil$ 
  using assms base-for-lit.simps wfG-nilI wfV-litI fresh-GNil wfB-intI wfC-v-eq wfG-cons1I wfG-cons2I
by metis

  show  $\Theta ; \{\llbracket \rrbracket\} ; ?G \vdash_{wf} ?c1 \text{ AND } ?c2$ 
  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  $\forall i. \Theta ; ?G \vdash i \wedge i \models ?G \longrightarrow i \models ?c1 \text{ AND } ?c2$  proof(rule,rule)
  fix  $i$ 
  assume  $a:\Theta ; ?G \vdash i \wedge i \models ?G$ 
  hence  $*:i \llbracket V\text{-var } x \rrbracket \sim SNum\ n$ 
  proof –
    obtain  $sv$  where  $sv: i\ x = \text{Some } sv \wedge \Theta \vdash sv : B\text{-int}$  using a wfI-def by force
    have  $i \llbracket (C\text{-eq} (CE\text{-val} (V\text{-var } x)) (CE\text{-val} (V\text{-lit} (L\text{-num } n)))) \rrbracket \sim True$ 
    using a is-satis-g.simps
    using is-satis.cases by blast
    hence  $i\ x = \text{Some}(SNum\ n)$  using  $sv$ 
    by (metis eval-c-elim(7) eval-e-elim(1) eval-l.simps(3) eval-v-elim(1) eval-v-elim(2))
    thus ?thesis using eval-v-varI by auto
  qed

  show  $i \models ?c1 \text{ AND } ?c2$ 

```



```

proof –
  have  $i \llbracket ?c1 \rrbracket \sim \text{True}$ 
proof –
  have  $i \llbracket \text{leq} \llbracket [x]^v \rrbracket^{ce} \llbracket [L\text{-num } m]^v \rrbracket^{ce} \rrbracket \sim \text{SBool True}$ 
    using eval-e-leqI assms eval-v-litI eval-l.simps *
    by (metis (full-types) eval-e-valI)
  moreover have  $i \llbracket \llbracket [L\text{-true}]^v \rrbracket^{ce} \rrbracket \sim \text{SBool True}$ 
    using eval-v-litI eval-e-valI eval-l.simps by metis
  ultimately show ?thesis using eval-c-eqI by metis
qed

moreover have  $i \llbracket ?c2 \rrbracket \sim \text{True}$ 
proof –
  have  $i \llbracket \text{leq} \llbracket [L\text{-num } 0]^v \rrbracket^{ce} \llbracket [x]^v \rrbracket^{ce} \rrbracket \sim \text{SBool True}$ 
    using eval-e-leqI assms eval-v-litI eval-l.simps *
    by (metis (full-types) eval-e-valI)
  moreover have  $i \llbracket \llbracket [L\text{-true}]^v \rrbracket^{ce} \rrbracket \sim \text{SBool True}$ 
    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  $\Gamma :: \Gamma$ 
  assumes  $0 \leq n \wedge n \leq \text{int}(\text{length } v)$  and  $\Theta ; \{\llbracket \rrbracket \vdash_{wf} \Gamma$  and  $\text{atom } x \nmid \Gamma$ 
  shows  $\Theta ; \{\llbracket \rrbracket ; (x, B\text{-int}, (C\text{-eq} (CE\text{-val} (V\text{-var } x)) (CE\text{-val} (V\text{-lit} (L\text{-num } n)))) \#_{\Gamma} \Gamma \models$ 
     $(C\text{-eq} (CE\text{-op } LEq (CE\text{-val} (V\text{-lit} (L\text{-num } 0))) (CE\text{-val} (V\text{-var } x))) \llbracket [L\text{-true}]^v \rrbracket^{ce}$ 
AND
     $(C\text{-eq} (CE\text{-op } LEq (CE\text{-val} (V\text{-var } x)) (\llbracket [ [L\text{-bitvec } v]^v \rrbracket^{ce} \rrbracket^{ce} )) \llbracket [L\text{-true}]^v \rrbracket^{ce})$ 
     $(\text{is } \Theta ; \{\llbracket \rrbracket ; ?G \models ?c1 \text{ AND } ?c2)$ 
proof(rule validI)
  have  $wfg : \Theta ; \{\llbracket \rrbracket \vdash_{wf} (x, B\text{-int}, \llbracket [x]^v \rrbracket^{ce} == \llbracket [L\text{-num } n]^v \rrbracket^{ce}) \#_{\Gamma} \Gamma$  apply(rule wfg-consII)
    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  $\Theta ; \{\llbracket \rrbracket ; ?G \vdash_{wf} ?c1 \text{ AND } ?c2$ 
    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  $\forall i. \Theta ; ?G \vdash i \wedge i \models ?G \longrightarrow i \models ?c1 \text{ AND } ?c2$  proof(rule,rule)
    fix  $i$ 
    assume  $a:\Theta ; ?G \vdash i \wedge i \models ?G$ 
    hence  $*:i \llbracket V\text{-var } x \rrbracket \sim \text{SNum } n$ 
    proof –

```

```

obtain sv where sv:  $i\ x = \text{Some } sv \wedge \Theta \vdash sv : B\text{-int}$  using a wfl-def by force
have  $i \llbracket (C\text{-eq } (CE\text{-val } (V\text{-var } x)) (CE\text{-val } (V\text{-lit } (L\text{-num } n)))) \rrbracket \sim \text{True}$ 
  using a is-satis-g.simps
  using is-satis.cases by blast
hence  $i\ x = \text{Some}(SNum\ n)$  using sv
  by (metis eval-c-elim(7) eval-e-elim(1) eval-l.simps(3) eval-v-elim(1) eval-v-elim(2))
thus ?thesis using eval-v-varI by auto
qed

show  $i \models ?c1 \text{ AND } ?c2$ 
proof –
  have  $i \llbracket ?c2 \rrbracket \sim \text{True}$ 
  proof –
    have  $i \llbracket \llbracket \text{leq } [\llbracket x \rrbracket^v]^{ce} \llbracket [\llbracket L\text{-bitvec } v \rrbracket^v]^{ce} \rrbracket^{ce} \rrbracket \sim SBool\ \text{True}$ 
      using eval-e-leqI assms eval-v-litI eval-l.simps *
      by (metis (full-types) eval-e-lenI eval-e-valI)
    moreover have  $i \llbracket [\llbracket L\text{-true} \rrbracket^v]^{ce} \rrbracket \sim SBool\ \text{True}$ 
      using eval-v-litI eval-e-valI eval-l.simps by metis
    ultimately show ?thesis using eval-c-eqI by metis
  qed

  moreover have  $i \llbracket ?c1 \rrbracket \sim \text{True}$ 
  proof –
    have  $i \llbracket \llbracket \text{leq } [\llbracket L\text{-num } 0 \rrbracket^v]^{ce} [\llbracket x \rrbracket^v]^{ce} \rrbracket \rrbracket \sim SBool\ \text{True}$ 
      using eval-e-leqI assms eval-v-litI eval-l.simps *
      by (metis (full-types) eval-e-valI)
    moreover have  $i \llbracket [\llbracket L\text{-true} \rrbracket^v]^{ce} \rrbracket \sim SBool\ \text{True}$ 
      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  $\Gamma::\Gamma$ 
  assumes  $\vdash_{wf} \Theta$ 
  and  $\Theta ; \{\llbracket \rrbracket ; (x, B\text{-int} , (C\text{-eq } (CE\text{-val } (V\text{-var } x)) (CE\text{-val } (V\text{-lit } (L\text{-num } n)))) \#_{\Gamma} GNil \models$ 
     $(C\text{-eq } (CE\text{-op } LEq (CE\text{-val } (V\text{-lit } (L\text{-num } 0))) (CE\text{-val } (V\text{-var } x))) \llbracket \llbracket L\text{-true} \rrbracket^v \rrbracket^{ce}$ 
  AND
     $(C\text{-eq } (CE\text{-op } LEq (CE\text{-val } (V\text{-var } x)) (\llbracket \llbracket L\text{-bitvec } v \rrbracket^v \rrbracket^{ce} \rrbracket^{ce})) \llbracket \llbracket L\text{-true} \rrbracket^v \rrbracket^{ce}$ 
     $(\text{is } \Theta ; \{\llbracket \rrbracket ; ?G \models ?c1 \text{ AND } ?c2)$ 
  shows  $0 \leq n \wedge n \leq \text{int } (\text{length } v)$ 
proof –
  have  $\ast:\forall i. \Theta ; ?G \vdash i \wedge i \models ?G \longrightarrow i \models ?c1 \text{ AND } ?c2$  using assms valid.simps by simp

  obtain i where  $i: i\ x = \text{Some } (SNum\ n)$  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  $i \llbracket ([x]^v)^{ce} == [L\text{-num } n]^v \rrbracket^{ce} \rrbracket \sim \text{True}$ 
  using * eval-c.intros( $\gamma$ ) 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  $**i \models ?c1 \text{ AND } ?c2$  using * by auto

hence  $1: i \llbracket ?c1 \rrbracket \sim \text{True}$  using eval-c.elims( $\beta$ ) is-satis.simps
  by fastforce
then obtain  $sv1$  and  $sv2$  where  $(sv1 = sv2) = \text{True} \wedge i \llbracket \text{leq } [L\text{-num } 0]^v \rrbracket^{ce} [x]^v \rrbracket^{ce} \rrbracket$ 
 $\sim sv1 \wedge i \llbracket [L\text{-true}]^v \rrbracket^{ce} \rrbracket \sim sv2$ 
  using eval-c.elims( $\gamma$ ) by metis
hence  $sv1 = \text{SBool True}$  using eval-e.elims eval-v.elims eval-l.simps  $i$  by metis
obtain  $n1$  and  $n2$  where  $\text{SBool True} = \text{SBool } (n1 \leq n2) \wedge (i \llbracket [L\text{-num } 0]^v \rrbracket^{ce} \rrbracket \sim \text{SNum } n1)$ 
 $\wedge (i \llbracket [x]^v \rrbracket^{ce} \rrbracket \sim \text{SNum } n2)$ 
  using eval-e.elims( $\beta$ )[of  $i \llbracket [L\text{-num } 0]^v \rrbracket^{ce} [x]^v \rrbracket^{ce} \text{SBool True}$ ]
  using  $\langle (sv1 = sv2) = \text{True} \wedge i \llbracket \text{leq } [L\text{-num } 0]^v \rrbracket^{ce} [x]^v \rrbracket^{ce} \rrbracket \sim sv1 \wedge i \llbracket [L\text{-true}]^v \rrbracket^{ce} \rrbracket \sim sv2 \rangle$ 
 $\langle sv1 = \text{SBool True} \rangle$  by fastforce
moreover hence  $n1 = 0$  and  $n2 = n$  using eval-e.elims eval-v.elims  $i$ 
  apply (metis eval-l.simps( $\beta$ ) 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  $2: i \llbracket ?c2 \rrbracket \sim \text{True}$  using ** eval-c.elims( $\beta$ ) is-satis.simps
  by fastforce
then obtain  $sv1$  and  $sv2$  where  $sv: (sv1 = sv2) = \text{True} \wedge i \llbracket \text{leq } [x]^v \rrbracket^{ce} \llbracket [L\text{-bitvec } v]^v \rrbracket^{ce} \rrbracket^{ce} \rrbracket$ 
 $\sim sv1 \wedge i \llbracket [L\text{-true}]^v \rrbracket^{ce} \rrbracket \sim sv2$ 
  using eval-c.elims( $\gamma$ ) by metis
hence  $sv1 = \text{SBool True}$  using eval-e.elims eval-v.elims eval-l.simps  $i$  by metis
obtain  $n1$  and  $n2$  where  $***\text{SBool True} = \text{SBool } (n1 \leq n2) \wedge (i \llbracket [x]^v \rrbracket^{ce} \rrbracket \sim \text{SNum } n1) \wedge (i$ 
 $\llbracket \llbracket [L\text{-bitvec } v]^v \rrbracket^{ce} \rrbracket^{ce} \rrbracket \sim \text{SNum } n2)$ 
  using eval-e.elims( $\beta$ )
  using  $sv \langle sv1 = \text{SBool True} \rangle$  by metis
moreover hence  $n1 = n$  using eval-e.elims( $1$ )[of  $i$ ] eval-v.elims( $2$ )[of  $i$   $x$   $\text{SNum } n1$ ]  $i$  by auto
moreover have  $n2 = \text{int } (\text{length } v)$  using eval-e.elims( $\gamma$ ) 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  $le2: n \leq \text{int } (\text{length } v)$  by simp

show ?thesis using  $le1$   $le2$  by auto
qed

thm wfI-def

lemma wfI-cons:
  fixes  $i::\text{valuation}$  and  $\Gamma::\Gamma$ 
  assumes  $i' \models \Gamma$  and  $\Theta ; \Gamma \vdash i'$  and  $i = i' (x \mapsto s)$  and  $\Theta \vdash s : b$  and  $\text{atom } x \nmid \Gamma$ 
  shows  $\Theta ; (x, b, c) \#_{\Gamma} \Gamma \vdash i$ 
unfolding wfI-def proof -
  {
    fix  $x' b' c'$ 

```

```

assume  $(x', b', c') \in \text{toSet } ((x, b, c) \#_{\Gamma} \Gamma)$ 
then consider  $(x', b', c') = (x, b, c) \mid (x', b', c') \in \text{toSet } \Gamma$  using toSet.simps by auto
then have  $\exists s. \text{Some } s = i \ x' \wedge \Theta \vdash s : b'$  proof(cases)
  case 1
    then show ?thesis using assms by auto
  next
    case 2
      then obtain  $s$  where  $s : \text{Some } s = i \ x' \wedge \Theta \vdash s : b'$  using assms wfl-def by auto
      moreover have  $x' \neq x$  using assms 2 fresh-dom-free by auto
      ultimately have  $\text{Some } s = i \ x'$  using assms by auto
      then show ?thesis using  $s$  wfl-def by auto
    qed
  }
thus  $\forall (x, b, c) \in \text{toSet } ((x, b, c) \#_{\Gamma} \Gamma). \exists s. \text{Some } s = i \ x \wedge \Theta \vdash s : b$  by auto
qed

lemma valid-range-length-inv:
  fixes  $\Gamma :: \Gamma$ 
  assumes  $\Theta ; B \vdash_{wf} \Gamma$  and atom  $x \# \Gamma$  and  $\exists i. i \models \Gamma \wedge \Theta ; \Gamma \vdash i$ 
  and  $\Theta ; B ; (x, B\text{-int}, (C\text{-eq } (CE\text{-val } (V\text{-var } x)) (CE\text{-val } (V\text{-lit } (L\text{-num } n)))) \#_{\Gamma} \Gamma \models$ 
     $(C\text{-eq } (CE\text{-op } LEq (CE\text{-val } (V\text{-lit } (L\text{-num } 0))) (CE\text{-val } (V\text{-var } x))) \llbracket L\text{-true} \rrbracket^v \rrbracket^{ce}$ 
  AND
     $(C\text{-eq } (CE\text{-op } LEq (CE\text{-val } (V\text{-var } x)) (\llbracket [ [ L\text{-bitvec } v ]^v \rrbracket^{ce} \rrbracket^{ce} )) \llbracket L\text{-true} \rrbracket^v \rrbracket^{ce}$ 

    (is  $\Theta ; ?B ; ?G \models ?c1$  AND  $?c2$  )
    shows  $0 \leq n \wedge n \leq \text{int } (\text{length } v)$ 
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 idash:  $is\text{-satis-g } i' \Gamma \wedge \Theta ; \Gamma \vdash i'$  using assms by auto
  obtain  $i$  where  $i : i = i' (x \mapsto SNum\ n)$  by auto
  hence  $ix : i \ x = \text{Some } (SNum\ n)$  by auto
  have  $\Theta ; ?G \vdash i \wedge i \models ?G$  proof
    show  $\Theta ; ?G \vdash i$  using wfl-cons i idash ix wflRCV-BIntI assms by simp

    have  $** : i \llbracket ([ [ x ]^v \rrbracket^{ce} == [ [ L\text{-num } n ]^v \rrbracket^{ce} ]) \rrbracket \sim \text{True}$ 
      using  $* \text{eval-c.intros}(7) \text{eval-e.intros eval-v.intros eval-l.simps } i$ 
      by (metis (full-types) ix)

    show  $i \models ?G$  unfolding is-satis-g.simps proof
      show  $\langle i \models [ [ x ]^v \rrbracket^{ce} == [ [ L\text{-num } n ]^v \rrbracket^{ce} \rangle$  using  $** \text{is-satis.simps}$  by auto
      show  $\langle i \models \Gamma \rangle$  using idash i assms is-satis-g-i-upd by metis
    qed
  qed
  hence  $** : i \models ?c1$  AND  $?c2$  using  $*$  by auto

  hence  $1 : i \llbracket ?c1 \rrbracket \sim \text{True}$  using eval-c.elims(3) is-satis.simps
    by fastforce
  then obtain  $sv1$  and  $sv2$  where  $(sv1 = sv2) = \text{True} \wedge i \llbracket \text{leq } [ [ L\text{-num } 0 ]^v \rrbracket^{ce} [ [ x ]^v \rrbracket^{ce} ] \rrbracket$ 
 $\sim sv1 \wedge i \llbracket [ [ L\text{-true} ]^v \rrbracket^{ce} \rrbracket \sim sv2$ 
    using eval-c.elims(7) by metis

```

hence $sv1 = SBool\ True$ **using** $eval-e-elim\ eval-v-elim\ eval-l.simps\ i$ **by** $metis$
obtain $n1$ **and** $n2$ **where** $SBool\ True = SBool\ (n1 \leq n2) \wedge (i \llbracket [[L-num\ 0]^v]^{ce} \rrbracket \sim SNum\ n1)$
 $\wedge (i \llbracket [[x]^v]^{ce} \rrbracket \sim SNum\ n2)$
using $eval-e-elim(3)[of\ i\ [[L-num\ 0]^v]^{ce}\ [[x]^v]^{ce}\ SBool\ True]$
using $\langle sv1 = sv2 \rangle = True \wedge i \llbracket [leq\ [[L-num\ 0]^v]^{ce}\ [[x]^v]^{ce}]^{ce} \rrbracket \sim sv1 \wedge i \llbracket [[L-true]^v]^{ce} \rrbracket \sim sv2 \rangle \langle sv1 = SBool\ True \rangle$ **by** $fastforce$
moreover **hence** $n1 = 0$ **and** $n2 = n$ **using** $eval-e-elim\ eval-v-elim\ i$
apply $(metis\ eval-l.simps(3)\ rcl-val.eq-iff(2))$
using $eval-e-elim\ eval-v-elim\ i$
 $calculation\ option.inject\ rcl-val.eq-iff(2)$
by $(metis\ ix)$
ultimately **have** $le1: 0 \leq n$ **by** $simp$

hence $2: i \llbracket ?c2 \rrbracket \sim True$ **using** $**\ eval-c-elim(3)\ is-satis.simps$
by $fastforce$
then **obtain** $sv1$ **and** $sv2$ **where** $sv: (sv1 = sv2) = True \wedge i \llbracket [leq\ [[x]^v]^{ce}\ [[L-bitvec\ v]^v]^{ce}]^{ce} \rrbracket \sim sv1 \wedge i \llbracket [[L-true]^v]^{ce} \rrbracket \sim sv2$
using $eval-c-elim(7)$ **by** $metis$
hence $sv1 = SBool\ True$ **using** $eval-e-elim\ eval-v-elim\ eval-l.simps\ i$ **by** $metis$
obtain $n1$ **and** $n2$ **where** $***: SBool\ True = SBool\ (n1 \leq n2) \wedge (i \llbracket [[x]^v]^{ce} \rrbracket \sim SNum\ n1) \wedge (i \llbracket [[L-bitvec\ v]^v]^{ce}]^{ce} \rrbracket \sim SNum\ n2)$
using $eval-e-elim(3)$
using $sv\ \langle sv1 = SBool\ True \rangle$ **by** $metis$
moreover **hence** $n1 = n$ **using** $eval-e-elim(1)[of\ i]\ eval-v-elim(2)[of\ i\ x\ SNum\ n1]\ i$ **by** $auto$
moreover **have** $n2 = int\ (length\ v)$ **using** $eval-e-elim(7)\ eval-v-elim(1)\ eval-l.simps\ i$
by $(metis\ ***\ eval-e-elim(1)\ rcl-val.eq-iff(1)\ rcl-val.eq-iff(2))$
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 $i \llbracket (C-conj\ c1\ c2) \rrbracket \sim True$
using $assms\ eval-c-conjI$ **by** $metis$

lemma $valid-split:$
assumes $split\ n\ v\ (v1, v2)$ **and** $\vdash_{wf}\ \Theta$
shows $\Theta ; \{|\}\ ; (z, [B-bitvec\ ,\ B-bitvec]^b, [[z]^v]^{ce} == [[[L-bitvec\ v1]^v, [L-bitvec\ v2]^v]^{ce}]_{\Gamma}\ GNil$
 $\models ([[L-bitvec\ v]^v]^{ce} == [[\#1[[z]^v]^{ce}]^{ce}\ @@\ [\#2[[z]^v]^{ce}]^{ce}]_{\Gamma}\ AND\ ([[\#1[[z]^v]^{ce}]^{ce}]_{\Gamma} == [[L-num\ n]^v]^{ce})$
 $(is\ \Theta ; \{|\}\ ;\ ?G \models ?c1\ AND\ ?c2)$
unfolding $valid.simps$ **proof**

have $wfg: \Theta ; \{|\}\ \vdash_{wf}\ (z, [B-bitvec\ ,\ B-bitvec]^b, [[z]^v]^{ce} == [[[L-bitvec\ v1]^v, [L-bitvec\ v2]^v]^{ce}]_{\Gamma}\ GNil$
using $wf-intros\ assms\ base-for-lit.simps\ fresh-GNil\ wfC-v-eq\ wfG-intros2$ **by** $metis$

```

show  $\Theta ; \{||\} ; ?G \vdash_{wf} ?c1 \text{ AND } ?c2$ 

  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  $\forall i. \Theta ; ?G \vdash i \wedge i \models ?G \longrightarrow i \models (?c1 \text{ AND } ?c2)$ 
proof(rule,rule)
  fix i
  assume a: $\Theta ; ?G \vdash i \wedge i \models ?G$ 
  hence i  $\llbracket [ [ z ]^v ]^{ce} \rrbracket == \llbracket [ [ L\text{-bitvec } v1 ]^v , [ L\text{-bitvec } v2 ]^v ]^v ]^{ce} \rrbracket \sim \text{True}$ 
    using is-satis-g.simps is-satis.simps by simp
  then obtain sv where i  $\llbracket [ [ z ]^v ]^{ce} \rrbracket \sim sv \wedge i \llbracket [ [ [ L\text{-bitvec } v1 ]^v , [ L\text{-bitvec } v2 ]^v ]^v ]^{ce} \rrbracket \sim sv$ 
    using eval-c-elim by metis
  hence i  $\llbracket [ [ z ]^v ]^{ce} \rrbracket \sim (SPair (SBitvec v1) (SBitvec v2))$  using eval-c-eqI eval-v.intros eval-l.simps

    by (metis eval-e-elim(1) eval-v-uniqueness)
  hence b: $i \ z = \text{Some } (SPair (SBitvec v1) (SBitvec v2))$  using a eval-e-elim eval-v-elim by metis

  have v1: i  $\llbracket [\#1 [ [ z ]^v ]^{ce} ]^{ce} \rrbracket \sim SBitvec v1$ 
    using eval-e-fstI eval-e-valI eval-v-varI b by metis
  have v2: i  $\llbracket [\#2 [ [ z ]^v ]^{ce} ]^{ce} \rrbracket \sim SBitvec v2$ 
    using eval-e-sndI eval-e-valI eval-v-varI b by metis

  have i  $\llbracket [ [ L\text{-bitvec } v ]^v ]^{ce} \rrbracket \sim SBitvec v$  using eval-e.intros eval-v.intros eval-l.simps by metis
  moreover have i  $\llbracket [ [\#1 [ [ z ]^v ]^{ce} ]^{ce} @@ [\#2 [ [ z ]^v ]^{ce} ]^{ce} ]^{ce} \rrbracket \sim SBitvec v$ 
    using assms split-concat v1 v2 eval-e-concatI by metis
  moreover have i  $\llbracket [ [\#1 [ [ z ]^v ]^{ce} ]^{ce} ]^{ce} \rrbracket \sim SNum (int (length v1))$ 
    using v1 eval-e-lenI by auto
  moreover have i  $\llbracket [ [ L\text{-num } n ]^v ]^{ce} \rrbracket \sim SNum n$  using eval-e.intros eval-v.intros eval-l.simps
by metis
  ultimately show  $i \models ?c1 \text{ AND } ?c2$  using is-satis.intros eval-c-conj2I eval-c-eqI len by metis
qed
qed

end

```

Chapter 12

Typing Lemmas

12.1 Subtyping

lemma *subtype-reflI2*:

fixes $\tau::\tau$

assumes $\Theta; \mathcal{B}; \Gamma \vdash_{wf} \tau$

shows $\Theta; \mathcal{B}; \Gamma \vdash \tau \lesssim \tau$

proof –

obtain $z\ b\ c$ **where** $*:\tau = \llbracket z : b \mid c \rrbracket \wedge \text{atom } z \# (\Theta, \mathcal{B}, \Gamma) \wedge \Theta; \mathcal{B}; (z, b, \text{TRUE}) \#_{\Gamma} \Gamma \vdash_{wf} c$
using *wfT-elim1* [*OF assms*] **by** *metis*

obtain $x::x$ **where** $**:\text{atom } x \# (\Theta, \mathcal{B}, \Gamma, c, z, c, z, c)$ **using** *obtain-fresh* **by** *metis*

have $\Theta; \mathcal{B}; \Gamma \vdash \llbracket z : b \mid c \rrbracket \lesssim \llbracket z : b \mid c \rrbracket$ **proof**

show $\Theta; \mathcal{B}; \Gamma \vdash_{wf} \llbracket z : b \mid c \rrbracket$ **using** $*$ *assms* **by** *auto*

show $\Theta; \mathcal{B}; \Gamma \vdash_{wf} \llbracket z : b \mid c \rrbracket$ **using** $*$ *assms* **by** *auto*

show $\text{atom } x \# (\Theta, \mathcal{B}, \Gamma, z, c, z, c)$ **using** *fresh-prod6* *fresh-prod5* $**$ **by** *metis*

thus $\Theta; \mathcal{B}; (x, b, c[z::=V\text{-var } x]_v) \#_{\Gamma} \Gamma \models c[z::=V\text{-var } x]_v$ **using** *wfT-wfC-cons* *assms* $*$ $**$

subst-v-c-def **by** *simp*

qed

thus *?thesis* **using** $*$ **by** *auto*

qed

lemma *subtype-reflI*:

assumes $\llbracket z1 : b \mid c1 \rrbracket = \llbracket z2 : b \mid c2 \rrbracket$ **and** $wf1: \Theta; \mathcal{B}; \Gamma \vdash_{wf} (\llbracket z1 : b \mid c1 \rrbracket)$

shows $\Theta; \mathcal{B}; \Gamma \vdash (\llbracket z1 : b \mid c1 \rrbracket) \lesssim (\llbracket z2 : b \mid c2 \rrbracket)$

using *assms* *subtype-reflI2* **by** *metis*

nominal-function *base-eq* $:: \Gamma \Rightarrow \tau \Rightarrow \tau \Rightarrow \text{bool}$ **where**

base-eq - $\llbracket z1 : b1 \mid c1 \rrbracket \llbracket z2 : b2 \mid c2 \rrbracket = (b1 = b2)$

apply(*auto*, *simp* *add: eqvt-def base-eq-graph-aux-def*)

by (*meson* $\tau.\text{exhaust}$)

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

lemma *subtype-wfT*:

fixes $t1::\tau$ **and** $t2::\tau$

assumes $\Theta; \mathcal{B}; \Gamma \vdash t1 \lesssim t2$

shows $\Theta; \mathcal{B}; \Gamma \vdash_{wf} t1 \wedge \Theta; \mathcal{B}; \Gamma \vdash_{wf} t2$

using *assms subtype-elim* **by** *metis*

lemma *subtype-eq-base*:
assumes $\Theta; \mathcal{B}; \Gamma \vdash (\llbracket z1 : b1 \mid c1 \rrbracket) \lesssim (\llbracket z2 : b2 \mid c2 \rrbracket)$
shows $b1 = b2$
using *subtype.simps assms* **by** *auto*

lemma *subtype-eq-base2*:
assumes $\Theta; \mathcal{B}; \Gamma \vdash t1 \lesssim t2$
shows $b\text{-of } t1 = b\text{-of } t2$
using *assms* **proof**(*rule subtype.induct[of $\Theta \ \mathcal{B} \ \Gamma \ t1 \ t2$],goal-cases*)
case (1 $\Theta \ \Gamma \ z1 \ b \ c1 \ z2 \ c2 \ x$)
then show *?case* **using** *subtype-eq-base* **by** *auto*
qed

lemma *subtype-wf*:
fixes $\tau1::\tau$ **and** $\tau2::\tau$
assumes $\Theta; \mathcal{B}; \Gamma \vdash \tau1 \lesssim \tau2$
shows $\Theta; \mathcal{B}; \Gamma \vdash_{wf} \tau1 \wedge \Theta; \mathcal{B}; \Gamma \vdash_{wf} \tau2$
using *assms*
proof(*rule subtype.induct[of $\Theta \ \mathcal{B} \ \Gamma \ \tau1 \ \tau2$],goal-cases*)
case (1 $\Theta \ \Gamma \ z1 \ b \ c1 \ z2 \ c2 \ x$)
then show *?case* **by** *blast*
qed

lemma *subtype-g-wf*:
fixes $\tau1::\tau$ **and** $\tau2::\tau$ **and** $\Gamma::\Gamma$
assumes $\Theta; \mathcal{B}; \Gamma \vdash \tau1 \lesssim \tau2$
shows $\Theta; \mathcal{B} \vdash_{wf} \Gamma$
using *assms*
proof(*rule subtype.induct[of $\Theta \ \mathcal{B} \ \Gamma \ \tau1 \ \tau2$],goal-cases*)
case (1 $\Theta \ \mathcal{B} \ \Gamma \ z1 \ b \ c1 \ z2 \ c2 \ x$)
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 $\Theta; \mathcal{B}; (z, b, c) \#_{\Gamma} \Gamma \models c'$ **and** $atom \ z \ \# \ \Gamma$ **and** $atom \ x \ \# \ (z, c, z, c', \Gamma)$
shows $\Theta; \mathcal{B}; (x, b, (z \leftrightarrow x) \cdot c) \#_{\Gamma} \Gamma \models (z \leftrightarrow x) \cdot c'$
proof –
have $(z \leftrightarrow x) \cdot \Theta; \mathcal{B}; (z \leftrightarrow x) \cdot ((z, b, c) \#_{\Gamma} \Gamma) \models (z \leftrightarrow x) \cdot c'$
using *True-eqv valid.eqv 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 $x \ \Gamma \ z \ b \ c$] assms fresh-Pair flip-commute* **by** *metis*
qed

lemma *valid-wf-all*:

assumes $\Theta; \mathcal{B}; (z0, b, c0) \#_{\Gamma} G \models c$
shows $wfG \Theta \mathcal{B} G$ **and** $wfC \Theta \mathcal{B} ((z0, b, c0) \#_{\Gamma} G) c$ **and** $atom\ z0 \# G$
using *valid.simps wfC-wf wfG-cons assms by metis+*

lemma *valid-wfT*:

fixes $z::x$

assumes $\Theta; \mathcal{B}; (z0, b, c0[z::=V-var\ z0]_v) \#_{\Gamma} G \models c[z::=V-var\ z0]_v$ **and** $atom\ z0 \# (\Theta, \mathcal{B}, G, c, c0)$

shows $\Theta; \mathcal{B}; G \vdash_{wf} \{ z : b \mid c0 \}$ **and** $\Theta; \mathcal{B}; G \vdash_{wf} \{ z : b \mid c \}$

proof –

have $atom\ z0 \# c0$ **using** *assms fresh-Pair by auto*

moreover **have** $*$: $\Theta; \mathcal{B} \vdash_{wf} (z0, b, c0[z::=V-var\ z0]_{cv}) \#_{\Gamma} G$ **using** *valid-wf-all wfX-wfY assms subst-v-c-def by metis*

ultimately show $wfT: \Theta; \mathcal{B}; G \vdash_{wf} \{ z : b \mid c0 \}$ **using** $wfG-wfT[OF *]$ **by** *auto*

have $atom\ z0 \# c$ **using** *assms fresh-Pair by auto*

moreover **have** $wfC: \Theta; \mathcal{B}; (z0, b, c0[z::=V-var\ z0]_v) \#_{\Gamma} G \vdash_{wf} c[z::=V-var\ z0]_v$ **using** *valid-wf-all assms by metis*

have $\Theta; \mathcal{B}; G \vdash_{wf} \{ z0 : b \mid c[z::=V-var\ z0]_v \}$ **proof**

show $\langle atom\ z0 \# (\Theta, \mathcal{B}, G) \rangle$ **using** *assms fresh-prodN by simp*

show $\langle \Theta; \mathcal{B} \vdash_{wf} b \rangle$ **using** *wfT wfT-wfB by force*

show $\langle \Theta; \mathcal{B}; (z0, b, TRUE) \#_{\Gamma} G \vdash_{wf} c[z::=[z0]^v]_v \rangle$ **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** $\{ z0 : b \mid c[z::=V-var\ z0]_v \} = \{ z : b \mid c \}$ **using** $\langle atom\ z0 \# c0 \rangle \tau.eq-iff\ Abs1-eq-iff(3)$

using *calculation(1) subst-v-c-def by auto*

ultimately show $\Theta; \mathcal{B}; G \vdash_{wf} \{ z : b \mid c \}$ **by** *auto*

qed

lemma *valid-flip*:

fixes $c::c$ **and** $z::x$ **and** $z0::x$ **and** $xx2::x$

assumes $\Theta; \mathcal{B}; (xx2, b, c0[z0::=V-var\ xx2]_v) \#_{\Gamma} \Gamma \models c[z::=V-var\ xx2]_v$ **and**

$atom\ xx2 \# (c0, \Gamma, c, z)$ **and** $atom\ z0 \# (\Gamma, c, z)$

shows $\Theta; \mathcal{B}; (z0, b, c0) \#_{\Gamma} \Gamma \models c[z::=V-var\ z0]_v$

proof –

have $\vdash_{wf} \Theta$ **using** *assms valid-wf-all wfX-wfY by metis*

hence $\Theta; \mathcal{B}; (xx2 \leftrightarrow z0) \cdot ((xx2, b, c0[z0::=V-var\ xx2]_v) \#_{\Gamma} \Gamma) \models ((xx2 \leftrightarrow z0) \cdot c[z::=V-var\ xx2]_v)$

using *valid.eqvt True-eqvt assms beta-flip-eq theta-flip-eq by metis*

hence $\Theta; \mathcal{B}; (((xx2 \leftrightarrow z0) \cdot xx2, b, (xx2 \leftrightarrow z0) \cdot c0[z0::=V-var\ xx2]_v) \#_{\Gamma} (xx2 \leftrightarrow z0) \cdot \Gamma) \models ((xx2 \leftrightarrow z0) \cdot (c[z::=V-var\ xx2]_v))$

using *G-cons-flip[of xx2 z0 xx2 b c0[z0::=V-var\ xx2]_v \Gamma] by auto*

moreover **have** $(xx2 \leftrightarrow z0) \cdot xx2 = z0$ **by** *simp*

moreover **have** $(xx2 \leftrightarrow z0) \cdot c0[z0::=V-var\ xx2]_v = c0$

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 $atom\ xx2 \# \Gamma$ **using** *assms by auto*

moreover **have** $atom\ z0 \# \Gamma$ **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-flip3* *assms* by *simp*
 ultimately show *?thesis* by *auto*
 qed

lemma *subtype-valid*:

assumes $\Theta; \mathcal{B}; \Gamma \vdash t1 \lesssim t2$ and *atom* $y \# \Gamma$ and $t1 = \{ z1 : b \mid c1 \}$ and $t2 = \{ z2 : b \mid c2 \}$
 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* $x \Theta \mathcal{B} \Gamma z c z' c' ba$)

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* $y \Gamma x ba$] 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* $z \# \Gamma$ and $t1 = \{ z : b \mid c1 \}$ and $t2 = \{ z : b \mid c2 \}$
 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* $x \# t$
 shows $\exists b c. t = \{ x : b \mid c \}$

proof –

obtain $z1 b1 c1$ where $*: t = \{ z1 : b1 \mid c1 \} \wedge \text{atom } z1 \# t$ using *obtain-fresh-z* by *metis*
 then have $t = (x \leftrightarrow z1) \cdot t$ using *flip-fresh-fresh* *assms* by *metis*
 also have $\dots = \{ (x \leftrightarrow z1) \cdot z1 : (x \leftrightarrow z1) \cdot b1 \mid (x \leftrightarrow z1) \cdot c1 \}$ using $*$ *assms* by *simp*
 also have $\dots = \{ x : b1 \mid (x \leftrightarrow z1) \cdot c1 \}$ 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 $\Theta; \mathcal{B}; \Gamma \vdash \tau1 \lesssim \tau3$

using *assms* **proof**(*nominal-induct* *avoiding*: $\tau3$ *rule*: *subtype.strong-induct*)
 case (*subtype-baseI* $x \Theta \mathcal{B} \Gamma z c z' c' b$)

hence $b\text{-of } \tau\beta = b$ **using** *subtype-eq-base2 b-of.simps* **by** *metis*
 then obtain $z''\ c''$ **where** $t\beta: \tau\beta = \{\!\{ z'' : b \mid c'' \}\!\} \wedge \text{atom } z'' \# x$
using *obtain-fresh-z2* **by** *metis*
 hence $xf: \text{atom } x \# (z'', c')$ **using** *fresh-prodN subtype-baseI τ .fresh* **by** *auto*
 have $\Theta; \mathcal{B}; \Gamma \vdash \{\!\{ z : b \mid c \}\!\} \lesssim \{\!\{ z'' : b \mid c'' \}\!\}$
proof(*rule Typing.subtype-baseI*)
 show $\langle \text{atom } x \# (\Theta, \mathcal{B}, \Gamma, z, c, z'', c'') \rangle$ **using** $t\beta$ *fresh-prodN subtype-baseI xf* **by** *simp*
 show $\langle \Theta; \mathcal{B}; \Gamma \vdash_{wf} \{\!\{ z : b \mid c \}\!\} \rangle$ **using** *subtype-baseI* **by** *auto*
 show $\langle \Theta; \mathcal{B}; \Gamma \vdash_{wf} \{\!\{ z'' : b \mid c'' \}\!\} \rangle$ **using** *subtype-baseI t β subtype-elim* **by** *metis*
 have $\Theta; \mathcal{B}; (x, b, c'[z'::=[x]^v]_v) \#_{\Gamma} \Gamma \models c''[z''::=[x]^v]_v$
using *subtype-valid[OF $\langle \Theta; \mathcal{B}; \Gamma \vdash \{\!\{ z' : b \mid c' \}\!\} \lesssim \tau\beta \rangle$, of $x\ z'\ b\ c'\ z''\ c''$]* *subtype-baseI t β* **by** *simp*
 thus $\langle \Theta; \mathcal{B}; (x, b, c[z::=[x]^v]_v) \#_{\Gamma} \Gamma \models c''[z''::=[x]^v]_v \rangle$
using *valid-trans-full[of $\Theta\ \mathcal{B}\ x\ b\ c\ z\ \Gamma\ c'\ z'\ c''\ z''$]* *subtype-baseI t β* **by** *simp*
qed
 thus *?case* **using** $t\beta$ **by** *simp*
qed

lemma *subtype-eq-e*:

assumes $\forall i\ s1\ s2\ G. wfG\ P\ \mathcal{B}\ G\ i \wedge wfI\ P\ G\ i \wedge eval\text{-}e\ i\ e1\ s1 \wedge eval\text{-}e\ i\ e2\ s2 \longrightarrow s1 = s2$ **and**
 $\text{atom } z1 \# e1$ **and** $\text{atom } z2 \# e2$ **and** $\text{atom } z1 \# \Gamma$ **and** $\text{atom } z2 \# \Gamma$

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 $wfC\text{-}e\text{-}eq\ wfTI\ assms\ wfX\text{-}wfB\ wfG\text{-}fresh\text{-}x$
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 $wfC\text{-}e\text{-}eq\ wfX\text{-}wfB\ wfTI\ assms\ wfG\text{-}fresh\text{-}x$
by (*simp add: wfT-e-eq*)

moreover obtain $x::x$ **where** $xf: \text{atom } x \# (P, \mathcal{B}, z1, CE\text{-}val\ (V\text{-}var\ z1) == e1, z2, CE\text{-}val\ (V\text{-}var\ z2) == e2, \Gamma)$ **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 $wbg: P; \mathcal{B} \vdash_{wf} ?G \wedge P; \mathcal{B} \vdash_{wf} \Gamma \wedge toSet\ \Gamma \subseteq toSet\ ?G$ **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 $\text{atom } x \# \Gamma$ **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 $wfC\text{-}e\text{-}eq[OF\ assms(6)]\ wf\text{-}subst(2)$

by (*simp add: $\langle \text{atom } x \# \Gamma \rangle\ assms(2)\ subst\text{-}v\text{-}c\text{-}def$*)

qed

moreover hence $P; \mathcal{B} \vdash_{wf} \Gamma$ **using** *wfG-elim* **by** *metis*
 ultimately show *?thesis* **using** *toSet.simps* **by** *auto*
 qed

have *wsc*: $wfC\ P\ \mathcal{B}\ ?G\ ?c$ **proof** –
 have $wfCE\ P\ \mathcal{B}\ ?G\ (CE\text{-}val\ (V\text{-}var\ x))\ b$ **proof**
 show $\langle P; \mathcal{B}; (x, b, (CE\text{-}val\ (V\text{-}var\ z1) == e1)[z1::=V\text{-}var\ x]_v) \#_{\Gamma} \Gamma \vdash_{wf} V\text{-}var\ x : b \rangle$
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 $wfC\ P\ \mathcal{B}\ ?G\ (CE\text{-}val\ (V\text{-}var\ x) == e2)$ **using** *wfC-eqI* **by** *simp*
 thus *?thesis* **using** *subst-cv.simps(6)* $\langle atom\ z2 \# e2 \rangle$ *subst-v-c-def* **by** *simp*
 qed

moreover have $\forall i. wfI\ P\ ?G\ i \wedge is\text{-}satis\text{-}g\ i\ ?G \longrightarrow is\text{-}satis\ i\ ?c$ **proof**(*rule allI* , *rule impI*)
 fix *i*
 assume *as*: $wfI\ P\ ?G\ i \wedge is\text{-}satis\text{-}g\ i\ ?G$
 hence $is\text{-}satis\ i\ ((CE\text{-}val\ (V\text{-}var\ z1) == e1)[z1::=V\text{-}var\ x]_v)$
 by (*simp add: is-satis-g.simps(2)*)
 hence $is\text{-}satis\ i\ (CE\text{-}val\ (V\text{-}var\ x) == e1)$ **using** *subst-cv.simps assms subst-v-c-def* **by** *auto*
 then obtain *s1* and *s2* where $*:eval\text{-}e\ i\ (CE\text{-}val\ (V\text{-}var\ x))\ s1 \wedge eval\text{-}e\ i\ e1\ s2 \wedge s1=s2$ **using**
is-satis.simps eval-c-elim **by** *metis*
 moreover hence $eval\text{-}e\ i\ e2\ s1$ **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 $eval\text{-}e\ i\ e2\ s2'$ **using** *assms wfI-wfCE-eval-e ** **by** *metis*
 ultimately show *?thesis* **using** $*$ *assms(1) wfX-wfY* **by** *metis*
 qed
 ultimately have $is\text{-}satis\ i\ (CE\text{-}val\ (V\text{-}var\ x) == e2)$ **using** *is-satis.simps eval-c-eqI* **by** *force*
 thus $is\text{-}satis\ i\ ((CE\text{-}val\ (V\text{-}var\ z2) == e2)[z2::=V\text{-}var\ x]_v)$ **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 $atom\ x \# (P, \mathcal{B}, \Gamma, z1, CE\text{-}val\ (V\text{-}var\ z1) == e1, z2, CE\text{-}val\ (V\text{-}var\ z2) == e2)$
 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 $\forall i\ s1\ s2\ G. wfG\ P\ \mathcal{B}\ G \wedge wfI\ P\ G\ i \wedge eval\text{-}e\ i\ e1\ s1 \wedge eval\text{-}e\ i\ e2\ s2 \longrightarrow s1 = s2$ **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** $atom\ z1 \# GNil$ **and** $atom\ z2 \# GNil$
 shows $P; \mathcal{B}; GNil \vdash \llbracket z1 : b \mid CE\text{-}val\ (V\text{-}var\ z1) == e1 \rrbracket \lesssim (\llbracket z2 : b \mid CE\text{-}val\ (V\text{-}var\ z2) == e2 \rrbracket)$
 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 $\text{supp } v_1 = \{\}$ **and** $\text{supp } (V\text{-pair } v_1 \ v_2) = \{\}$ **and** $\text{wfTh } P$ **and** $\text{wfCE } P \ \mathcal{B} \ \text{GNil } (CE\text{-val } v_1) \ b$ **and** $\text{wfCE } P \ \mathcal{B} \ \text{GNil } (CE\text{-fst } [V\text{-pair } v_1 \ v_2]^{ce}) \ b$ **and**
 $\text{wfCE } P \ \mathcal{B} \ \text{GNil } (CE\text{-val } v_2) \ b2$ **and** $\text{atom } z1 \ \# \ \text{GNil}$ **and** $\text{atom } z2 \ \# \ \text{GNil}$
shows $P; \mathcal{B}; \text{GNil} \vdash (\llbracket z1 : b \mid CE\text{-val } (V\text{-var } z1) \rrbracket == CE\text{-val } v_1 \ \rrbracket) \lesssim (\llbracket z2 : b \mid CE\text{-val } (V\text{-var } z2) \rrbracket == CE\text{-fst } [V\text{-pair } v_1 \ v_2]^{ce} \ \rrbracket)$

proof –

have $\forall i \ s1 \ s2 \ G. \text{wfG } P \ \mathcal{B} \ G \wedge \text{wfI } P \ G \ i \wedge \text{eval-e } i \ (CE\text{-val } v_1) \ s1 \wedge \text{eval-e } i \ (CE\text{-fst } [V\text{-pair } v_1 \ v_2]^{ce}) \ s2 \longrightarrow s1 = s2$ **proof**(rule+)

fix $i \ s1 \ s2 \ G$

assume $as: \text{wfG } P \ \mathcal{B} \ G \wedge \text{wfI } P \ G \ i \wedge \text{eval-e } i \ (CE\text{-val } v_1) \ s1 \wedge \text{eval-e } i \ (CE\text{-fst } [V\text{-pair } v_1 \ v_2]^{ce}) \ s2$

hence $\text{wfCE } P \ \mathcal{B} \ G \ (CE\text{-val } v_2) \ b2$ **using** *assms wf-weakening*

by (*metis empty-subsetI toSet.simps(1)*)

then obtain $s3$ **where** $\text{eval-e } i \ (CE\text{-val } v_2) \ s3$ **using** *wfI-wfCE-eval-e as* **by** *metis*

hence $\text{eval-v } i \ ((V\text{-pair } v_1 \ v_2)) \ (SPair \ s1 \ s3)$

by (*meson as eval-e-elim(1) eval-v-pairI*)

hence $\text{eval-e } i \ (CE\text{-fst } [V\text{-pair } v_1 \ v_2]^{ce}) \ s1$ **using** *eval-e-fstI eval-e-valI* **by** *metis*

show $s1 = s2$ **using** *as eval-e-uniqueness*

using $\langle \text{eval-e } i \ (CE\text{-fst } [V\text{-pair } v_1 \ v_2]^{ce}) \ s1 \rangle$ **by** *auto*

qed

thus *?thesis* **using** *subtype-eq-e-nil ce.supp assms* **by** *auto*

qed

lemma *subtype-gnil-snd-aux*:

assumes $\text{supp } v_2 = \{\}$ **and** $\text{supp } (V\text{-pair } v_1 \ v_2) = \{\}$ **and** $\text{wfTh } P$ **and** $\text{wfCE } P \ \mathcal{B} \ \text{GNil } (CE\text{-val } v_2) \ b$ **and**

$\text{wfCE } P \ \mathcal{B} \ \text{GNil } (CE\text{-snd } [(V\text{-pair } v_1 \ v_2)]^{ce}) \ b$ **and**

$\text{wfCE } P \ \mathcal{B} \ \text{GNil } (CE\text{-val } v_1) \ b2$ **and** $\text{atom } z1 \ \# \ \text{GNil}$ **and** $\text{atom } z2 \ \# \ \text{GNil}$

shows $P; \mathcal{B}; \text{GNil} \vdash (\llbracket z1 : b \mid CE\text{-val } (V\text{-var } z1) \rrbracket == CE\text{-val } v_2 \ \rrbracket) \lesssim (\llbracket z2 : b \mid CE\text{-val } (V\text{-var } z2) \rrbracket == CE\text{-snd } [(V\text{-pair } v_1 \ v_2)]^{ce} \ \rrbracket)$

proof –

have $\forall i \ s1 \ s2 \ G. \text{wfG } P \ \mathcal{B} \ G \wedge \text{wfI } P \ G \ i \wedge \text{eval-e } i \ (CE\text{-val } v_2) \ s1 \wedge \text{eval-e } i \ (CE\text{-snd } [(V\text{-pair } v_1 \ v_2)]^{ce}) \ s2 \longrightarrow s1 = s2$ **proof**(rule+)

fix $i \ s1 \ s2 \ G$

assume $as: \text{wfG } P \ \mathcal{B} \ G \wedge \text{wfI } P \ G \ i \wedge \text{eval-e } i \ (CE\text{-val } v_2) \ s1 \wedge \text{eval-e } i \ (CE\text{-snd } [(V\text{-pair } v_1 \ v_2)]^{ce}) \ s2$

hence $\text{wfCE } P \ \mathcal{B} \ G \ (CE\text{-val } v_1) \ b2$ **using** *assms wf-weakening*

by (*metis empty-subsetI toSet.simps(1)*)

then obtain $s3$ **where** $\text{eval-e } i \ (CE\text{-val } v_1) \ s3$ **using** *wfI-wfCE-eval-e as* **by** *metis*

hence $\text{eval-v } i \ ((V\text{-pair } v_1 \ v_2)) \ (SPair \ s3 \ s1)$

by (*meson as eval-e-elim eval-v-pairI*)

hence $\text{eval-e } i \ (CE\text{-snd } [(V\text{-pair } v_1 \ v_2)]^{ce}) \ s1$ **using** *eval-e-sndI eval-e-valI* **by** *metis*

show $s1 = s2$ **using** *as eval-e-uniqueness*

using $\langle \text{eval-e } i \ (CE\text{-snd } [(V\text{-pair } v_1 \ v_2)]^{ce}) \ s1 \rangle$ **by** *auto*

qed

thus *?thesis* **using** *assms subtype-eq-e-nil* **by** (*simp add: ce.supp ce.supp*)

qed

lemma *subtype-gnil-fst*:

assumes $\Theta ; \{\|\}; GNil \vdash_{wf} [\#1[[v_1, v_2]^v]^{ce}]^{ce} : b$
shows $\Theta ; \{\|\}; GNil \vdash (\{\| z_1 : b \mid [[z_1]^v]^{ce} == [v_1]^{ce} \}) \lesssim$
 $(\{\| z_2 : b \mid [[z_2]^v]^{ce} == [\#1[[v_1, v_2]^v]^{ce}]^{ce} \})$
proof –
obtain $b2$ **where** $** : \Theta ; \{\|\}; GNil \vdash_{wf} V\text{-pair } v_1 \ v_2 : B\text{-pair } b \ b2$ **using** $wfCE\text{-elems}(4)[OF \text{ assms}]$
 $wfCE\text{-elems}$ **by** *metis*
obtain $b1' \ b2'$ **where** $* : B\text{-pair } b \ b2 = B\text{-pair } b1' \ b2' \wedge \Theta ; \{\|\}; GNil \vdash_{wf} v_1 : b1' \wedge \Theta ; \{\|\}$
 $; GNil \vdash_{wf} v_2 : b2'$
using $wfV\text{-elems}(3)[OF **]$ **by** *metis*

show *?thesis* **proof**(*rule subtype-gnil-fst-aux*)
show $\langle supp \ v_1 = \{\} \rangle$ **using** $* \ wfV\text{-supp-nil}$ **by** *auto*
show $\langle supp \ (V\text{-pair } v_1 \ v_2) = \{\} \rangle$ **using** $** \ wfV\text{-supp-nil } e.supp$ **by** *metis*
show $\langle \vdash_{wf} \Theta \rangle$ **using** *assms wfX-wfY* **by** *metis*
show $\langle \Theta ; \{\|\}; GNil \vdash_{wf} CE\text{-val } v_1 : b \rangle$ **using** $wfCE\text{-valI } *$ **by** *auto*
show $\langle \Theta ; \{\|\}; GNil \vdash_{wf} CE\text{-fst } [V\text{-pair } v_1 \ v_2]^{ce} : b \rangle$ **using** *assms* **by** *auto*
show $\langle \Theta ; \{\|\}; GNil \vdash_{wf} CE\text{-val } v_2 : b2 \rangle$ **using** $wfCE\text{-valI } *$ **by** *auto*
show $\langle atom \ z_1 \ \# \ GNil \rangle$ **using** *fresh-GNil* **by** *metis*
show $\langle atom \ z_2 \ \# \ GNil \rangle$ **using** *fresh-GNil* **by** *metis*
qed
qed

lemma *subtype-gnil-snd*:
assumes $wfCE \ P \ \{\|\} \ GNil \ (CE\text{-snd } ([V\text{-pair } v_1 \ v_2]^{ce})) \ b$
shows $P ; \{\|\}; GNil \vdash (\{\| z_1 : b \mid CE\text{-val } (V\text{-var } z1) == CE\text{-val } v_2 \}) \lesssim (\{\| z2 : b \mid CE\text{-val}$
 $(V\text{-var } z2) == CE\text{-snd } ([V\text{-pair } v_1 \ v_2]^{ce}) \})$
proof –
obtain $b1$ **where** $** : P ; \{\|\}; GNil \vdash_{wf} V\text{-pair } v_1 \ v_2 : B\text{-pair } b1 \ b$ **using** $wfCE\text{-elems } assms$ **by**
metis
obtain $b1' \ b2'$ **where** $* : B\text{-pair } b1 \ b = B\text{-pair } b1' \ b2' \wedge P ; \{\|\}; GNil \vdash_{wf} v_1 : b1' \wedge P ; \{\|\}$
 $; GNil \vdash_{wf} v_2 : b2'$ **using** $wfV\text{-elems}(3)[OF **]$ **by** *metis*

show *?thesis* **proof**(*rule subtype-gnil-snd-aux*)
show $\langle supp \ v_2 = \{\} \rangle$ **using** $* \ wfV\text{-supp-nil}$ **by** *auto*
show $\langle supp \ (V\text{-pair } v_1 \ v_2) = \{\} \rangle$ **using** $** \ wfV\text{-supp-nil } e.supp$ **by** *metis*
show $\langle \vdash_{wf} P \rangle$ **using** *assms wfX-wfY* **by** *metis*
show $\langle P ; \{\|\}; GNil \vdash_{wf} CE\text{-val } v_1 : b1 \rangle$ **using** $wfCE\text{-valI } *$ **by** *simp*
show $\langle P ; \{\|\}; GNil \vdash_{wf} CE\text{-snd } ([V\text{-pair } v_1 \ v_2]^{ce}) : b \rangle$ **using** *assms* **by** *auto*
show $\langle P ; \{\|\}; GNil \vdash_{wf} CE\text{-val } v_2 : b \rangle$ **using** $wfCE\text{-valI } *$ **by** *simp*
show $\langle atom \ z1 \ \# \ GNil \rangle$ **using** *fresh-GNil* **by** *metis*
show $\langle atom \ z2 \ \# \ GNil \rangle$ **using** *fresh-GNil* **by** *metis*
qed
qed

lemma *subtype-fresh-tau*:
fixes $x :: x$
assumes $atom \ x \ \# \ t1$ **and** $atom \ x \ \# \ \Gamma$ **and** $P ; \mathcal{B} ; \Gamma \vdash t1 \lesssim t2$
shows $atom \ x \ \# \ t2$
proof –
have $wfg : P ; \mathcal{B} \vdash_{wf} \Gamma$ **using** *subtype-wf wfX-wfY assms* **by** *metis*
have $wft : wft \ P \ \mathcal{B} \ \Gamma \ t2$ **using** *subtype-wf wfX-wfY assms* **by** *blast*

hence $\text{supp } t2 \subseteq \text{atom-dom } \Gamma \cup \text{supp } \mathcal{B}$ **using** wf-supp
using atom-dom.simps **by** auto
moreover have $\text{atom } x \notin \text{atom-dom } \Gamma$ **using** $\langle \text{atom } x \# \Gamma \rangle \text{ wfG-atoms-suppl-eq wfg fresh-def}$ **by** blast
ultimately show $\text{atom } x \# t2$ **using** fresh-def
by $(\text{metis } \text{Un-iff contra-subsetD } x\text{-not-in-b-set})$
qed

lemma subtype-if-simp :

assumes $\text{wfT } P \ \mathcal{B} \ GNil \ (\llbracket z1 : b \mid \text{CE-val } (V\text{-lit } l) \rrbracket == \text{CE-val } (V\text{-lit } l) \ \text{IMP} \ c[z ::= V\text{-var } z1]_v \rrbracket)$ **and**

$\text{wfT } P \ \mathcal{B} \ GNil \ (\llbracket z : b \mid c \rrbracket)$ **and** $\text{atom } z1 \# c$
shows $P; \mathcal{B}; GNil \vdash (\llbracket z1 : b \mid \text{CE-val } (V\text{-lit } l) \rrbracket == \text{CE-val } (V\text{-lit } l) \ \text{IMP} \ c[z ::= V\text{-var } z1]_v \rrbracket)$
 $\lesssim (\llbracket z : b \mid c \rrbracket)$

proof –

obtain $xx::x$ **where** $xx: \text{atom } x \# (P, \mathcal{B}, z1, \text{CE-val } (V\text{-lit } l) == \text{CE-val } (V\text{-lit } l) \ \text{IMP} \ c[z ::= V\text{-var } z1]_v, z, c, GNil)$ **using** obtain-fresh-z **by** blast

hence $xx2: \text{atom } x \# (\text{CE-val } (V\text{-lit } l) == \text{CE-val } (V\text{-lit } l) \ \text{IMP} \ c[z ::= V\text{-var } z1]_v, c, GNil)$
using $\text{fresh-prod7 fresh-prod3}$ **by** fast

have $*:P; \mathcal{B}; (x, b, (\text{CE-val } (V\text{-lit } l) == \text{CE-val } (V\text{-lit } l) \ \text{IMP} \ c[z ::= V\text{-var } z1]_v)[z1 ::= V\text{-var } x]_v) \#_{\Gamma} GNil \models c[z ::= V\text{-var } x]_v$ **(is** $P; \mathcal{B}; ?G \models ?c$ **)** **proof** –

have $\text{wfC } P \ \mathcal{B} \ ?G \ ?c$ **using** $\text{wfT-wfC-cons}[OF \ \text{assms}(1) \ \text{assms}(2), of \ x] \ xx \ \text{fresh-prod5 fresh-prod3}$
 subst-v-c-def **by** metis

moreover have $(\forall i. \text{wfI } P \ ?G \ i \wedge \text{is-satis-g } i \ ?G \longrightarrow \text{is-satis } i \ ?c)$ **proof**($\text{rule allI}, \text{rule impI}$)

fix i

assume $as1: \text{wfI } P \ ?G \ i \wedge \text{is-satis-g } i \ ?G$

have $((\text{CE-val } (V\text{-lit } l) == \text{CE-val } (V\text{-lit } l) \ \text{IMP} \ c[z ::= V\text{-var } z1]_v)[z1 ::= V\text{-var } x]_v) = ((\text{CE-val } (V\text{-lit } l) == \text{CE-val } (V\text{-lit } l) \ \text{IMP} \ c[z ::= V\text{-var } x]_v))$

using $\text{assms subst-v-c-def}$ **by** auto

hence $\text{is-satis } i ((\text{CE-val } (V\text{-lit } l) == \text{CE-val } (V\text{-lit } l) \ \text{IMP} \ c[z ::= V\text{-var } x]_v))$ **using**
 $\text{is-satis-g.simps } as1$ **by** presburger

moreover have $\text{is-satis } i ((\text{CE-val } (V\text{-lit } l) == \text{CE-val } (V\text{-lit } l)))$ **using** $\text{is-satis.simps eval-c-eqI}[of \ i \ (\text{CE-val } (V\text{-lit } l)) \ \text{eval-l } l] \ \text{eval-e-uniqueness}$

$\text{eval-e-valI eval-v-litI}$ **by** metis

ultimately show $\text{is-satis } i \ ?c$ **using** $\text{is-satis-mp}[of \ i]$ **by** metis

qed

ultimately show $?thesis$ **using** valid.simps **by** simp

qed

moreover have $\text{atom } x \# (P, \mathcal{B}, GNil, z1, \text{CE-val } (V\text{-lit } l) == \text{CE-val } (V\text{-lit } l) \ \text{IMP} \ c[z ::= V\text{-var } z1]_v, z, c)$

unfolding $\text{fresh-prod5 } \tau.\text{fresh}$ **using** $xx \ \text{fresh-prodN } x\text{-fresh-b}$ **by** metis

ultimately show $?thesis$ **using** $\text{subtype-baseI assms } xx \ xx2$ **by** metis
qed

lemma subtype-if :

assumes $P; \mathcal{B}; \Gamma \vdash \llbracket z : b \mid c \rrbracket \lesssim \llbracket z' : b \mid c' \rrbracket$ **and**

$\text{wfT } P \ \mathcal{B} \ \Gamma \ (\llbracket z1 : b \mid \text{CE-val } v == \text{CE-val } (V\text{-lit } l) \ \text{IMP} \ c[z ::= V\text{-var } z1]_v \rrbracket)$ **and**

$\text{wfT } P \ \mathcal{B} \ \Gamma \ (\llbracket z2 : b \mid \text{CE-val } v == \text{CE-val } (V\text{-lit } l) \ \text{IMP} \ c'[z' ::= V\text{-var } z2]_v \rrbracket)$ **and**

$\text{atom } z1 \# v$ **and** $\text{atom } z \# \Gamma$ **and** $\text{atom } z1 \# c$ **and** $\text{atom } z2 \# c'$ **and** $\text{atom } z2 \# v$

shows $P; \mathcal{B}; \Gamma \vdash \llbracket z1 : b \mid \text{CE-val } v == \text{CE-val } (V\text{-lit } l) \ \text{IMP} \ c[z ::= V\text{-var } z1]_v \rrbracket \lesssim \llbracket z2 : b \mid \text{CE-val } v == \text{CE-val } (V\text{-lit } l) \ \text{IMP} \ c'[z' ::= V\text{-var } z2]_v \rrbracket$

proof –

obtain $xx::x$ **where** $xx: \text{atom } x \# (P, \mathcal{B}, z, c, z', c', z1, \text{CE-val } v == \text{CE-val } (V\text{-lit } l) \ \text{IMP} \ c[z ::= V\text{-var } z1]_v)$

$z1]_v, z2, CE\text{-}val\ v == CE\text{-}val\ (V\text{-}lit\ l)\ IMP\ c'[z'::=V\text{-}var\ z2]_v, \Gamma)$
using *obtain-fresh-z* **by** *blast*
hence $xf: atom\ x \# (z, c, z', c', \Gamma)$ **by** *simp*
have $xf2: atom\ x \# (z1, CE\text{-}val\ v == CE\text{-}val\ (V\text{-}lit\ l)\ IMP\ c[z::=V\text{-}var\ z1]_v, z2, CE\text{-}val\ v ==$
 $CE\text{-}val\ (V\text{-}lit\ l)\ IMP\ c'[z'::=V\text{-}var\ z2]_v, \Gamma)$
using $xx\ fresh\text{-}prod4\ fresh\text{-}prodN$ **by** *metis*

moreover **have** $P; \mathcal{B}; (x, b, (CE\text{-}val\ v == CE\text{-}val\ (V\text{-}lit\ l)\ IMP\ c[z::=V\text{-}var\ z1]_v)[z1::=V\text{-}var\ x]_v) \#_{\Gamma} \Gamma \models (CE\text{-}val\ v == CE\text{-}val\ (V\text{-}lit\ l)\ IMP\ c'[z'::=V\text{-}var\ z2]_v)[z2::=V\text{-}var\ x]_v$
 $(is\ P; \mathcal{B}; ?G \models ?c)$
proof –
have $wbc: wfC\ P\ \mathcal{B}\ ?G\ ?c$ **using** *assms* $xx\ fresh\text{-}prod4\ fresh\text{-}prod2\ wfT\text{-}wfC\text{-}cons\ assms\ subst\text{-}v\text{-}c\text{-}def$
by *metis*

moreover **have** $\forall i. wfI\ P\ ?G\ i \wedge is\text{-}satis\text{-}g\ i\ ?G \longrightarrow is\text{-}satis\ i\ ?c$ **proof**(*rule allI, rule impI*)
fix i
assume $a1: wfI\ P\ ?G\ i \wedge is\text{-}satis\text{-}g\ i\ ?G$
thm *is-satis.simps*
have $*$: $is\text{-}satis\ i\ ((CE\text{-}val\ v == CE\text{-}val\ (V\text{-}lit\ l))) \longrightarrow is\text{-}satis\ i\ ((c'[z'::=V\text{-}var\ z2]_v)[z2::=V\text{-}var\ x]_v)$
proof
assume $a2: is\text{-}satis\ i\ ((CE\text{-}val\ v == CE\text{-}val\ (V\text{-}lit\ l)))$

have $is\text{-}satis\ i\ ((CE\text{-}val\ v == CE\text{-}val\ (V\text{-}lit\ l)\ IMP\ (c[z::=V\text{-}var\ z1]_v))[z1::=V\text{-}var\ x]_v)$
using $a1\ is\text{-}satis\text{-}g.\text{simps}$ **by** *simp*
moreover **have** $((CE\text{-}val\ v == CE\text{-}val\ (V\text{-}lit\ l)\ IMP\ (c[z::=V\text{-}var\ z1]_v))[z1::=V\text{-}var\ x]_v =$
 $(CE\text{-}val\ v == CE\text{-}val\ (V\text{-}lit\ l)\ IMP\ ((c[z::=V\text{-}var\ z1]_v)[z1::=V\text{-}var\ x]_v))$
using *assms* *subst-v-c-def* **by** *simp*
ultimately **have** $is\text{-}satis\ i\ (CE\text{-}val\ v == CE\text{-}val\ (V\text{-}lit\ l)\ IMP\ ((c[z::=V\text{-}var\ z1]_v)[z1::=V\text{-}var\ x]_v))$ **by** *arg0*

hence $is\text{-}satis\ i\ ((c[z::=V\text{-}var\ z1]_v)[z1::=V\text{-}var\ x]_v)$ **using** $a2\ is\text{-}satis\text{-}mp$ **by** *auto*
moreover **have** $((c[z::=V\text{-}var\ z1]_v)[z1::=V\text{-}var\ x]_v) = ((c[z::=V\text{-}var\ x]_v))$ **using** *assms* **by**
auto
ultimately **have** $is\text{-}satis\ i\ ((c[z::=V\text{-}var\ x]_v))$ **using** $a2\ is\text{-}satis.\text{simps}$ **by** *auto*

hence $is\text{-}satis\text{-}g\ i\ ((x, b, (c[z::=V\text{-}var\ x]_v)) \#_{\Gamma} \Gamma)$ **using** $a1\ is\text{-}satis\text{-}g.\text{simps}$ **by** *meson*
moreover **have** $wfI\ P\ ((x, b, (c[z::=V\text{-}var\ x]_v)) \#_{\Gamma} \Gamma)\ i$ **proof** –
obtain s **where** $Some\ s = i\ x \wedge wfRCV\ P\ s\ b \wedge wfI\ P\ \Gamma\ i$ **using** *wfI-def* $a1$ **by** *auto*
thus *?thesis* **using** *wfI-def* **by** *auto*
qed
ultimately **have** $is\text{-}satis\ i\ ((c'[z'::=V\text{-}var\ x]_v))$ **using** *subtype-valid* *assms*(1) *xf* *valid.simps* **by**
simp

moreover **have** $(c'[z'::=V\text{-}var\ x]_v) = ((c'[z'::=V\text{-}var\ z2]_v)[z2::=V\text{-}var\ x]_v)$ **using** *assms* **by**
auto
ultimately **show** $is\text{-}satis\ i\ ((c'[z'::=V\text{-}var\ z2]_v)[z2::=V\text{-}var\ x]_v)$ **by** *auto*
qed

moreover **have** $?c = ((CE\text{-}val\ v == CE\text{-}val\ (V\text{-}lit\ l)\ IMP\ ((c'[z'::=V\text{-}var\ z2]_v)[z2::=V\text{-}var\ x]_v))$
using *assms* *subst-v-c-def* **by** *simp*

thm *wfC-elim*
moreover have $\exists b1\ b2. \text{eval-c } i \ (CE\text{-val } v == CE\text{-val } (V\text{-lit } l)) \ b1 \wedge$
 $\text{eval-c } i \ c'[z'::=V\text{-var } z2]_v [z2::=V\text{-var } x]_v \ b2$ **proof** –
thm *assms(2)*
have $wfC\ P\ \mathcal{B}\ ?G\ (CE\text{-val } v == CE\text{-val } (V\text{-lit } l))$ **using** *wbc wfC-elim(7) assms subst-cv.simps*
subst-v-c-def **by** *fastforce*

moreover have $wfC\ P\ \mathcal{B}\ ?G\ (c'[z'::=V\text{-var } z2]_v [z2::=V\text{-var } x]_v)$ **proof** (*rule wfT-wfC-cons*)
show $\langle P; \mathcal{B}; \Gamma \vdash_{wf} \{ z1 : b \mid CE\text{-val } v == CE\text{-val } (V\text{-lit } l) \text{ IMP } (c[z::=V\text{-var } z1]_v) \} \rangle$
using *assms subst-v-c-def* **by** *auto*
have $\{ z2 : b \mid c'[z'::=V\text{-var } z2]_v \} = \{ z' : b \mid c' \}$ **using** *assms subst-v-c-def* **by** *auto*
thus $\langle P; \mathcal{B}; \Gamma \vdash_{wf} \{ z2 : b \mid c'[z'::=V\text{-var } z2]_v \} \rangle$ **using** *assms subtype-elim* **by** *metis*
show $\langle atom\ x \ \sharp \ (CE\text{-val } v == CE\text{-val } (V\text{-lit } l) \text{ IMP } c[z::=V\text{-var } z1]_v, c'[z'::=V\text{-var } z2]_v,$
 $\Gamma) \rangle$ **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 $atom\ x \ \sharp \ (P, \mathcal{B}, \Gamma, z1, CE\text{-val } v == CE\text{-val } (V\text{-lit } l) \text{ IMP } c[z::=V\text{-var } z1]_v,$
 $z2, CE\text{-val } v == CE\text{-val } (V\text{-lit } l) \text{ IMP } c'[z'::=V\text{-var } z2]_v)$
unfolding *fresh-prod5* $\tau.fresh$ **using** *xx xf2 fresh-prodN x-fresh-b* **by** *metis*
ultimately show *?thesis* **using** *subtype-baseI assms xf2* **by** *metis*
qed

lemma *eval-e-concat-eq*:

assumes $wfI\ \Theta\ \Gamma\ i$
shows $\exists s. \text{eval-e } i \ (CE\text{-val } (V\text{-lit } (L\text{-bitvec } (v1\ @\ v2)))) \ s \wedge \text{eval-e } i \ (CE\text{-concat } [(V\text{-lit } (L\text{-bitvec } v1))]^{ce} [(V\text{-lit } (L\text{-bitvec } v2))]^{ce}) \ s$
using *eval-e-valI eval-e-concatI eval-v-litI eval-l.simps* **by** *metis*

lemma *is-satis-eval-e-eq-imp*:

assumes $wfI\ \Theta\ \Gamma\ i$ **and** $\text{eval-e } i\ e1\ s$ **and** $\text{eval-e } i\ e2\ s$
and $\text{is-satis } i \ (CE\text{-val } (V\text{-var } x) == e1)$ (**is** $\text{is-satis } i\ ?c1$)
shows $\text{is-satis } i \ (CE\text{-val } (V\text{-var } x) == e2)$
proof –
have $*:\text{eval-c } i\ ?c1\ \text{True}$ **using** *assms is-satis.simps* **by** *blast*
hence $\text{eval-e } i \ (CE\text{-val } (V\text{-var } x))\ s$ **using** *assms is-satis.simps eval-c-elim*
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 $\forall \Gamma\ i. wfI\ \Theta\ \Gamma\ i \longrightarrow (\exists s. \text{eval-e } i\ e1\ s \wedge \text{eval-e } i\ e2\ s)$ **and** $\Theta; \mathcal{B}; GNil \vdash_{wf} e1 : b$ **and**
 $\Theta; \mathcal{B}; GNil \vdash_{wf} e2 : b$
shows $\Theta; \mathcal{B}; (x, b, (CE\text{-val } (V\text{-var } x) == e1)) \#_{\Gamma} GNil \models (CE\text{-val } (V\text{-var } x) == e2)$

proof(*rule validI*)
show $\Theta; \mathcal{B}; (x, b, CE\text{-}val (V\text{-}var\ x) == e1) \#_{\Gamma} GNil \vdash_{wf} CE\text{-}val (V\text{-}var\ x) == e2$
proof
have $\Theta; \mathcal{B}; (x, b, TRUE) \#_{\Gamma} GNil \vdash_{wf} CE\text{-}val (V\text{-}var\ x) == e1$ **using** *assms wfC-eqI wfE-valI wfV-varI wfX-wfY*
by (*simp add: fresh-GNil wfC-e-eq*)
hence $\Theta; \mathcal{B} \vdash_{wf} (x, b, CE\text{-}val (V\text{-}var\ x) == e1) \#_{\Gamma} GNil$ **using** *wfG-consI fresh-GNil wfX-wfY assms wfX-wfB* **by** *metis*
thus $\Theta; \mathcal{B}; (x, b, CE\text{-}val (V\text{-}var\ x) == e1) \#_{\Gamma} GNil \vdash_{wf} CE\text{-}val (V\text{-}var\ x) : b$ **using** *wfCE-valI wfV-varI wfX-wfY*
lookup.simps assms wfX-wfY **by** *simp*
show $\Theta; \mathcal{B}; (x, b, CE\text{-}val (V\text{-}var\ x) == e1) \#_{\Gamma} GNil \vdash_{wf} e2 : b$ **using** *assms wf-weakening wfX-wfY*
by (*metis (full-types) (Θ; B; (x, b, CE-val (V-var x) == e1) #Γ GNil ⊢_{wf} CE-val (V-var x) : b) empty-iff subsetI toSet.simps(1)*)
qed
show $\forall i. wfI\ \Theta\ ((x, b, CE\text{-}val (V\text{-}var\ x) == e1) \#_{\Gamma} GNil)\ i \wedge is\text{-}satis\text{-}g\ i\ ((x, b, CE\text{-}val (V\text{-}var\ x) == e1) \#_{\Gamma} GNil) \longrightarrow is\text{-}satis\ i\ (CE\text{-}val (V\text{-}var\ x) == e2)$
proof(*rule, rule*)
fix *i*
assume $wfI\ \Theta\ ((x, b, CE\text{-}val (V\text{-}var\ x) == e1) \#_{\Gamma} GNil)\ i \wedge is\text{-}satis\text{-}g\ i\ ((x, b, CE\text{-}val (V\text{-}var\ x) == e1) \#_{\Gamma} GNil)$
moreover then obtain *s* **where** $eval\text{-}e\ i\ e1\ s \wedge eval\text{-}e\ i\ e2\ s$ **using** *assms* **by** *auto*
ultimately show $is\text{-}satis\ i\ (CE\text{-}val (V\text{-}var\ x) == e2)$ **using** *assms is-satis-eval-e-eq-imp is-satis-g.simps* **by** *meson*
qed
qed

lemma *subtype-concat*:

assumes $\vdash_{wf} \Theta$
shows $\Theta; \mathcal{B}; GNil \vdash \{ z : B\text{-}bitvec \mid CE\text{-}val (V\text{-}var\ z) == CE\text{-}val (V\text{-}lit (L\text{-}bitvec (v1 @ v2))) \} \lesssim$
 $\{ z : B\text{-}bitvec \mid CE\text{-}val (V\text{-}var\ z) == CE\text{-}concat [(V\text{-}lit (L\text{-}bitvec v1))]^{ce} [(V\text{-}lit (L\text{-}bitvec v2))]^{ce} \} \text{ (is } \Theta; \mathcal{B}; GNil \vdash ?t1 \lesssim ?t2)$
proof –
obtain $x :: x$ **where** $x : atom\ x \# (\Theta, \mathcal{B}, GNil, z, CE\text{-}val (V\text{-}var\ z) == CE\text{-}val (V\text{-}lit (L\text{-}bitvec (v1 @ v2))))$,
 $z, CE\text{-}val (V\text{-}var\ z) == CE\text{-}concat [V\text{-}lit (L\text{-}bitvec v1)]^{ce} [V\text{-}lit (L\text{-}bitvec v2)]^{ce}$
(is ?xfree)
using *obtain-fresh* **by** *auto*

have $wb1 : \Theta; \mathcal{B}; GNil \vdash_{wf} CE\text{-}val (V\text{-}lit (L\text{-}bitvec (v1 @ v2))) : B\text{-}bitvec$ **using** *wfX-wfY wfCE-valI wfV-litI assms base-for-lit.simps wfG-nilI* **by** *metis*
hence $wb2 : \Theta; \mathcal{B}; GNil \vdash_{wf} CE\text{-}concat [(V\text{-}lit (L\text{-}bitvec v1))]^{ce} [(V\text{-}lit (L\text{-}bitvec v2))]^{ce} : B\text{-}bitvec$
using *wfCE-concatI wfX-wfY wfV-litI base-for-lit.simps wfCE-valI* **by** *metis*

show *?thesis* **proof**
show $\Theta; \mathcal{B}; GNil \vdash_{wf} ?t1$ **using** *wfT-e-eq fresh-GNil wb1 wb2* **by** *metis*
show $\Theta; \mathcal{B}; GNil \vdash_{wf} ?t2$ **using** *wfT-e-eq fresh-GNil wb1 wb2* **by** *metis*
show *?xfree* **using** *x* **by** *auto*

show $\Theta; \mathcal{B}; (x, B\text{-bitvec}, (CE\text{-val } (V\text{-var } z) == CE\text{-val } (V\text{-lit } (L\text{-bitvec } (v1 @ v2)))) [z::=V\text{-var } x]_v) \#_{\Gamma}$
 $GNil \models (CE\text{-val } (V\text{-var } z) == CE\text{-concat } [(V\text{-lit } (L\text{-bitvec } v1))]^{ce} [(V\text{-lit } (L\text{-bitvec } v2))]^{ce}) [z::=V\text{-var } x]_v$
using *valid-eval-e-eq eval-e-concat-eq wb1 wb2 subst-v-c-def* **by** *fastforce*
qed
qed

lemma *subtype-len*:

assumes $\vdash_{wf} \Theta$
shows $\Theta; \mathcal{B}; GNil \vdash \{ z' : B\text{-int} \mid CE\text{-val } (V\text{-var } z') == CE\text{-val } (V\text{-lit } (L\text{-num } (int (length v)))) \}$
 \lesssim
 $\{ z : B\text{-int} \mid CE\text{-val } (V\text{-var } z) == CE\text{-len } [(V\text{-lit } (L\text{-bitvec } v))]^{ce} \}$ **(is** $\Theta; \mathcal{B}; GNil \vdash ?t1 \lesssim ?t2)$
proof –

have $*$: $\Theta \vdash_{wf} [] \wedge \Theta; \mathcal{B}; GNil \vdash_{wf} []_{\Delta}$ **using** *assms wfG-nilI wfD-emptyI wfPhi-emptyI* **by** *auto*
obtain $x::x$ **where** x : $atom\ x \# (\Theta, \mathcal{B}, GNil, z', CE\text{-val } (V\text{-var } z') == CE\text{-val } (V\text{-lit } (L\text{-num } (int (length v)))) , z, CE\text{-val } (V\text{-var } z) == CE\text{-len } [(V\text{-lit } (L\text{-bitvec } v))]^{ce})$
(is $atom\ x \# ?F)$
using *obtain-fresh* **by** *metis*
then show *?thesis* **proof**
have $\Theta; \mathcal{B}; GNil \vdash_{wf} CE\text{-val } (V\text{-lit } (L\text{-num } (int (length v)))) : B\text{-int}$
using *wfCE-valI * wfV-litI base-for-lit.simps*
by *(metis wfE-valI wfX-wfY)*

thus $\Theta; \mathcal{B}; GNil \vdash_{wf} ?t1$ **using** *wfT-e-eq fresh-GNil* **by** *auto*

have $\Theta; \mathcal{B}; GNil \vdash_{wf} CE\text{-len } [(V\text{-lit } (L\text{-bitvec } v))]^{ce} : B\text{-int}$
using *wfE-valI * wfV-litI base-for-lit.simps wfE-valI wfX-wfY*
by *(metis wfCE-lenI wfCE-valI)*

thus $\Theta; \mathcal{B}; GNil \vdash_{wf} ?t2$ **using** *wfT-e-eq fresh-GNil* **by** *auto*

show $\Theta; \mathcal{B}; (x, B\text{-int}, (CE\text{-val } (V\text{-var } z') == CE\text{-val } (V\text{-lit } (L\text{-num } (int (length v))))) [z'::=V\text{-var } x]_v) \#_{\Gamma} GNil \models (CE\text{-val } (V\text{-var } z) == CE\text{-len } [(V\text{-lit } (L\text{-bitvec } v))]^{ce}) [z::=V\text{-var } x]_v$
(is $\Theta; \mathcal{B}; ?G \models ?c$ **)** **using** *valid-len assms subst-v-c-def* **by** *auto*
qed
qed

lemma *subtype-base-fresh*:

assumes $\Theta; \mathcal{B}; \Gamma \vdash_{wf} \{ z : b \mid c \}$ **and** $\Theta; \mathcal{B}; \Gamma \vdash_{wf} \{ z : b \mid c' \}$ **and**
 $atom\ z \# \Gamma$ **and** $\Theta; \mathcal{B}; (z, b, c) \#_{\Gamma} \Gamma \models c'$
shows $\Theta; \mathcal{B}; \Gamma \vdash \{ z : b \mid c \} \lesssim \{ z : b \mid c' \}$
proof –
obtain $x::x$ **where** $*$: $atom\ x \# ((\Theta, \mathcal{B}, z, c, z, c', \Gamma), (\Theta, \mathcal{B}, \Gamma, \{ z : b \mid c \}, \{ z : b \mid c' \}))$ **using** *obtain-fresh* **by** *metis*
moreover **hence** $atom\ x \# \Gamma$ **using** *fresh-Pair* **by** *auto*
moreover **hence** $\Theta; \mathcal{B}; (x, b, c [z::=V\text{-var } x]_v) \#_{\Gamma} \Gamma \models c' [z::=V\text{-var } x]_v$ **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 $wfG \ \Theta \ \mathcal{B} \ \Gamma$ **and** $opp = Plus \wedge ll = (L-num \ (n1+n2)) \vee (opp = LEq \wedge ll = (\text{if } n1 \leq n2 \text{ then } L-true \text{ else } L-false))$

and $(opp = Plus \longrightarrow b = B-int) \wedge (opp = LEq \longrightarrow b = B-bool)$

shows $\Theta; \mathcal{B}; \Gamma \vdash (\llbracket z : b \mid C-eq \ (CE-val \ (V-var \ z)) \ (CE-val \ (V-lit \ (ll))) \rrbracket) \lesssim$
 $\llbracket z : b \mid C-eq \ (CE-val \ (V-var \ z)) \ (CE-op \ opp \ [(V-lit \ (L-num \ n1))]^{ce} \ [(V-lit \ (L-num \ n2))]^{ce}) \rrbracket$ **(is** $\Theta; \mathcal{B}; \Gamma \vdash ?T1 \lesssim ?T2)$

proof –

obtain $x::x$ **where** $xf: atom \ x \ \# \ (z, CE-val \ (V-var \ z) == CE-val \ (V-lit \ (ll)) , z, CE-val \ (V-var \ z) == CE-op \ opp \ [(V-lit \ (L-num \ n1))]^{ce} \ [(V-lit \ (L-num \ n2))]^{ce} , \Gamma)$

using *obtain-fresh* **by** *blast*

have $\Theta; \mathcal{B}; \Gamma \vdash (\llbracket x : b \mid C-eq \ (CE-val \ (V-var \ x)) \ (CE-val \ (V-lit \ (ll))) \rrbracket) \lesssim$
 $\llbracket x : b \mid C-eq \ (CE-val \ (V-var \ x)) \ (CE-op \ opp \ [(V-lit \ (L-num \ n1))]^{ce} \ [(V-lit \ (L-num \ n2))]^{ce}) \rrbracket$ **(is** $\Theta; \mathcal{B}; \Gamma \vdash ?S1 \lesssim ?S2)$

proof(*rule subtype-base-fresh*)

show $atom \ x \ \# \ \Gamma$ **using** *xf* *fresh-Pair* **by** *auto*

show $wfT \ \Theta \ \mathcal{B} \ \Gamma \ (\llbracket x : b \mid CE-val \ (V-var \ x) == CE-val \ (V-lit \ ll) \rrbracket) \text{ (is } wfT \ \Theta \ \mathcal{B} \ ?A \ ?B)$

proof(*rule wfT-e-eq*)

have $\Theta; \mathcal{B}; \Gamma \vdash_{wf} (V-lit \ ll) : b$ **using** *wfV-litI* *base-for-lit.simps* *assms* **by** *metis*

thus $\Theta; \mathcal{B}; \Gamma \vdash_{wf} CE-val \ (V-lit \ ll) : b$ **using** *wfCE-valI* **by** *auto*

show $atom \ x \ \# \ \Gamma$ **using** *xf* *fresh-Pair* **by** *auto*

qed

show $wfT \ \Theta \ \mathcal{B} \ \Gamma \ (\llbracket x : b \mid CE-val \ (V-var \ x) == CE-op \ opp \ [(V-lit \ (L-num \ n1))]^{ce} \ [(V-lit \ (L-num \ n2))]^{ce} \rrbracket) \text{ (is } wfT \ \Theta \ \mathcal{B} \ ?A \ ?C)$

proof(*rule wfT-e-eq,rule opp.exhaust[of opp]*)

{ **assume** $opp = Plus$

thus $\Theta; \mathcal{B}; \Gamma \vdash_{wf} CE-op \ opp \ [(V-lit \ (L-num \ n1))]^{ce} \ [(V-lit \ (L-num \ n2))]^{ce} : b$ **using** *wfCE-valI* *wfCE-plusI* *assms* *wfV-litI* *base-for-lit.simps* *assms* **by** *metis*

}

next

{ **assume** $opp = LEq$

thus $\Theta; \mathcal{B}; \Gamma \vdash_{wf} CE-op \ opp \ [(V-lit \ (L-num \ n1))]^{ce} \ [(V-lit \ (L-num \ n2))]^{ce} : b$ **using** *wfCE-valI* *wfCE-leqI* *assms* *wfV-litI* *base-for-lit.simps* *assms* **by** *metis*

}

show $atom \ x \ \# \ \Gamma$ **using** *xf* *fresh-Pair* **by** *auto*

qed

show $\Theta; \mathcal{B}; (x, b, (CE-val \ (V-var \ x) == CE-val \ (V-lit \ (ll)) \)) \#_{\Gamma} \Gamma$
 $\models (CE-val \ (V-var \ x) == CE-op \ opp \ [(V-lit \ (L-num \ n1))]^{ce} \ [(V-lit \ (L-num \ n2))]^{ce})$
(is $\Theta; \mathcal{B}; ?G \models ?c)$

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 $wfT \ \Theta \ \mathcal{B} \ G \ (\llbracket z : b \mid c \rrbracket)$
shows $\Theta; \mathcal{B}; G \vdash (\llbracket z : b \mid c \rrbracket) \lesssim (\llbracket z : b \mid TRUE \rrbracket)$

proof –

obtain $x::x$ **where** $*$: *atom* $x \# (\Theta, \mathcal{B}, G, z, c, z, TRUE)$ **using** *obtain-fresh* **by** *blast*

then show $?thesis$ **proof**(*rule subtype-baseI*)

show $\langle \Theta; \mathcal{B}; G \vdash_{wf} \llbracket z : b \mid c \rrbracket \rangle$ **using** *assms* **by** *auto*

show $\langle \Theta; \mathcal{B}; G \vdash_{wf} \llbracket z : b \mid TRUE \rrbracket \rangle$ **using** $wfT-TRUE$ *assms* $wfX-wfY$ *b-of.simps* $wfT-wf$

by (*metis wfX-wfB(8)*)

hence $\Theta; \mathcal{B} \vdash_{wf} (x, b, c[z::=V-var \ x]_v) \#_{\Gamma} G$ **using** $wfT-wf-cons3$ *assms* *fresh-Pair* $*$ *subst-v-c-def*
by *auto*

thus $\langle \Theta; \mathcal{B}; (x, b, c[z::=V-var \ x]_v) \#_{\Gamma} G \models (TRUE)[z::=V-var \ x]_v \rangle$ **using** *valid-trueI* *subst-cv.simps* *subst-v-c-def* **by** *metis*

qed

qed

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 $\Theta; \{\mid\}; GNil \vdash \llbracket z : [B-bitvec, B-bitvec]^b \mid [z]^v \rrbracket^{ce} == [[L-bitvec$

$v1]^v, [L-bitvec$

$v2]^v \rrbracket^{ce} \rrbracket \lesssim \llbracket z : [B-bitvec, B-bitvec]^b \mid [[L-bitvec$

$v]^v \rrbracket^{ce} == [[\#1 [[z]^v \rrbracket^{ce}]^{ce} @@ [\#2 [[z]^v \rrbracket^{ce}]^{ce}]^{ce} \ AND \ [[\#1 [[z]^v \rrbracket^{ce}]^{ce}]^{ce} == [$

$[L-num$

$(is \ \Theta; ?B; GNil \vdash \llbracket z : [B-bitvec, B-bitvec]^b \mid ?c1 \rrbracket \lesssim \llbracket z : [B-bitvec, B-bitvec]^b \mid ?c2 \rrbracket)$

proof –

obtain $x::x$ **where** $xf:atom \ x \# (\Theta, ?B, GNil, z, ?c1, z, ?c2)$ **using** *obtain-fresh* **by** *auto*

then show $?thesis$ **proof**(*rule subtype-baseI*)

show $*$: $\langle \Theta; ?B; (x, [B-bitvec, B-bitvec]^b, (?c1)[z::=[x]^v]_v) \#_{\Gamma}$

$GNil \models (?c2)[z::=[x]^v]_v \rangle$

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 $\langle \Theta; ?B; GNil \vdash_{wf} \llbracket z : [B-bitvec, B-bitvec]^b \mid ?c1 \rrbracket \rangle$ **using** *valid-wfT[OF *]* xf *fresh-prodN*

by *metis*

show $\langle \Theta; ?B; GNil \vdash_{wf} \llbracket z : [B-bitvec, B-bitvec]^b \mid ?c2 \rrbracket \rangle$ **using** *valid-wfT[OF *]* xf *fresh-prodN* **by** *metis*

qed

qed

lemma *subtype-range*:

fixes $n::int$ **and** $\Gamma::\Gamma$

assumes $0 \leq n \wedge n \leq int (length\ v)$ **and** $\Theta ; \{\|\} \vdash_{wf} \Gamma$

shows $\Theta ; \{\|\} ; \Gamma \vdash \{ \mid z : B-int \mid [[z]^v]^{ce} == [[L-num\ n]^v]^{ce} \} \lesssim$
 $\{ \mid z : B-int \mid ([leq [[L-num\ 0]^v]^{ce} [[z]^v]^{ce}]^{ce} == [[L-true]^v]^{ce}) \ AND \ ($
 $[leq [[z]^v]^{ce} [[[L-bitvec\ v]^v]^{ce}]^{ce}]^{ce} == [[L-true]^v]^{ce}) \}$
(is $\Theta ; ?B ; \Gamma \vdash \{ \mid z : B-int \mid ?c1 \} \lesssim \{ \mid z : B-int \mid ?c2\ AND\ ?c3 \}$ **)**

proof –

obtain $x::x$ **where** $*(atom\ x \# (\Theta, ?B, \Gamma, z, ?c1, z, ?c2\ AND\ ?c3))$ **using** *obtain-fresh* **by** *auto*

moreover have $*(\Theta ; ?B ; (x, B-int, (?c1)[z::=[x]^v_v) \#_{\Gamma} \Gamma \models (?c2\ AND\ ?c3)[z::=[x]^v_v])$

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 $\langle \Theta ; ?B ; \Gamma \vdash_{wf} \{ \mid z : B-int \mid [[z]^v]^{ce} == [[L-num\ n]^v]^{ce} \} \rangle$ **using**
*valid-wfT * fresh-prodN* **by** *metis*

moreover have $\langle \Theta ; ?B ; \Gamma \vdash_{wf} \{ \mid z : B-int \mid ?c2\ AND\ ?c3 \} \rangle$

using *valid-wfT[OF **] * fresh-prodN* **by** *metis*

ultimately show *?thesis* **using** *subtype-baseI* **by** *auto*

qed

lemma *check-num-range*:

assumes $0 \leq n \wedge n \leq int (length\ v)$ **and** $\vdash_{wf} \Theta$

shows $\Theta ; \{\|\} ; GNil \vdash ([L-num\ n]^v) \Leftarrow \{ \mid z : B-int \mid ([leq [[L-num\ 0]^v]^{ce} [[z]^v]^{ce}]^{ce} ==$
 $[[L-true]^v]^{ce}) \ AND$
 $[leq [[z]^v]^{ce} [[[L-bitvec\ v]^v]^{ce}]^{ce}]^{ce} == [[L-true]^v]^{ce} \}$

using *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*) = $(\{ \mid z : B-bool \mid [[z]^v]^{ce} == [V-lit\ L-true]^{ce} \})$

| *type-for-lit* (*L-false*) = $(\{ \mid z : B-bool \mid [[z]^v]^{ce} == [V-lit\ L-false]^{ce} \})$

| *type-for-lit* (*L-num n*) = $(\{ \mid z : B-int \mid [[z]^v]^{ce} == [V-lit\ (L-num\ n)]^{ce} \})$

| *type-for-lit* (*L-unit*) = $(\{ \mid z : B-unit \mid [[z]^v]^{ce} == [V-lit\ (L-unit)]^{ce} \})$

| *type-for-lit* (*L-bitvec v*) = $(\{ \mid z : B-bitvec \mid [[z]^v]^{ce} == [V-lit\ (L-bitvec\ v)]^{ce} \})$

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 $G\ \tau\ x = (case\ lookup\ G\ x\ of$

None $\Rightarrow \tau$

| *Some* (*b,c*) $\Rightarrow (\{ \mid x : b \mid c \})$)

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 $\vdash l \Rightarrow \tau$
shows $\exists z b. \tau = (\llbracket z : b \mid C\text{-eq } (CE\text{-val } (V\text{-var } z)) (CE\text{-val } (V\text{-lit } l)) \rrbracket) \wedge \text{atom } z \# tm$
proof –
obtain z' **and** b **where** $t:\tau = (\llbracket z' : b \mid C\text{-eq } (CE\text{-val } (V\text{-var } z')) (CE\text{-val } (V\text{-lit } l)) \rrbracket)$ **using**
infer-l-elim *assms* **using** *infer-l.simps* *type-for-lit.simps*
type-for-lit.cases **by** *blast*
obtain $z::x$ **where** $zf: \text{atom } z \# tm$ **using** *obtain-fresh* **by** *metis*
have $\tau = \llbracket z : b \mid C\text{-eq } (CE\text{-val } (V\text{-var } z)) (CE\text{-val } (V\text{-lit } l)) \rrbracket$ **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 $\vdash l \Rightarrow \tau$
shows $\exists z. \tau = (\llbracket z : \text{base-for-lit } l \mid C\text{-eq } (CE\text{-val } (V\text{-var } z)) (CE\text{-val } (V\text{-lit } l)) \rrbracket)$
using *infer-l-elim* **using** *assms* **using** *infer-l.simps* *type-for-lit.simps* *base-for-lit.simps* **by** *auto*

lemma *infer-l-form4*[*simp*]:
fixes $\Gamma::\Gamma$
assumes $\Theta ; \mathcal{B} \vdash_{wf} \Gamma$
shows $\exists z. \vdash l \Rightarrow (\llbracket z : \text{base-for-lit } l \mid C\text{-eq } (CE\text{-val } (V\text{-var } z)) (CE\text{-val } (V\text{-lit } l)) \rrbracket)$
using *assms* *infer-l-form2* *infer-l-form3* **by** *metis*

lemma *infer-v-unit-form*:
fixes $v::v$
assumes $P ; \mathcal{B} ; \Gamma \vdash v \Rightarrow (\llbracket z1 : B\text{-unit} \mid c1 \rrbracket)$ **and** $\text{supp } v = \{\}$
shows $v = V\text{-lit } L\text{-unit}$
using *assms* **proof**(*nominal-induct* $\Gamma v \llbracket z1 : B\text{-unit} \mid c1 \rrbracket$ *rule: infer-v.strong-induct*)
case (*infer-v-varI* $\Theta \mathcal{B} c x z$)
then show *?case* **using** *supp-at-base* **by** *auto*
next
case (*infer-v-litI* $\Theta \mathcal{B} \Gamma l$)
from $\vdash l \Rightarrow \llbracket z1 : B\text{-unit} \mid c1 \rrbracket$ **show** *?case* **by**(*nominal-induct* $\llbracket z1 : B\text{-unit} \mid c1 \rrbracket$ *rule:*
infer-l.strong-induct,auto)
qed

lemma *base-for-lit-wf*:
assumes $\vdash_{wf} \Theta$
shows $\Theta ; \mathcal{B} \vdash_{wf} \text{base-for-lit } l$
using *base-for-lit.simps* **using** *wfV-elim* *wf-intros* *assms* *l.exhaust* **by** *metis*

lemma *infer-l-t-wf*:
fixes $\Gamma::\Gamma$
assumes $\Theta ; \mathcal{B} \vdash_{wf} \Gamma \wedge \text{atom } z \# \Gamma$
shows $\Theta ; \mathcal{B} ; \Gamma \vdash_{wf} \llbracket z : \text{base-for-lit } l \mid C\text{-eq } (CE\text{-val } (V\text{-var } z)) (CE\text{-val } (V\text{-lit } l)) \rrbracket$
proof

show $\text{atom } z \# (\Theta, \mathcal{B}, \Gamma)$ **using** $\text{wfG-fresh-}x \text{ assms}$ **by** auto
show $\Theta ; \mathcal{B} \vdash_{\text{wf}} \text{base-for-lit } l$ **using** $\text{base-for-lit-wf assms wfX-wfY}$ **by** metis
thus $\Theta ; \mathcal{B} ; (z, \text{base-for-lit } l, \text{TRUE}) \#_{\Gamma} \Gamma \vdash_{\text{wf}} \text{CE-val } (V\text{-var } z) == \text{CE-val } (V\text{-lit } l)$ **using**
 $\text{wfC-v-eq wfV-litI assms wfX-wfY}$ **by** metis
qed

lemma infer-l-wf :
fixes $l::l$ **and** $\Gamma::\Gamma$ **and** $\tau::\tau$ **and** $\Theta::\Theta$
assumes $\vdash l \Rightarrow \tau$ **and** $\Theta ; \mathcal{B} \vdash_{\text{wf}} \Gamma$
shows $\vdash_{\text{wf}} \Theta$ **and** $\Theta ; \mathcal{B} \vdash_{\text{wf}} \Gamma$ **and** $\Theta ; \mathcal{B} ; \Gamma \vdash_{\text{wf}} \tau$
proof –
show $\Theta ; \mathcal{B} \vdash_{\text{wf}} \Gamma$ **using** $\text{assms infer-l-elim}$ **by** auto
thus $\vdash_{\text{wf}} \Theta$ **using** wfX-wfY **by** auto
show $\Theta ; \mathcal{B} ; \Gamma \vdash_{\text{wf}} \tau$ **using** $\text{infer-l-t-wf assms infer-l-form3}$ *
by $(\text{metis } \langle \vdash_{\text{wf}} \Theta \rangle \text{ fresh-GNil wfG-nilI wfT-weakening-nil})$
qed

lemma $\text{infer-l-uniqueness}$:
fixes $l::l$
assumes $\vdash l \Rightarrow \tau$ **and** $\vdash l \Rightarrow \tau'$
shows $\tau = \tau'$
using assms
proof –
obtain z **and** b **where** $z\text{t}: \tau = (\llbracket z : b \mid C\text{-eq } (\text{CE-val } (V\text{-var } z)) (\text{CE-val } (V\text{-lit } l)) \rrbracket)$ **using**
 $\text{infer-l-form assms}$ **by** blast
obtain z' **and** b **where** $z't: \tau' = (\llbracket z' : b \mid C\text{-eq } (\text{CE-val } (V\text{-var } z')) (\text{CE-val } (V\text{-lit } l)) \rrbracket)$ **using**
 $\text{infer-l-form assms}$ **by** blast
thus $?thesis$ **using** $\text{type-l-eq } z\text{t } z't \text{ assms infer-l.simps infer-l-elim } l.\text{distinct}$
by $(\text{metis infer-l-form3})$
qed

12.3 Values

lemma type-v-eq :
assumes $\llbracket z1 : b1 \mid c1 \rrbracket = \llbracket z : b \mid C\text{-eq } (\text{CE-val } (V\text{-var } z)) (\text{CE-val } (V\text{-var } x)) \rrbracket$ **and** $\text{atom } z \# x$
shows $b = b1$ **and** $c1 = C\text{-eq } (\text{CE-val } (V\text{-var } z1)) (\text{CE-val } (V\text{-var } x))$
using assms **by** $(\text{auto,metis Abs1-eq-iff } \tau.\text{eq-iff assms c.fresh ce.fresh type-e-eq v.fresh})$

lemma infer-var2 [elim] :
assumes $P ; \mathcal{B} ; G \vdash V\text{-var } x \Rightarrow \tau$
shows $\exists b \ c. \text{Some } (b,c) = \text{lookup } G \ x$
using $\text{assms infer-v-elim lookup-iff}$ **by** $(\text{metis (no-types, lifting)})$

lemma infer-var3 [elim] :
assumes $\Theta ; \mathcal{B} ; \Gamma \vdash V\text{-var } x \Rightarrow \tau$
shows $\exists z \ b \ c. \text{Some } (b,c) = \text{lookup } \Gamma \ x \wedge \tau = (\llbracket z : b \mid C\text{-eq } (\text{CE-val } (V\text{-var } z)) (\text{CE-val } (V\text{-var } x)) \rrbracket) \wedge \text{atom } z \# x \wedge \text{atom } z \# (\Theta, \mathcal{B}, \Gamma)$
using $\text{infer-v-elim}(1)[\text{OF assms}(1)]$ **by** metis

lemma $\text{infer-bool-options2}$:
fixes $v::v$
assumes $\Theta ; \mathcal{B} ; \Gamma \vdash v \Rightarrow \llbracket z : b \mid c \rrbracket$ **and** $\text{supp } v = \{\} \wedge b = B\text{-bool}$


```

  shows  $v = V\text{-lit } L\text{-true} \vee (v = (V\text{-lit } L\text{-false}))$ 
  using assms
proof(nominal-induct  $\{ z : b \mid c \}$  rule: infer-v.strong-induct)
  case (infer-v-varI  $\Theta \mathcal{B} \Gamma c x z$ )
  then show ?case using v.supp supp-at-base by auto
next
  case (infer-v-litI  $\Theta \mathcal{B} \Gamma l$ )
  from  $\langle \vdash l \Rightarrow \{ z : b \mid c \} \rangle$  show ?case proof(nominal-induct  $\{ z : b \mid c \}$  rule: infer-l.strong-induct)
    case (infer-trueI  $z$ )
    then show ?case by auto
  next
    case (infer-falseI  $z$ )
    then show ?case by auto
  next
    case (infer-natI  $n z$ )
    then show ?case using infer-v-litI by simp
  next
    case (infer-unitI  $z$ )
    then show ?case using infer-v-litI by simp
  next
    case (infer-bitvecI  $bv z$ )
    then show ?case using infer-v-litI by simp
qed
qed(auto+)

```

lemma *infer-bool-options*:

```

  fixes  $v::v$ 
  assumes  $\Theta; \mathcal{B}; \Gamma \vdash v \Rightarrow \{ z : B\text{-bool} \mid c \}$  and  $\text{supp } v = \{ \}$ 
  shows  $v = V\text{-lit } L\text{-true} \vee (v = (V\text{-lit } L\text{-false}))$ 
  using infer-bool-options2 assms by blast

```

lemma *infer-int2*:

```

  fixes  $v::v$ 
  assumes  $\Theta; \mathcal{B}; \Gamma \vdash v \Rightarrow \{ z : b \mid c \}$ 
  shows  $\text{supp } v = \{ \} \wedge b = B\text{-int} \longrightarrow (\exists n. v = V\text{-lit } (L\text{-num } n))$ 
  using assms
proof(nominal-induct  $\{ z : b \mid c \}$  rule: infer-v.strong-induct)
  case (infer-v-varI  $\Theta \mathcal{B} \Gamma c x z$ )
  then show ?case using v.supp supp-at-base by auto
next
  case (infer-v-litI  $\Theta \mathcal{B} \Gamma l$ )
  from  $\langle \vdash l \Rightarrow \{ z : b \mid c \} \rangle$  show ?case proof(nominal-induct  $\{ z : b \mid c \}$  rule: infer-l.strong-induct)
    case (infer-trueI  $z$ )
    then show ?case by auto
  next
    case (infer-falseI  $z$ )
    then show ?case by auto
  next
    case (infer-natI  $n z$ )
    then show ?case using infer-v-litI by simp
  next

```

```

    case (infer-unitI z)
    then show ?case using infer-v-litI by simp
next
    case (infer-bitvecI bv z)
    then show ?case using infer-v-litI by simp
qed
qed(auto+)

```

```

lemma infer-bitvec:
  fixes  $\Theta::\Theta$  and  $v::v$ 
  assumes  $\Theta; \mathcal{B}; \Gamma \vdash v \Rightarrow \llbracket z' : B\text{-bitvec} \mid c' \rrbracket$  and  $\text{supp } v = \{\}$ 
  shows  $\exists bv. v = V\text{-lit } (L\text{-bitvec } bv)$ 
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-elim( $\gamma$ )[OF V-consp(2)]  $\tau.\text{eq-iff}$  by auto
next
  case (V-var x)
  then show ?case using supp-at-base by auto
qed(force+)

```

```

lemma infer-int:
  assumes infer-v  $\Theta \mathcal{B} \Gamma v \ (\llbracket z : B\text{-int} \mid c \rrbracket)$  and  $\text{supp } v = \{\}$ 
  shows  $\exists n. V\text{-lit } (L\text{-num } n) = v$ 
  using assms infer-int2 by (metis (no-types, lifting))

```

```

lemma infer-v-form[simp]:
  fixes  $v::v$ 
  assumes  $\Theta; \mathcal{B}; \Gamma \vdash v \Rightarrow \tau$ 
  shows  $\exists z b. \tau = (\llbracket z : b \mid C\text{-eq } (CE\text{-val } (V\text{-var } z)) (CE\text{-val } v) \rrbracket) \wedge \text{atom } z \# v \wedge \text{atom } z \# (\Theta, \mathcal{B}, \Gamma)$ 
  using assms
proof(nominal-induct rule: infer-v.strong-induct)
  case (infer-v-varI  $\Theta \mathcal{B} \Gamma b c x z$ )
  then show ?case by force
next
  case (infer-v-litI  $\Theta \mathcal{B} \Gamma l \tau$ )
  then obtain z and b where  $\tau = \llbracket z : b \mid CE\text{-val } (V\text{-var } z) == CE\text{-val } (V\text{-lit } l) \rrbracket \wedge \text{atom } z \# (\Theta, \mathcal{B}, \Gamma)$ 
  using infer-l-form by metis
  moreover hence  $\text{atom } z \# (V\text{-lit } l)$  using supp-l-empty v.fresh(1) fresh-prod2 fresh-def by blast
  ultimately show ?case by metis
next
  case (infer-v-pairI  $z v1 v2 \Theta \mathcal{B} \Gamma t1 t2$ )
  then show ?case by force
next
  case (infer-v-consI  $s dclist \Theta dc tc \mathcal{B} \Gamma v tv z$ )
  moreover hence  $\text{atom } z \# (V\text{-cons } s dc v)$  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 using fresh-prodN by metis
next

```

case (*infer-v-conspI* *s bv dclist* Θ *dc tc* \mathcal{B} Γ *v tv b z*)
 moreover hence $\text{atom } z \# (V\text{-consp } s \text{ dc } b \text{ v})$ **unfolding** *v.fresh* **using** *pure-fresh fresh-prodN* * **by** *metis*
 ultimately show *?case* **using** *fresh-prodN* **by** *metis*
qed

lemma *infer-v-form2*:

fixes *v::v*
 assumes $\Theta; \mathcal{B}; \Gamma \vdash v \Rightarrow (\llbracket z : b \mid c \rrbracket)$ **and** $\text{atom } z \# v$
 shows $c = C\text{-eq } (CE\text{-val } (V\text{-var } z)) (CE\text{-val } v)$
using *assms*
proof –
 obtain *z'* and *b'* where $(\llbracket z : b \mid c \rrbracket) = (\llbracket z' : b' \mid CE\text{-val } (V\text{-var } z') \rrbracket == CE\text{-val } v \rrbracket) \wedge \text{atom } z' \# v$
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 $\Theta; \mathcal{B}; \Gamma \vdash v \Rightarrow \tau$ **and** $\text{atom } z \# (v, \Gamma)$
 shows $\Theta; \mathcal{B}; \Gamma \vdash v \Rightarrow \llbracket z : b\text{-of } \tau \mid C\text{-eq } (CE\text{-val } (V\text{-var } z)) (CE\text{-val } v) \rrbracket$
proof –
 obtain *z'* and *b'* where $\tau = \llbracket z' : b' \mid C\text{-eq } (CE\text{-val } (V\text{-var } z')) (CE\text{-val } v) \rrbracket \wedge \text{atom } z' \# v \wedge \text{atom } z' \# (\Theta, \mathcal{B}, \Gamma)$
using *infer-v-form* *assms* **by** *metis*
 moreover hence $\llbracket z' : b' \mid C\text{-eq } (CE\text{-val } (V\text{-var } z')) (CE\text{-val } v) \rrbracket = \llbracket z : b' \mid C\text{-eq } (CE\text{-val } (V\text{-var } z)) (CE\text{-val } v) \rrbracket$
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 $\Theta; \mathcal{B}; \Gamma \vdash v \Rightarrow \tau$ **and** $\text{atom } z \# (v, \Gamma)$ **and** $b = b\text{-of } \tau$
 shows $\Theta; \mathcal{B}; \Gamma \vdash v \Rightarrow \llbracket z : b \mid C\text{-eq } (CE\text{-val } (V\text{-var } z)) (CE\text{-val } v) \rrbracket$
using *assms* *infer-v-form3* **by** *simp*

lemma *infer-v-v-wf*:

fixes *v::v*
 shows $\Theta; \mathcal{B}; G \vdash v \Rightarrow \tau \Longrightarrow \Theta; \mathcal{B}; G \vdash_{wf} v : (b\text{-of } \tau)$
proof(*induct* *rule: infer-v.induct*)
 case (*infer-v-varI* $\Theta \mathcal{B} \Gamma b c x z$)
 then show *?case* **using** *wfC-elim* *wf-intros* **by** *auto*
next
 case (*infer-v-pairI* *z v1 v2* $\Theta \mathcal{B} \Gamma t1 t2$)
 then show *?case* **using** *wfC-elim* *wf-intros* **by** *auto*
next
 case (*infer-v-litI* $\Theta \mathcal{B} \Gamma l \tau$)
 hence $b\text{-of } \tau = \text{base-for-lit } l$ **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-consI s dclist Θ dc tc \mathcal{B} Γ v tv z)
 then show ?case using wfC-elim wf-intros
 by (metis (no-types, lifting) b-of.simps has-fresh-z2 subtype-eq-base2)
 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 = $\llbracket z1 : b1 \mid c1 \rrbracket$ using obtain-fresh-z by metis
 show ?case unfolding b-of.simps proof(rule wfV-conspI)
 show $\langle AF\text{-typedef-poly } s \text{ bv dclist} \in \text{set } \Theta \rangle$ using infer-v-conspI by auto
 show $\langle (dc, \llbracket z1 : b1 \mid c1 \rrbracket) \in \text{set dclist} \rangle$ using infer-v-conspI t by auto
 show $\langle \Theta ; \mathcal{B} \vdash_{wf} b \rangle$ using infer-v-conspI by auto
 show $\langle \text{atom bv} \# (\Theta, \mathcal{B}, \Gamma, b, v) \rangle$ 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 $\langle \Theta ; \mathcal{B} ; \Gamma \vdash_{wf} v : b1[bv::=b]_{bb} \rangle$ using infer-v-conspI by auto
 qed
 qed

 lemma infer-v-t-form-wf:
 assumes wfB Θ \mathcal{B} b and wfV Θ \mathcal{B} Γ v b and atom z $\# \Gamma$
 shows wfT Θ \mathcal{B} Γ $\llbracket z : b \mid C\text{-eq } (CE\text{-val } (V\text{-var } z)) (CE\text{-val } v) \rrbracket$
 using wfT-v-eq assms by auto

 lemma infer-v-t-wf:
 fixes v::v
 assumes $\Theta ; \mathcal{B} ; G \vdash v \Rightarrow \tau$
 shows wfT Θ \mathcal{B} G $\tau \wedge$ wfB Θ \mathcal{B} (b-of τ)
 proof –
 obtain z and b where $\tau = \llbracket z : b \mid CE\text{-val } (V\text{-var } z) == CE\text{-val } v \rrbracket \wedge \text{atom } z \# v \wedge \text{atom } z \#$
 (Θ, \mathcal{B}, 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 $\tau \wedge$ 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 $\Theta ; \mathcal{B} ; G \vdash v \Rightarrow \tau$
 shows $\Theta ; \mathcal{B} ; G \vdash_{wf} v : (b\text{-of } \tau)$ and wfT Θ \mathcal{B} G τ and wfTh Θ and wfG Θ \mathcal{B} G
 proof –
 show $\Theta ; \mathcal{B} ; G \vdash_{wf} v : b\text{-of } \tau$ using infer-v-v-wf assms by auto
 show $\Theta ; \mathcal{B} ; G \vdash_{wf} \tau$ using infer-v-t-wf assms by auto
 thus $\Theta ; \mathcal{B} \vdash_{wf} G$ using wfX-wfY by auto
 thus $\vdash_{wf} \Theta$ using wfX-wfY by auto
 qed

 lemma check-bool-options:
 assumes $\Theta ; \mathcal{B} ; \Gamma \vdash v \Leftarrow \llbracket z : B\text{-bool} \mid TRUE \rrbracket$ and supp v = {}
 shows v = V-lit L-true \vee v = V-lit L-false
 proof –

obtain $t1$ **where** $\Theta; \mathcal{B}; \Gamma \vdash t1 \lesssim \llbracket z : B\text{-bool} \mid TRUE \rrbracket \wedge \Theta; \mathcal{B}; \Gamma \vdash v \Rightarrow t1$ **using** *check-v-elim*

using *assms* **by** *blast*

thus *?thesis* **using** *infer-bool-options* *assms*

by (*metis* $\tau.\text{exhaust } b\text{-of.simps subtype-eq-base2}$)

qed

lemma *check-v-wf*:

fixes $v::v$ **and** $\Gamma::\Gamma$ **and** $\tau::\tau$

assumes $\Theta; \mathcal{B}; \Gamma \vdash v \Leftarrow \tau$

shows $\Theta; \mathcal{B} \vdash_{wf} \Gamma$ **and** $\Theta; \mathcal{B}; \Gamma \vdash_{wf} v : b\text{-of } \tau$ **and** $\Theta; \mathcal{B}; \Gamma \vdash_{wf} \tau$

proof –

obtain τ' **where** $\ast; \Theta; \mathcal{B}; \Gamma \vdash \tau' \lesssim \tau \wedge \Theta; \mathcal{B}; \Gamma \vdash v \Rightarrow \tau'$ **using** *check-v-elim* *assms* **by** *auto*

thus $\Theta; \mathcal{B} \vdash_{wf} \Gamma$ **and** $\Theta; \mathcal{B}; \Gamma \vdash_{wf} v : b\text{-of } \tau$ **and** $\Theta; \mathcal{B}; \Gamma \vdash_{wf} \tau$

using *infer-v-wf infer-v-v-wf subtype-eq-base2* \ast *subtype-wf* **by** *metis+*

qed

lemma *infer-v-form-fresh*:

fixes $v::v$ **and** $t::'a::fs$

assumes $\Theta; \mathcal{B}; \Gamma \vdash v \Rightarrow \tau$

shows $\exists z b. \tau = \llbracket z : b \mid C\text{-eq } (CE\text{-val } (V\text{-var } z)) (CE\text{-val } v) \rrbracket \wedge \text{atom } z \# (t, v)$

proof –

obtain z' **and** b' **where** $\tau = \llbracket z' : b' \mid C\text{-eq } (CE\text{-val } (V\text{-var } z')) (CE\text{-val } v) \rrbracket$ **using** *infer-v-form* *assms* **by** *blast*

moreover then obtain z **and** b **and** c **where** $\tau = \llbracket z : b \mid c \rrbracket \wedge \text{atom } z \# (t, v)$ **using** *obtain-fresh-z* **by** *metis*

ultimately have $\tau = \llbracket z : b \mid C\text{-eq } (CE\text{-val } (V\text{-var } z)) (CE\text{-val } v) \rrbracket \wedge \text{atom } z \# (t, v)$

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 $\Theta; \mathcal{B}; \Gamma \vdash V\text{-consp } s \text{ dc } b \text{ } v \Rightarrow \tau$

shows $b\text{-of } \tau = B\text{-app } s \text{ } b$

using *assms* **proof**(*nominal-induct* $V\text{-consp } s \text{ dc } b \text{ } v \text{ } \tau$ *rule: infer-v.strong-induct*)

case (*infer-v-conspI* $bv \text{ dclist } \Theta \text{ tc } \mathcal{B} \Gamma \text{ tv } z$)

then show *?case* **using** *b-of.simps* **by** *metis*

qed

lemma *lookup-in-rig-b*:

assumes $\text{Some } (b2, c2) = \text{lookup } (\Gamma[x \mapsto c']) \text{ } x'$ **and**

$\text{Some } (b1, c1) = \text{lookup } \Gamma \text{ } x'$

shows $b1 = b2$

using *assms lookup-in-rig[OF assms(2)]*

by (*metis option.inject prod.inject*)

lemma *infer-v-uniqueness-rig*:

fixes $x::x$ **and** $c::c$

assumes *infer-v* $P \text{ } B \text{ } G \text{ } v \text{ } \tau$ **and** *infer-v* $P \text{ } B \text{ } (\text{replace-in-g } G \text{ } x \text{ } c') \text{ } v \text{ } \tau'$

shows $\tau = \tau'$
using *assms*
proof(*nominal-induct v arbitrary: $\tau' \tau$ rule: $v.\text{strong-induct}$*)
case (*V-lit l*)
hence *infer-l l τ and infer-l l τ'* **using** *assms(1) infer-v-elim(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-elim(2)* **using** *V-var.prem(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** $\tau = (\llbracket z : b \mid C\text{-eq} (CE\text{-val} (V\text{-var } z)) (CE\text{-val} (V\text{-var } y)) \rrbracket)$ **using** *infer-v-elim(1)[of P B G y τ] V-var*
bc option.inject prod.inject lookup-in-g **by** *metis*
moreover obtain *z'* **where** $\tau' = (\llbracket z' : b \mid C\text{-eq} (CE\text{-val} (V\text{-var } z')) (CE\text{-val} (V\text{-var } y)) \rrbracket)$ **using** *infer-v-elim(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.prem(1) V-var.prem(2) $\tau.\text{eq-iff}$ ce.fresh(1) finite.emptyI fresh-atom-at-base fresh-finite-insert infer-v-elim(1) v.fresh(2))*
next
case (*V-pair v1 v2*)
obtain *z* **and** *z1* **and** *z2* **and** *t1* **and** *t2* **and** *c1* **and** *c2* **where**
 $t1: \tau = (\llbracket z : [b\text{-of } t1, b\text{-of } t2]^b \mid CE\text{-val} (V\text{-var } z) == CE\text{-val} (V\text{-pair } v1 v2) \rrbracket) \wedge$
 $atom\ z \# (v1, v2) \wedge P ; B ; G \vdash v1 \Rightarrow t1 \wedge P ; B ; G \vdash v2 \Rightarrow t2$
using *infer-v-elim(3)[OF V-pair(3)]* **by** *metis*
moreover obtain *z'* **and** *z1'* **and** *z2'* **and** *t1'* **and** *t2'* **and** *c1'* **and** *c2'* **where**
 $t2: \tau' = (\llbracket z' : [b\text{-of } t1', b\text{-of } t2']^b \mid CE\text{-val} (V\text{-var } z') == CE\text{-val} (V\text{-pair } v1 v2) \rrbracket) \wedge$
 $atom\ z' \# (v1, v2) \wedge P ; B ; (replace\text{-in-g } G\ x\ c') \vdash v1 \Rightarrow t1' \wedge$
 $P ; B ; (replace\text{-in-g } G\ x\ c') \vdash v2 \Rightarrow t2'$
using *infer-v-elim(3)[OF V-pair(4)]* **by** *metis*
ultimately have $t1 = t1' \wedge t2 = t2'$ **using** *V-pair.hyps(1) V-pair.hyps(2) $\tau.\text{eq-iff}$* **by** *blast*
then show *?case using t1 t2* **by** *simp*
next
case (*V-cons s dc v*)
obtain *x* **and** *z* **and** *tc* **and** *dclist* **where** $t1: \tau = (\llbracket z : B\text{-id } s \mid CE\text{-val} (V\text{-var } z) == CE\text{-val} (V\text{-cons } s\ dc\ v) \rrbracket) \wedge$
 $AF\text{-typedef } s\ dclist \in set\ P \wedge$
 $(dc, tc) \in set\ dclist \wedge atom\ z \# v$
using *infer-v-elim(4)[OF V-cons(2)]* **by** *metis*
moreover obtain *x'* **and** *z'* **and** *tc'* **and** *dclist'* **where** $t2: \tau' = (\llbracket z' : B\text{-id } s \mid CE\text{-val} (V\text{-var } z') == CE\text{-val} (V\text{-cons } s\ dc\ v) \rrbracket)$
 $\wedge AF\text{-typedef } s\ dclist' \in set\ P \wedge (dc, tc') \in set\ dclist' \wedge atom\ z' \# v$
using *infer-v-elim(4)[OF V-cons(3)]* **by** *metis*
moreover have *a: AF-typedef s dclist' \in set P* **and** *b:(dc,tc') \in set dclist'* **and** *c:AF-typedef s dclist \in set P* **and**
 $d:(dc, tc) \in set\ dclist$ **using** *t1 t2* **by** *auto*
ultimately have $tc = tc'$ **using** *wfTh-dc-t-unique2 infer-v-wf(3)[OF V-cons(2)]* **by** *metis*

```

moreover have  $atom\ z \# CE\text{-}val\ (V\text{-}cons\ s\ dc\ v) \wedge atom\ z' \# CE\text{-}val\ (V\text{-}cons\ s\ dc\ v)$ 
  using  $e.fresh(1)\ v.fresh(4)\ t1\ t2\ pure\text{-}fresh$  by  $auto$ 
ultimately have  $(\llbracket z : B\text{-}id\ s \mid CE\text{-}val\ (V\text{-}var\ z) == CE\text{-}val\ (V\text{-}cons\ s\ dc\ v) \rrbracket) = (\llbracket z' : B\text{-}id\ s \mid CE\text{-}val\ (V\text{-}var\ z') == CE\text{-}val\ (V\text{-}cons\ s\ dc\ v) \rrbracket)$ 
  using  $type\text{-}e\text{-}eq$  by  $metis$ 
thus  $?case$  using  $t1\ t2$  by  $simp$ 
next
  case  $(V\text{-}consp\ s\ dc\ b\ v)$ 
    from  $V\text{-}consp(2)\ V\text{-}consp$  show  $?case$  proof( $nominal\text{-}induct\ V\text{-}consp\ s\ dc\ b\ v\ \tau\ arbitrary: v$ 
 $rule:infer\text{-}v.\text{strong}\text{-}induct$ )

      case  $(infer\text{-}v.\text{conspI}\ bv\ dclist\ \Theta\ tc\ \mathcal{B}\ \Gamma\ v\ tv\ z)$ 

        obtain  $z3$  and  $b3$  where  $*:\tau' = \llbracket z3 : b3 \mid \llbracket z3 \rrbracket^v \rrbracket^{ce} == \llbracket V\text{-}consp\ s\ dc\ b\ v \rrbracket^{ce} \rrbracket \wedge atom\ z3 \# V\text{-}consp\ s\ dc\ b\ v$ 
          using  $infer\text{-}v.\text{form}[OF\ \langle\Theta; \mathcal{B}; \Gamma[x \mapsto c] \vdash V\text{-}consp\ s\ dc\ b\ v \Rightarrow \tau'\rangle]$  by  $metis$ 
          moreover then have  $b3 = B\text{-}app\ s\ b$  using  $infer\text{-}v.\text{form}\text{-}consp\ b\text{-}of.\text{simps} * infer\text{-}v.\text{conspI}$  by  $metis$ 

          moreover have  $\llbracket z3 : B\text{-}app\ s\ b \mid \llbracket z3 \rrbracket^v \rrbracket^{ce} == \llbracket V\text{-}consp\ s\ dc\ b\ v \rrbracket^{ce} \rrbracket = \llbracket z : B\text{-}app\ s\ b \mid \llbracket z \rrbracket^v \rrbracket^{ce} == \llbracket V\text{-}consp\ s\ dc\ b\ v \rrbracket^{ce} \rrbracket$ 
            proof –
              have  $atom\ z3 \# \llbracket V\text{-}consp\ s\ dc\ b\ v \rrbracket^{ce}$  using  $*\ ce.fresh$  by  $auto$ 
              moreover have  $atom\ z \# \llbracket V\text{-}consp\ s\ dc\ b\ v \rrbracket^{ce}$  using  $*\ infer\text{-}v.\text{conspI}\ ce.fresh\ v.fresh\ pure\text{-}fresh$ 
by  $metis$ 
              ultimately show  $?thesis$  using  $type\text{-}e\text{-}eq\ infer\text{-}v.\text{conspI}\ v.fresh\ ce.fresh$  by  $metis$ 
              qed
              ultimately show  $?case$  using  $*$  by  $auto$ 
              qed
            qed
          qed

lemma  $infer\text{-}v.\text{uniqueness}$ :
  assumes  $infer\text{-}v\ P\ B\ G\ v\ \tau$  and  $infer\text{-}v\ P\ B\ G\ v\ \tau'$ 
  shows  $\tau = \tau'$ 
proof –
  obtain  $x::x$  where  $atom\ x \# G$  using  $obtain\text{-}fresh$  by  $metis$ 
  hence  $G\ [x \mapsto C\text{-}true] = G$  using  $replace\text{-}in\text{-}g\text{-}forget\ assms\ infer\text{-}v.\text{wf}$  by  $fast$ 
  thus  $?thesis$  using  $infer\text{-}v.\text{uniqueness}\text{-}rig\ assms$  by  $metis$ 
qed

lemma  $infer\text{-}v.\text{tid}\text{-}form$ :
  fixes  $v::v$ 
  assumes  $\Theta ; B ; \Gamma \vdash v \Rightarrow \llbracket z : B\text{-}id\ tid \mid c \rrbracket$  and  $AF\text{-}typedef\ tid\ dclist \in set\ \Theta$  and  $supp\ v = \{\}$ 
  shows  $\exists dc\ v'\ t. v = V\text{-}cons\ tid\ dc\ v' \wedge (dc, t) \in set\ dclist$ 
using  $assms$  proof( $nominal\text{-}induct\ v\ \llbracket z : B\text{-}id\ tid \mid c \rrbracket\ rule: infer\text{-}v.\text{strong}\text{-}induct$ )
  case  $(infer\text{-}v.\text{varI}\ \Theta\ \mathcal{B}\ c\ x\ z)$ 
  then show  $?case$  using  $v.\text{supp}\ supp\text{-}at\text{-}base$  by  $auto$ 
next
  case  $(infer\text{-}v.\text{litI}\ \Theta\ \mathcal{B}\ l)$ 
  then show  $?case$  by  $auto$ 
next

```

case (infer-v-consI dclist1 Θ dc tc \mathcal{B} Γ v tv z)
 hence $\text{supp } v = \{\}$ using v.supp by simp
 then obtain dca and v' where *: $V\text{-cons tid dc } v = V\text{-cons tid dca } v'$ using infer-v-consI by auto
 hence $dca = dc$ using v.eq-iff(4) by auto
 hence $V\text{-cons tid dc } v = V\text{-cons tid dca } v' \wedge (dca, tc) \in \text{set dclist1}$ using infer-v-consI * by auto
 moreover have $dclist = dclist1$ using wfTh-dclist-unique infer-v-consI wfX-wfY $\langle dca=dc \rangle$
 proof –
 show ?thesis
 by (meson $\langle AF\text{-typedef tid dclist1} \in \text{set } \Theta \rangle \langle \Theta; \mathcal{B}; \Gamma \vdash v \Rightarrow tv \rangle$ infer-v-consI.premis infer-v-wf(4)
 wfTh-dclist-unique wfX-wfY)
 qed
 ultimately show ?case by auto
 qed

lemma check-v-tid-form:

assumes $\Theta; \mathcal{B}; \Gamma \vdash v \Leftarrow \llbracket z : B\text{-id tid} \mid TRUE \rrbracket$ and $AF\text{-typedef tid dclist} \in \text{set } \Theta$ and $\text{supp } v = \{\}$
 shows $\exists dc \ v' \ t. v = V\text{-cons tid dc } v' \wedge (dc, t) \in \text{set dclist}$
 using assms proof(nominal-induct v $\llbracket z : B\text{-id tid} \mid TRUE \rrbracket$ rule: check-v.strong-induct)
 case (check-v-subtypeI $\Theta \mathcal{B} \Gamma \tau 1$ v)
 then obtain z and c where $\tau 1 = \llbracket z : B\text{-id tid} \mid c \rrbracket$ 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 $n::\text{int}$ and $\Gamma::\Gamma$
 assumes $0 \leq n \wedge n \leq \text{int } (\text{length } v)$ and $\vdash_{wf} \Theta$ and $\Theta; \{\llbracket \rrbracket\} \vdash_{wf} \Gamma$
 shows $\Theta; \{\llbracket \rrbracket\}; \Gamma \vdash [L\text{-num } n]^v \Leftarrow \llbracket z : B\text{-int} \mid ([\text{leq } [[L\text{-num } 0]^v]^{ce} [[z]^v]^{ce}]^{ce} == [[L\text{-true}]^v]^{ce}]$
 $AND ([\text{leq } [[z]^v]^{ce} [[[L\text{-bitvec } v]^v]^{ce}]^{ce}]^{ce} == [[L\text{-true}]^v]^{ce}) \rrbracket$
 proof –
 have $\Theta; \{\llbracket \rrbracket\}; \Gamma \vdash [L\text{-num } n]^v \Rightarrow \llbracket z : B\text{-int} \mid [[z]^v]^{ce} == [[L\text{-num } n]^v]^{ce} \rrbracket$
 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 $\text{check-v } \Theta \mathcal{B} \Gamma v (\llbracket z : B\text{-int} \mid c \rrbracket)$ and $\text{supp } v = \{\}$
 shows $\exists n. V\text{-lit } (L\text{-num } n) = v$
 using assms infer-int check-v-elim by (metis b-of.simps infer-v-form subtype-eq-base2)

definition sble :: $\Theta \Rightarrow \Gamma \Rightarrow \text{bool}$ where

$sble \ \Theta \ \Gamma = (\exists i. i \models \Gamma \wedge \Theta; \Gamma \vdash i)$

lemma check-v-range:

assumes $\Theta; \mathcal{B}; \Gamma \vdash v2 \Leftarrow \llbracket z : B\text{-int} \mid [\text{leq } [[L\text{-num } 0]^v]^{ce} [[z]^v]^{ce}]^{ce} == [[L\text{-true}]^v]^{ce}$
 AND
 $[\text{leq } [[z]^v]^{ce} [[[v1]^v]^{ce}]^{ce}]^{ce} == [[L\text{-true}]^v]^{ce} \rrbracket$

(is $\Theta ; ?B ; \Gamma \vdash v2 \Leftarrow \llbracket z : B\text{-int} \mid ?c1 \rrbracket$)
 and $v1 = V\text{-lit } (L\text{-bitvec } bv) \wedge v2 = V\text{-lit } (L\text{-num } n)$ and $\text{atom } z \# \Gamma$ and $\text{sble } \Theta \Gamma$
 shows $0 \leq n \wedge n \leq \text{int } (\text{length } bv)$
proof –
 have $\Theta ; ?B ; \Gamma \vdash \llbracket z : B\text{-int} \mid \llbracket [z]^v \rrbracket^{ce} == \llbracket [L\text{-num } n]^v \rrbracket^{ce} \rrbracket \lesssim \llbracket z : B\text{-int} \mid ?c1 \rrbracket$
 using *check-v-elim* *assms*
 by (*metis infer-l-uniqueness infer-natI infer-v-elim*(2))
 moreover have $\text{atom } z \# \Gamma$ using *fresh-GNil* *assms* **by** *simp*
 ultimately have $\Theta ; ?B ; ((z, B\text{-int}, \llbracket [z]^v \rrbracket^{ce} == \llbracket [L\text{-num } n]^v \rrbracket^{ce}) \#_{\Gamma} \Gamma) \models ?c1$
 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 $\Theta ; \Phi ; \mathcal{B} ; \Gamma ; \Delta \vdash AE\text{-op } Plus \ v1 \ v2 \Rightarrow \tau$
 shows $\exists z . (\llbracket z : B\text{-int} \mid C\text{-eq } (CE\text{-val } (V\text{-var } z)) (CE\text{-op } Plus \ [v1]^{ce} \ [v2]^{ce}) \rrbracket = \tau)$
 using *infer-e-elim* *assms* **by** *metis*

lemma *infer-e-leq*[*elim*]:
 assumes $\Theta ; \Phi ; \mathcal{B} ; \Gamma ; \Delta \vdash AE\text{-op } LEq \ v1 \ v2 \Rightarrow \tau$
 shows $\exists z . (\llbracket z : B\text{-bool} \mid C\text{-eq } (CE\text{-val } (V\text{-var } z)) (CE\text{-op } LEq \ [v1]^{ce} \ [v2]^{ce}) \rrbracket = \tau)$
 using *infer-e-elim* *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 $\Theta ; \Phi ; \mathcal{B} ; \Gamma ; \Delta \vdash e \Rightarrow \tau$
 shows $\Theta ; \Phi ; \mathcal{B} ; \Gamma ; \Delta \vdash_{wf} e : b\text{-of } \tau$
using *assms* **proof**(*nominal-induct* τ *avoiding*: τ *rule*: *infer-e.strong-induct*)
 case (*infer-e-valI* $\Theta \mathcal{B} \Gamma \Delta' \Phi \ v \ \tau$)
 then show *?case* using *infer-v-v-wf wf-intros* **by** *metis*
next
 case (*infer-e-plusI* $\Theta \mathcal{B} \Gamma \Delta' \Phi \ v1 \ z1 \ c1 \ v2 \ z2 \ c2 \ z3$)
 then show *?case* using *b-of.simps* *infer-v-v-wf wf-intros* **by** *metis*
next
 case (*infer-e-leqI* $\Theta \mathcal{B} \Gamma \Delta' \ v1 \ z1 \ c1 \ v2 \ z2 \ c2 \ z3$)
 then show *?case* using *b-of.simps* *infer-v-v-wf wf-intros* **by** *metis*
next
 case (*infer-e-appI* $\Theta \mathcal{B} \Gamma \Delta \ \Phi \ f \ x \ b \ c \ \tau' \ s' \ v \ \tau''$)
 have $\Theta ; \Phi ; \mathcal{B} ; \Gamma ; \Delta \vdash_{wf} AE\text{-app } f \ v : b\text{-of } \tau'$ **proof**
 show $\langle \Theta \vdash_{wf} \Phi \rangle$ using *infer-e-appI* **by** *auto*
 show $\langle \Theta ; \mathcal{B} ; \Gamma \vdash_{wf} \Delta \rangle$ using *infer-e-appI* **by** *auto*
 show $\langle \text{Some } (AF\text{-fundef } f \ (AF\text{-fun-typ-none } (AF\text{-fun-typ } x \ b \ c \ \tau' \ s')))) = \text{lookup-fun } \Phi \ f \rangle$ using
infer-e-appI **by** *auto*
 show $\Theta ; \mathcal{B} ; \Gamma \vdash_{wf} v : b$ using *infer-e-appI* *check-v-wf* *b-of.simps* **by** *metis*
qed
 moreover have $b\text{-of } \tau' = b\text{-of } (\tau'[x::=v]_v)$ 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  $\langle \text{Some } (AF\text{-fundef } f \ (AF\text{-fun-typ-some } bv \ (AF\text{-fun-typ } x b c \tau'' s')) = \text{lookup-fun } \Phi f) \rangle$  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  $\text{atom } bv \# (b\text{-of } \tau')[bv::=b]_{bb}$  using fresh-subst-if subst-b-b-def infer-e-appPI by metis
  thus  $\text{atom } bv \# (\Phi, \Theta, \mathcal{B}, \Gamma, \Delta, b', v, (b\text{-of } \tau')[bv::=b]_b)$  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- $\tau$ -def
subst-b- $\tau$ -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  $\Theta ; \mathcal{B} ; \Gamma \vdash_{wf} \tau \wedge \Theta \vdash_{wf} \Phi$ 
using assms proof(induct rule: infer-e.induct)
  case (infer-e-valI  $\Theta \mathcal{B} \Gamma \Delta' \Phi v \tau$ )
  then show ?case using infer-v-t-wf by auto

```

next
 case (*infer-e-plusI* $\Theta \mathcal{B} \Gamma \Delta \Phi \ v1 \ z1 \ c1 \ v2 \ z2 \ c2 \ z3$)
 hence $\Theta; \mathcal{B}; \Gamma \vdash_{wf} CE\text{-}op \ Plus \ [v1]^{ce} \ [v2]^{ce} : B\text{-}int$ **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* $\Theta \mathcal{B} \Gamma \Delta \Phi \ v1 \ z1 \ c1 \ v2 \ z2 \ c2 \ z3$)
 hence $\Theta; \mathcal{B}; \Gamma \vdash_{wf} CE\text{-}op \ LEq \ [v1]^{ce} \ [v2]^{ce} : B\text{-}bool$ **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* $\Theta \mathcal{B} \Gamma \Delta \Phi \ f \ x \ b \ c \ \tau \ s' \ v \ \tau'$)
 show ?case **proof**
 show $\Theta \vdash_{wf} \Phi$ **using** *infer-e-appI* **by** *auto*
 show $\Theta; \mathcal{B}; \Gamma \vdash_{wf} \tau'$ **proof** –
 have $*$: $\Theta; \mathcal{B}; \Gamma \vdash_{wf} v : b$ **using** *infer-e-appI check-v-wf(2) b-of.simps* **by** *metis*
 moreover have $*$: $\Theta; \mathcal{B}; (x, b, c) \#_{\Gamma} \Gamma \vdash_{wf} \tau$ **proof** (*rule wf-weakening1(4)*)
 show $\langle \Theta; \mathcal{B}; (x, b, c) \#_{\Gamma} GNil \vdash_{wf} \tau \rangle$ **using** *wfPhi-f-simple-wfT wfD-wf infer-e-appI wb-b-weakening*
by *fastforce*
 have $\Theta; \mathcal{B}; \Gamma \vdash_{wf} \{ x : b \mid c \}$ **using** *infer-e-appI check-v-wf(3)* **by** *auto*
 thus $\langle \Theta; \mathcal{B} \vdash_{wf} (x, b, c) \#_{\Gamma} \Gamma \rangle$ **using** *infer-e-appI*
*wfT-wfC[THEN wfG-consI[rotated 3]] * wfT-wf-cons fresh-prodN* **by** *simp*
 show $\langle toSet((x, b, c) \#_{\Gamma} GNil) \subseteq toSet((x, b, c) \#_{\Gamma} \Gamma) \rangle$ **using** *toSet.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 \Gamma v] subst-v-\tau-def*
by *auto*
qed
qed
next
 case (*infer-e-appPI* $\Theta \mathcal{B} \Gamma \Delta \Phi \ b' \ f \ bv \ x \ b \ c \ \tau' \ s' \ v \ \tau$)

 have $\Theta; \mathcal{B}; ((x, b[bv::=b]_{bb}, c[bv::=b]_{cb}) \#_{\Gamma} \Gamma)[x::=v]_{\Gamma v} \vdash_{wf} (\tau'[bv::=b]_b)[x::=v]_{\tau v}$
proof(*rule wf-subst(4)*)
 show $\langle \Theta; \mathcal{B}; (x, b[bv::=b]_{bb}, c[bv::=b]_{cb}) \#_{\Gamma} \Gamma \vdash_{wf} \tau'[bv::=b]_b \rangle$
proof(*rule wf-weakening1(4)*)
 have $\langle \Theta; \{ | bv | \}; (x, b, c) \#_{\Gamma} GNil \vdash_{wf} \tau' \rangle$ **using** *wfPhi-f-poly-wfT infer-e-appI infer-e-appPI*
by *simp*
 thus $\langle \Theta; \mathcal{B}; (x, b[bv::=b]_{bb}, c[bv::=b]_{cb}) \#_{\Gamma} GNil \vdash_{wf} \tau'[bv::=b]_b \rangle$
using *wfT-subst-wfT infer-e-appPI wb-b-weakening subst-b-\tau-def subst-v-\tau-def* **by** *presburger*
 have $\Theta; \mathcal{B}; \Gamma \vdash_{wf} \{ x : b[bv::=b]_{bb} \mid c[bv::=b]_{cb} \}$
using *infer-e-appPI check-v-wf(3) subst-b-b-def subst-b-c-def* **by** *metis*
 thus $\langle \Theta; \mathcal{B} \vdash_{wf} (x, b[bv::=b]_{bb}, c[bv::=b]_{cb}) \#_{\Gamma} \Gamma \rangle$
using *infer-e-appPI wfT-wfC[THEN wfG-consI[rotated 3]] * wfX-wfY wfT-wf-cons wb-b-weakening*
by *metis*
 show $\langle toSet((x, b[bv::=b]_{bb}, c[bv::=b]_{cb}) \#_{\Gamma} GNil) \subseteq toSet((x, b[bv::=b]_{bb}, c[bv::=b]_{cb}) \#_{\Gamma} \Gamma) \rangle$
using *toSet.simps* **by** *auto*
qed
 show $\langle (x, b[bv::=b]_{bb}, c[bv::=b]_{cb}) \#_{\Gamma} \Gamma = GNil @ (x, b[bv::=b]_{bb}, c[bv::=b]_{cb}) \#_{\Gamma} \Gamma \rangle$ **using**

```

append-g.simps by auto
  show  $\langle \Theta; \mathcal{B}; \Gamma \vdash_{wf} v : b[bv ::= b]_{bb} \rangle$  using infer-e-appPI check-v-wf(2) b-of.simps subst-b-b-def
by metis
qed
moreover have  $((x, b[bv ::= b]_{bb}, c[bv ::= b]_{cb}) \#_{\Gamma} \Gamma)[x ::= v]_{\Gamma v} = \Gamma$  using subst-gv.simps by auto
ultimately show ?case using infer-e-appPI subst-v- $\tau$ -def by simp
next
case (infer-e-fstI  $\Theta \mathcal{B} \Gamma \Delta \Phi v z' b1 b2 c z$ )
  hence  $\Theta; \mathcal{B}; \Gamma \vdash_{wf} CE\text{-fst } [v]^{ce} : b1$  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  $\Theta \mathcal{B} \Gamma \Delta \Phi v z' b1 b2 c z$ )
  hence  $\Theta; \mathcal{B}; \Gamma \vdash_{wf} CE\text{-snd } [v]^{ce} : b2$  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  $\Theta \mathcal{B} \Gamma \Delta \Phi v z' c z$ )
  hence  $\Theta; \mathcal{B}; \Gamma \vdash_{wf} CE\text{-len } [v]^{ce} : B\text{-int}$  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  $\Theta \Gamma \Phi \Delta u \tau$ )
  then show ?case using wfD-wfT by blast
next
case (infer-e-concatI  $\Theta \mathcal{B} \Gamma \Delta \Phi v1 z1 c1 v2 z2 c2 z3$ )
  hence  $\Theta; \mathcal{B}; \Gamma \vdash_{wf} CE\text{-concat } [v1]^{ce} [v2]^{ce} : B\text{-bitvec}$  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  $\Theta \mathcal{B} \Gamma \Delta \Phi v1 z1 c1 v2 z2 z3$ )

  hence wfg:  $\Theta; \mathcal{B} \vdash_{wf} (z3, [B\text{-bitvec}, B\text{-bitvec}]^b, TRUE) \#_{\Gamma} \Gamma$ 
  using infer-v-wf wfG-cons2I wfB-pairI wfB-bitvecI by simp
  have wfz:  $\Theta; \mathcal{B}; (z3, [B\text{-bitvec}, B\text{-bitvec}]^b, TRUE) \#_{\Gamma} \Gamma \vdash_{wf} [[z3]^v]^{ce} : [B\text{-bitvec}, B\text{-bitvec}]^b$ 
  apply(rule wfCE-valI, rule wfV-varI)
  using wfg apply simp
  using lookup.simps(2)[of z3 [B-bitvec, B-bitvec]b TRUE  $\Gamma$  z3] by simp
  have 1:  $\Theta; \mathcal{B}; (z3, [B\text{-bitvec}, B\text{-bitvec}]^b, TRUE) \#_{\Gamma} \Gamma \vdash_{wf} [v2]^{ce} : B\text{-int}$ 
  using check-v-wf[OF infer-e-splitI(4)] wf-weakening(1)[OF - wfg] b-of.simps toSet.simps wfCE-valI
  by fastforce
  have 2:  $\Theta; \mathcal{B}; (z3, [B\text{-bitvec}, B\text{-bitvec}]^b, TRUE) \#_{\Gamma} \Gamma \vdash_{wf} [v1]^{ce} : B\text{-bitvec}$ 
  using infer-v-wf[OF infer-e-splitI(3)] wf-weakening(1)[OF - wfg] b-of.simps toSet.simps wfCE-valI
  by fastforce

  have  $\Theta; \mathcal{B}; \Gamma \vdash_{wf} \{ z3 : [B\text{-bitvec}, B\text{-bitvec}]^b \mid [v1]^{ce} \} == [ \#1[ [z3]^v ]^{ce} ]^{ce} @@ [ \#2[ [z3]^v ]^{ce} ]^{ce}$ 
  AND  $\{ [ \#1[ [z3]^v ]^{ce} ]^{ce} \} == [v2]^{ce} \}$ 
  proof
    show atom z3  $\# (\Theta, \mathcal{B}, \Gamma)$  using infer-e-splitI wfTh-x-fresh wfX-wfY fresh-prod3 wfG-fresh-x by

```

metis

show $\Theta ; \mathcal{B} \vdash_{wf} [B\text{-bitvec} , B\text{-bitvec}]^b$ **using** *wfB-pairI wfB-bitvecI infer-e-splitI wfX-wfY* **by** *metis*
show $\Theta ; \mathcal{B} ; (z3, [B\text{-bitvec} , B\text{-bitvec}]^b, TRUE) \#_{\Gamma}$
 $\Gamma \vdash_{wf} [v1]^{ce} == [\#1[[z3]^v]^{ce}]^{ce} @@ [\#2[[z3]^v]^{ce}]^{ce} \text{ AND } [[\#1[[z3]^v]^{ce}]^{ce}]^{ce} == [v2]^{ce}$
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** $\Gamma::\Gamma$ **and** $\tau::\tau$ **and** $\Delta::\Delta$ **and** $\Phi::\Phi$
assumes $\Theta ; \Phi ; \mathcal{B} ; \Gamma ; \Delta \vdash e \Rightarrow \tau$
shows $\Theta ; \mathcal{B} ; \Gamma \vdash_{wf} \tau$ **and** $\Theta ; \mathcal{B} \vdash_{wf} \Gamma$ **and** $\Theta ; \mathcal{B} ; \Gamma \vdash_{wf} \Delta$ **and** $\Theta \vdash_{wf} \Phi$ **and** $\Theta ; \Phi ; \mathcal{B} ; \Gamma ; \Delta \vdash_{wf} e : (b\text{-of } \tau)$
using *infer-e-t-wf infer-e-e-wf wfE-wf assms* **by** *metis+*

lemma *infer-e-fresh*:

fixes $x::x$
assumes $\Theta ; \Phi ; \mathcal{B} ; \Gamma ; \Delta \vdash e \Rightarrow \tau$ **and** *atom* $x \# \Gamma$
shows *atom* $x \# (e, \tau)$

proof –

have *atom* $x \# e$ **using** *infer-e-e-wf[THEN wfE-x-fresh, OF assms(1)] assms(2)* **by** *auto*
moreover **have** *atom* $x \# \tau$ **using** *assms infer-e-wf wfT-x-fresh* **by** *metis*
ultimately show *?thesis* **using** *fresh-Pair* **by** *auto*
qed

inductive *check-e* :: $\Theta \Rightarrow \Phi \Rightarrow \mathcal{B} \Rightarrow \Gamma \Rightarrow \Delta \Rightarrow e \Rightarrow \tau \Rightarrow \text{bool}$ (- ; - ; - ; - ; - \vdash - \Leftarrow - [50, 50, 50]
50) **where**

check-e-subtypeI: $\llbracket \text{infer-e } T P B G D e \tau' ; \text{subtype } T B G \tau' \tau \rrbracket \Longrightarrow \text{check-e } T P B G D e \tau$

equivariance *check-e*

nominal-inductive *check-e* .

inductive-cases *check-e-elim*[*elim!*]:

check-e $F D B G \Theta (AE\text{-val } v) \tau$

check-e $F D B G \Theta e \tau$

lemma *infer-e-fst-pair*:

fixes $v1::v$

assumes $\Theta ; \Phi ; \{\|\} ; GNil ; \Delta \vdash [\#1[v1 , v2]^v]^e \Rightarrow \tau$

shows $\exists \tau'. \Theta ; \Phi ; \{\|\} ; GNil ; \Delta \vdash [v1]^e \Rightarrow \tau' \wedge$

$\Theta ; \{\|\} ; GNil \vdash \tau' \lesssim \tau$

proof –

obtain z' **and** $b1$ **and** $b2$ **and** c **and** z **where** $** : \tau = (\llbracket z : b1 \mid C\text{-eq } (CE\text{-val } (V\text{-var } z)) \rrbracket (CE\text{-fst } [(V\text{-pair } v1 v2)]^{ce}) \rrbracket) \wedge$

$wfD \Theta \{\|\} GNil \Delta \wedge wfPhi \Theta \Phi \wedge$

$\Theta ; \{\|\} ; GNil \vdash V\text{-pair } v1 v2 \Rightarrow \llbracket z' : B\text{-pair } b1 b2 \mid c \rrbracket \wedge \text{atom } z \# V\text{-pair } v1 v2$

using *infer-e-elim* *assms* **by** *metis*

hence $\ast: \Theta; \{\|\}; GNil \vdash V\text{-pair } v1 \ v2 \Rightarrow \{\| z' : B\text{-pair } b1 \ b2 \mid c \|\}$ **by** *auto*

obtain $t1a$ **and** $t2a$ **where**
 $\ast: \Theta; \{\|\}; GNil \vdash v1 \Rightarrow t1a \wedge \Theta; \{\|\}; GNil \vdash v2 \Rightarrow t2a \wedge B\text{-pair } b1 \ b2 = B\text{-pair } (b\text{-of } t1a)$
 $(b\text{-of } t2a)$
using *infer-v-elim5*[*OF* \ast] **by** *metis*

hence *suppv*: $\text{supp } v1 = \{\} \wedge \text{supp } v2 = \{\} \wedge \text{supp } (V\text{-pair } v1 \ v2) = \{\}$ **using** \ast *infer-v-v-wf*
wfV-supp atom-dom.simps toSet.simps supp-GNil
by (*meson wfV-supp-nil*)

thm *infer-v-form*
obtain $z1$ **and** $b1'$ **where** $t1a = \{\| z1 : b1' \mid [\| z1 \|^v]^{ce} == [\| v1 \|^ce] \|\}$
using *infer-v-form*[*of* $\Theta \{\|\} GNil \ v1 \ t1a$] \ast **by** *auto*
moreover **hence** $b1' = b1$ **using** \ast *b-of.simps* **by** *simp*
ultimately **have** $\Theta; \{\|\}; GNil \vdash v1 \Rightarrow \{\| z1 : b1 \mid CE\text{-val } (V\text{-var } z1) == CE\text{-val } v1 \|\}$ **using** \ast
by *auto*
moreover **have** $\Theta; \{\|\}; GNil \vdash_{wf} CE\text{-fst } [V\text{-pair } v1 \ v2]^{ce} : b1$ **using** *wfCE-fstI infer-v-wf(1) \ast*
b-of.simps wfCE-valI **by** *metis*
moreover **hence** $st: \Theta; \{\|\}; GNil \vdash \{\| z1 : b1 \mid CE\text{-val } (V\text{-var } z1) == CE\text{-val } v1 \|\} \lesssim (\{\| z : b1 \mid CE\text{-val } (V\text{-var } z) == CE\text{-fst } [V\text{-pair } v1 \ v2]^{ce} \|\})$
using *subtype-gnil-fst infer-v-v-wf* **by** *auto*
moreover **have** $wfD \ \Theta \ \{\|\} \ GNil \ \Delta \wedge \ wfPhi \ \Theta \ \Phi$ **using** \ast **by** *auto*
ultimately **show** *?thesis* **using** *wfX-wfY \ast infer-e-valI* **by** *metis*
qed

lemma *infer-e-snd-pair*:
assumes $\Theta; \Phi; \{\|\}; GNil; \Delta \vdash AE\text{-snd } (V\text{-pair } v1 \ v2) \Rightarrow \tau$
shows $\exists \tau'. \Theta; \Phi; \{\|\}; GNil; \Delta \vdash AE\text{-val } v2 \Rightarrow \tau' \wedge \Theta; \{\|\}; GNil \vdash \tau' \lesssim \tau$
proof –
obtain z' **and** $b1$ **and** $b2$ **and** c **and** z **where** $\ast: (\tau = (\{\| z : b2 \mid C\text{-eq } (CE\text{-val } (V\text{-var } z)) (CE\text{-snd } [(V\text{-pair } v1 \ v2)]^{ce}) \|\})) \wedge$
 $(wfD \ \Theta \ \{\|\} \ GNil \ \Delta) \wedge (wfPhi \ \Theta \ \Phi) \wedge$
 $\Theta; \{\|\}; GNil \vdash V\text{-pair } v1 \ v2 \Rightarrow \{\| z' : B\text{-pair } b1 \ b2 \mid c \|\} \wedge \text{atom } z \ \# \ V\text{-pair } v1 \ v2$
using *infer-e-elim9*[*OF* *assms(1)*] **by** *metis*
hence $\ast: \Theta; \{\|\}; GNil \vdash V\text{-pair } v1 \ v2 \Rightarrow \{\| z' : B\text{-pair } b1 \ b2 \mid c \|\}$ **by** *auto*

obtain $t1a$ **and** $t2a$ **where**
 $\ast: \Theta; \{\|\}; GNil \vdash v1 \Rightarrow t1a \wedge \Theta; \{\|\}; GNil \vdash v2 \Rightarrow t2a \wedge B\text{-pair } b1 \ b2 = B\text{-pair } (b\text{-of } t1a)$
 $(b\text{-of } t2a)$
using *infer-v-elim5*[*OF* \ast] **by** *metis*

hence *suppv*: $\text{supp } v1 = \{\} \wedge \text{supp } v2 = \{\} \wedge \text{supp } (V\text{-pair } v1 \ v2) = \{\}$ **using** *infer-v-v-wf wfV.simps*
v.supp **by** (*meson \ast wfV-supp-nil*)

obtain $z2$ **and** $b2'$ **where** $t2a = \{\| z2 : b2' \mid [\| z2 \|^v]^{ce} == [\| v2 \|^ce] \|\}$
using *infer-v-form*[*of* $\Theta \{\|\} GNil \ v2 \ t2a$] \ast **by** *auto*
moreover **hence** $b2' = b2$ **using** \ast *b-of.simps* **by** *simp*

ultimately **have** $\Theta; \{\|\}; GNil \vdash v2 \Rightarrow \{\| z2 : b2 \mid CE\text{-val } (V\text{-var } z2) == CE\text{-val } v2 \|\}$ **using** \ast
by *auto*
moreover **have** $\Theta; \{\|\}; GNil \vdash_{wf} CE\text{-snd } [V\text{-pair } v1 \ v2]^{ce} : b2$ **using** *wfCE-sndI infer-v-wf(1) \ast*

b-of.simps wfCE-valI by metis
moreover hence $st: \Theta ; \{\|\} ; GNil \vdash \llbracket z2 : b2 \mid CE\text{-val } (V\text{-var } z2) == CE\text{-val } v2 \rrbracket \lesssim (\llbracket z : b2 \mid CE\text{-val } (V\text{-var } z) == CE\text{-snd } [V\text{-pair } v1 \ v2]^{ce} \rrbracket)$
using *subtype-gnil-snd infer-v-v-wf by auto*
moreover have $wfD \ \Theta \ \{\|\} \ GNil \ \Delta \wedge \ wfPhi \ \Theta \ \Phi$ **using** *** by metis*
ultimately show *?thesis* **using** $wfX\text{-}wfY \ \text{**} \ infer\text{-e}\text{-valI}$ **by metis**
qed

12.5 Statements

lemma *check-s-v-unit*:
assumes $\Theta ; \mathcal{B} ; \Gamma \vdash (\llbracket z : B\text{-unit} \mid TRUE \rrbracket) \lesssim \tau$ **and** $wfD \ \Theta \ \mathcal{B} \ \Gamma \ \Delta$ **and** $wfPhi \ \Theta \ \Phi$
shows $\Theta ; \Phi ; \mathcal{B} ; \Gamma ; \Delta \vdash AS\text{-val } (V\text{-lit } L\text{-unit}) \Leftarrow \tau$
proof –
have $wfG \ \Theta \ \mathcal{B} \ \Gamma$ **using** *assms subtype-g-wf by meson*
moreover hence $wfTh \ \Theta$ **using** *wfG-wf by simp*
moreover obtain $z'::x$ **where** $atom \ z' \ \# \ \Gamma$ **using** *obtain-fresh by auto*
ultimately have $*:\Theta ; \mathcal{B} ; \Gamma \vdash V\text{-lit } L\text{-unit} \Rightarrow \llbracket z' : B\text{-unit} \mid CE\text{-val } (V\text{-var } z') == CE\text{-val } (V\text{-lit } L\text{-unit}) \rrbracket$
using *infer-v-litI infer-unitI by simp*
moreover have $wfT \ \Theta \ \mathcal{B} \ \Gamma \ (\llbracket z' : B\text{-unit} \mid CE\text{-val } (V\text{-var } z') == CE\text{-val } (V\text{-lit } L\text{-unit}) \rrbracket)$ **using** *infer-v-t-wf*
by (*meson calculation*)
moreover then have $\Theta ; \mathcal{B} ; \Gamma \vdash (\llbracket z' : B\text{-unit} \mid CE\text{-val } (V\text{-var } z') == CE\text{-val } (V\text{-lit } L\text{-unit}) \rrbracket) \lesssim \tau$ **using** *subtype-trans subtype-top assms*
type-for-lit.simps(4) $wfX\text{-}wfY$ **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 original term

lemma $\tau\text{-fresh-c}[simp]$:
assumes $atom \ x \ \# \ \llbracket z : b \mid c \rrbracket$ **and** $atom \ z \ \# \ x$
shows $atom \ x \ \# \ c$
using $\tau.\text{fresh} \ assms \ fresh\text{-at}\text{-base}$
by (*simp add: fresh-at-base(2)*)

lemma $wfT\text{-}wfT\text{-}if1$:
assumes $wfT \ \Theta \ \mathcal{B} \ \Gamma \ (\llbracket z : b\text{-of } t \mid CE\text{-val } v == CE\text{-val } (V\text{-lit } L\text{-false}) \ IMP \ c\text{-of } t \ z \rrbracket)$ **and** $atom \ z \ \# \ (\Gamma, t)$
shows $wfT \ \Theta \ \mathcal{B} \ \Gamma \ t$
using *assms proof(nominal-induct t avoiding: $\Gamma \ z$ rule: $\tau.\text{strong-induct}$)*
case (*T-refined-type* $z' \ b' \ c'$)
show *?case proof(rule wfT-wfT-if)*
show $\langle \Theta ; \mathcal{B} ; \Gamma \vdash_{wf} \llbracket z : b' \mid [v]^{ce} == [[L\text{-false}]^v]^{ce} \ IMP \ c'[z'::=[z]^v]_{cv} \rrbracket \rangle$
using *T-refined-type b-of.simps c-of.simps subst-defs by metis*
show $\langle atom \ z \ \# \ (c', \Gamma) \rangle$ **using** *T-refined-type fresh-prodN $\tau\text{-fresh-c}$ by metis*
qed

qed

lemma *check-s-check-branch-s-wf*:

fixes $s::s$ **and** $cs::branch-s$ **and** $\Theta::\Theta$ **and** $\Phi::\Phi$ **and** $\Gamma::\Gamma$ **and** $\Delta::\Delta$ **and** $v::v$ **and** $\tau::\tau$ **and** $css::branch-list$
shows $\Theta ; \Phi ; B ; \Gamma ; \Delta \vdash s \Leftarrow \tau \implies \Theta ; B \vdash_{wf} \Gamma \wedge wfTh \Theta \wedge wfD \Theta B \Gamma \Delta \wedge wfT \Theta B \Gamma$
 $\tau \wedge wfPhi \Theta \Phi$ **and**

$check-branch-s \Theta \Phi B \Gamma \Delta \text{ tid cons const } v \text{ cs } \tau \implies \Theta ; B \vdash_{wf} \Gamma \wedge wfTh \Theta \wedge wfD \Theta B \Gamma \Delta$
 $\wedge wfT \Theta B \Gamma \tau \wedge wfPhi \Theta \Phi$

$check-branch-list \Theta \Phi B \Gamma \Delta \text{ tid dclist } v \text{ css } \tau \implies \Theta ; B \vdash_{wf} \Gamma \wedge wfTh \Theta \wedge wfD \Theta B \Gamma \Delta$
 $\wedge wfT \Theta B \Gamma \tau \wedge wfPhi \Theta \Phi$

proof(*induct rule: check-s-check-branch-s-check-branch-list.inducts*)

case (*check-valI* $\Theta B \Gamma \Delta \Phi v \tau' \tau$)

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* $x \Theta \Phi B \Gamma \Delta e \tau z s b c$)

then have $*, wfT \Theta B ((x, b, c[z::=V-var x]_v) \#_{\Gamma} \Gamma) \tau$ **by force**

moreover have $atom x \nmid \tau$ **using** *check-letI fresh-prodN* **by force**

ultimately have $\Theta ; B ; \Gamma \vdash_{wf} \tau$ **using** *wfT-restrict2* **by force**

then show ?case **using** *check-letI infer-e-wf wfS-letI wfX-wfY wfG-elim* **by metis**

next

case (*check-assertI* $x \Theta \Phi B \Gamma \Delta c \tau s$)

then have $*, wfT \Theta B ((x, B-bool, c) \#_{\Gamma} \Gamma) \tau$ **by force**

moreover have $atom x \nmid \tau$ **using** *check-assertI fresh-prodN* **by force**

ultimately have $\Theta ; B ; \Gamma \vdash_{wf} \tau$ **using** *wfT-restrict2* **by force**

then show ?case **using** *check-assertI wfS-assertI wfX-wfY wfG-elim* **by metis**

next

case (*check-branch-s-branchI* $\Theta B \Gamma \Delta \tau \text{ cons const } x v \Phi s \text{ tid}$)

then show ?case **using** *wfX-wfY* **by metis**

next

case (*check-branch-list-consI* $\Theta \Phi B \Gamma \Delta \text{ tid dclist' } v \text{ cs } \tau \text{ css }$)

then show ?case **using** *wfX-wfY* **by metis**

next

case (*check-branch-list-finalI* $\Theta \Phi B \Gamma \Delta \text{ tid dclist' } v \text{ cs } \tau$)

then show ?case **using** *wfX-wfY* **by metis**

next

case (*check-ifI* $z \Theta \Phi B \Gamma \Delta v s1 s2 \tau$)

hence $*, wfT \Theta B \Gamma (\llbracket z : b-of \tau \mid CE-val v == CE-val (V-lit L-false) IMP c-of \tau z \rrbracket)$ **(is wfT**
 $\Theta B \Gamma ?tau)$ **by auto**

hence $wfT \Theta B \Gamma \tau$ **using** *wfT-wfT-if1 check-ifI fresh-prodN* **by metis**

hence $\Theta ; B ; \Gamma \vdash_{wf} \tau$ **using** *check-ifI b-of-c-of-eq fresh-prodN* **by auto**

thus ?case **using** *check-ifI* **by metis**

next

case (*check-let2I* $x \Theta \Phi B G \Delta t s1 \tau s2$)

then have $wfT \Theta B ((x, b-of t, (c-of t x)) \#_{\Gamma} G) \tau$ **by fastforce**

moreover have $atom x \nmid \tau$ **using** *check-let2I* **by force**

ultimately have $wfT \Theta B G \tau$ **using** *wfT-restrict2* **by metis**

then show ?case **using** *check-let2I* **by argo**

next

case (*check-varI* $u \Delta P G v \tau' \Phi s \tau$)

then show ?case **using** *wfG-elim wfD-elim*

list.distinct list.inject **by metis**


```

next
  case (check-assignI  $\Theta \Phi \mathcal{B} \Gamma \Delta u \tau v z \tau'$ )
  obtain  $z'::x$  where  $*:atom\ z' \# \Gamma$  using obtain-fresh by metis
  moreover have  $\llbracket z : B\text{-unit} \mid TRUE \rrbracket = \llbracket z' : B\text{-unit} \mid TRUE \rrbracket$  by auto
  moreover hence  $wfT \Theta \mathcal{B} \Gamma \llbracket z' : B\text{-unit} \mid TRUE \rrbracket$  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  $\Phi \Delta G P s1 z s2 \tau'$ )
  then show ?case using subtype-wf subtype-wf by auto
next
  case (check-seqI  $\Delta G P s1 z s2 \tau$ )
  then show ?case by fast
next
  case (check-caseI  $\Theta \Phi \mathcal{B} \Gamma \Delta dclist\ cs\ \tau\ tid\ v\ z$ )
  then show ?case by fast
qed

lemma fresh-u-replace-true:
  fixes  $bv::bv$  and  $\Gamma::\Gamma$ 
  assumes  $atom\ bv \# \Gamma' @ (x, b, c) \#_{\Gamma} \Gamma$ 
  shows  $atom\ bv \# \Gamma' @ (x, b, TRUE) \#_{\Gamma} \Gamma$ 
  using fresh-append-g fresh-GCons assms fresh-Pair c.fresh(1) by auto

lemma wf-replace-true1:
  fixes  $\Gamma::\Gamma$  and  $\Phi::\Phi$  and  $\Theta::\Theta$  and  $\Gamma':\Gamma$  and  $v::v$  and  $e::e$  and  $c::c$  and  $c'::c$  and  $c'::c$  and  $\tau::\tau$ 
  and  $ts::(string*\tau)$  list and  $\Delta::\Delta$  and  $b'::b$  and  $b::b$  and  $s::s$ 
  and  $ftq::fun\text{-}typ\text{-}q$  and  $ft::fun\text{-}typ$  and  $ce::ce$  and  $td::type\text{-}def$  and  $cs::branch\text{-}s$  and
   $css::branch\text{-}list$ 

  shows  $\Theta; \mathcal{B}; G \vdash_{wf} v : b' \implies G = \Gamma' @ (x, b, c) \#_{\Gamma} \Gamma \implies \Theta; \mathcal{B}; \Gamma' @ ((x, b, TRUE) \#_{\Gamma} \Gamma) \vdash_{wf} v : b'$  and
   $\Theta; \mathcal{B}; G \vdash_{wf} c'' \implies G = \Gamma' @ (x, b, c) \#_{\Gamma} \Gamma \implies \Theta; \mathcal{B}; \Gamma' @ ((x, b, TRUE) \#_{\Gamma} \Gamma) \vdash_{wf} c''$  and
   $\Theta; \mathcal{B} \vdash_{wf} G \implies G = \Gamma' @ (x, b, c) \#_{\Gamma} \Gamma \implies \Theta; \mathcal{B} \vdash_{wf} \Gamma' @ ((x, b, TRUE) \#_{\Gamma} \Gamma)$  and
   $\Theta; \mathcal{B}; G \vdash_{wf} \tau \implies G = \Gamma' @ (x, b, c) \#_{\Gamma} \Gamma \implies \Theta; \mathcal{B}; \Gamma' @ ((x, b, TRUE) \#_{\Gamma} \Gamma) \vdash_{wf} \tau$ 
  and
   $\Theta; \mathcal{B}; \Gamma \vdash_{wf} ts \implies True$  and
   $\vdash_{wf} P \implies True$  and
   $\Theta; \mathcal{B} \vdash_{wf} b \implies True$  and
   $\Theta; \mathcal{B}; G \vdash_{wf} ce : b' \implies G = \Gamma' @ (x, b, c) \#_{\Gamma} \Gamma \implies \Theta; \mathcal{B}; \Gamma' @ ((x, b, TRUE) \#_{\Gamma} \Gamma) \vdash_{wf} ce : b'$  and
   $\Theta \vdash_{wf} td \implies True$ 
proof(nominal-induct
   $b'$  and  $c''$  and  $G$  and  $\tau$  and  $ts$  and  $P$  and  $b$  and  $b'$  and  $td$ 
  arbitrary:  $\Gamma \Gamma'$  and  $\Gamma \Gamma'$  and  $\Gamma \Gamma'$  and  $\Gamma \Gamma'$  and  $\Gamma \Gamma'$  and  $\Gamma \Gamma'$  and  $\Gamma \Gamma'$  and  $\Gamma \Gamma'$  and  $\Gamma \Gamma'$  and  $\Gamma \Gamma'$  and  $\Gamma \Gamma'$  and  $\Gamma \Gamma'$  and  $\Gamma \Gamma'$  and  $\Gamma \Gamma'$  and  $\Gamma \Gamma'$ 
  rule:wfV-wfC-wfG-wfT-wfTs-wfTh-wfB-wfCE-wfTD.strong-induct)
case (wfB-intI  $\Theta \mathcal{B}$ )
  then show ?case using wf-intros by metis

```

```

next
  case (wfB-boolI  $\Theta \mathcal{B}$ )
  then show ?case using wf-intros by metis
next
  case (wfB-unitI  $\Theta \mathcal{B}$ )
  then show ?case using wf-intros by metis
next
  case (wfB-bitvecI  $\Theta \mathcal{B}$ )
  then show ?case using wf-intros by metis
next
  case (wfB-pairI  $\Theta \mathcal{B} b1 b2$ )
  then show ?case using wf-intros by metis
next
  case (wfB-consI  $\Theta s dclist \mathcal{B}$ )
  then show ?case using wf-intros by metis
next
  case (wfB-appI  $\Theta b s bv dclist \mathcal{B}$ )
  then show ?case using wf-intros by metis
next
  case (wfV-varI  $\Theta \mathcal{B} \Gamma'' b' c x'$ )
  hence wfg:  $\langle \Theta ; \mathcal{B} \vdash_{wf} \Gamma' @ (x, b, TRUE) \#_{\Gamma} \Gamma \rangle$  by auto
  show ?case proof(cases  $x=x'$ )
    case True
    hence Some  $(b, TRUE) = \text{lookup } (\Gamma' @ (x, b, TRUE) \#_{\Gamma} \Gamma) x'$  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.prem)
next
  case False
  hence Some  $(b', c) = \text{lookup } (\Gamma' @ (x, b, TRUE) \#_{\Gamma} \Gamma) x'$  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.prem Wellformed.wfV-varI wf-replace-inside(1))
qed
next
  case (wfV-litI  $\Theta \mathcal{B} \Gamma l$ )
  then show ?case using wf-intros using wf-intros by metis
next
  case (wfV-pairI  $\Theta \mathcal{B} \Gamma v1 b1 v2 b2$ )
  then show ?case using wf-intros by metis
next
  case (wfV-consI  $s dclist \Theta dc x b' c \mathcal{B} \Gamma v$ )
  then show ?case using wf-intros by metis
next
  case (wfV-conspI  $s bv dclist \Theta dc xc bc cc \mathcal{B} b' \Gamma'' v$ )
  show ?case proof
  show  $\langle AF\text{-typedef-poly } s bv dclist \in \text{set } \Theta \rangle$  using wfV-conspI by metis
  show  $\langle (dc, \{ xc : bc \mid cc \}) \in \text{set } dclist \rangle$  using wfV-conspI by metis
  show  $\langle \Theta ; \mathcal{B} \vdash_{wf} b' \rangle$  using wfV-conspI by metis
  show  $\langle \Theta ; \mathcal{B} ; \Gamma' @ (x, b, TRUE) \#_{\Gamma} \Gamma \vdash_{wf} v : bc[bv ::= b]_{bb} \rangle$  using wfV-conspI by metis

```

```

    have atom bv  $\# \Gamma' @ (x, b, TRUE) \#_{\Gamma} \Gamma$  using fresh-u-replace-true wfV-conspI by metis
    thus  $\langle atom bv \# (\Theta, \mathcal{B}, \Gamma' @ (x, b, TRUE) \#_{\Gamma} \Gamma, b', v) \rangle$  using wfV-conspI fresh-prodN by metis
  qed
next
case (wfCE-valI  $\Theta \mathcal{B} \Gamma v b$ )
then show ?case using wf-intros by metis
next
case (wfCE-plusI  $\Theta \mathcal{B} \Gamma v1 v2$ )
then show ?case using wf-intros by metis
next
case (wfCE-leqI  $\Theta \mathcal{B} \Gamma v1 v2$ )
then show ?case using wf-intros by metis
next
case (wfCE-fstI  $\Theta \mathcal{B} \Gamma v1 b1 b2$ )
then show ?case using wf-intros by metis
next
case (wfCE-sndI  $\Theta \mathcal{B} \Gamma v1 b1 b2$ )
then show ?case using wf-intros by metis
next
case (wfCE-concatI  $\Theta \mathcal{B} \Gamma v1 v2$ )
then show ?case using wf-intros by metis
next
case (wfCE-lenI  $\Theta \mathcal{B} \Gamma v1$ )
then show ?case using wf-intros by metis
next
case (wfTI z  $\Theta \mathcal{B} \Gamma'' b' c'$ )
show ?case proof
  show  $\langle atom z \# (\Theta, \mathcal{B}, \Gamma' @ (x, b, TRUE) \#_{\Gamma} \Gamma) \rangle$  using wfTI fresh-append-g fresh-GCons fresh-prodN
by auto
  show  $\langle \Theta ; \mathcal{B} \vdash_{wf} b' \rangle$  using wfTI by metis
  show  $\langle \Theta ; \mathcal{B}; (z, b', TRUE) \#_{\Gamma} \Gamma' @ (x, b, TRUE) \#_{\Gamma} \Gamma \vdash_{wf} c' \rangle$  using wfTI append-g.simps
by metis
qed
next
case (wfC-eqI  $\Theta \mathcal{B} \Gamma e1 b e2$ )
then show ?case using wf-intros by metis
next
case (wfC-trueI  $\Theta \mathcal{B} \Gamma$ )
then show ?case using wf-intros by metis
next
case (wfC-falseI  $\Theta \mathcal{B} \Gamma$ )
then show ?case using wf-intros by metis
next
case (wfC-conjI  $\Theta \mathcal{B} \Gamma c1 c2$ )
then show ?case using wf-intros by metis
next
case (wfC-disjI  $\Theta \mathcal{B} \Gamma c1 c2$ )
then show ?case using wf-intros by metis
next
case (wfC-notI  $\Theta \mathcal{B} \Gamma c1$ )
then show ?case using wf-intros by metis
next

```

```

  case (wfC-impI  $\Theta \mathcal{B} \Gamma c1 c2$ )
  then show ?case using wf-intros by metis
next
  case (wfG-nilI  $\Theta \mathcal{B}$ )
  then show ?case using GNil-append by blast
next
  case (wfG-cons1I  $c \Theta \mathcal{B} \Gamma'' x b$ )
  then show ?case using wf-intros wfG-cons-TRUE2 wfG-elim(2) wfG-replace-inside wfG-suffix
    by (metis (no-types, lifting))
next
  case (wfG-cons2I  $c \Theta \mathcal{B} \Gamma'' x' b$ )
  then show ?case using wf-intros
    by (metis wfG-cons-TRUE2 wfG-elim(2) wfG-replace-inside wfG-suffix)
next
  case wfTh-emptyI
  then show ?case using wf-intros by metis
next
  case (wfTh-consI tdef  $\Theta$ )
  then show ?case using wf-intros by metis
next
  case (wfTD-simpleI  $\Theta lst s$ )
  then show ?case using wf-intros by metis
next
  case (wfTD-poly  $\Theta bv lst s$ )
  then show ?case using wf-intros by metis
next
  case (wfTs-nil  $\Theta \mathcal{B} \Gamma$ )
  then show ?case using wf-intros by metis
next
  case (wfTs-cons  $\Theta \mathcal{B} \Gamma \tau dc ts$ )
  then show ?case using wf-intros by metis
qed

```

lemma wf-replace-true2:

fixes $\Gamma::\Gamma$ and $\Phi::\Phi$ and $\Theta::\Theta$ and $\Gamma'::\Gamma$ and $v::v$ and $e::e$ and $c::c$ and $c'::c$ and $c'::c$ and $\tau::\tau$
 and $ts::(string*\tau)$ list and $\Delta::\Delta$ and $b'::b$ and $b::b$ and $s::s$
 and $ftq::fun\text{-}typ\text{-}q$ and $ft::fun\text{-}typ$ and $ce::ce$ and $td::type\text{-}def$ and $cs::branch\text{-}s$ and
 $css::branch\text{-}list$

shows $\Theta ; \Phi ; \mathcal{B} ; G ; D \vdash_{wf} e : b' \implies G = \Gamma' @ (x, b, c) \#_{\Gamma} \Gamma \implies \Theta ; \Phi ; \mathcal{B} ; \Gamma' @ ((x, b, TRUE) \#_{\Gamma} \Gamma) ; D \vdash_{wf} e : b'$ **and**
 $\Theta ; \Phi ; \mathcal{B} ; G ; \Delta \vdash_{wf} s : b' \implies G = \Gamma' @ (x, b, c) \#_{\Gamma} \Gamma \implies \Theta ; \Phi ; \mathcal{B} ; \Gamma' @ ((x, b, TRUE) \#_{\Gamma} \Gamma) ; \Delta \vdash_{wf} s : b'$ **and**
 $\Theta ; \Phi ; \mathcal{B} ; G ; \Delta ; tid ; dc ; t \vdash_{wf} cs : b' \implies G = \Gamma' @ (x, b, c) \#_{\Gamma} \Gamma \implies \Theta ; \Phi ; \mathcal{B} ; \Gamma' @ ((x, b, TRUE) \#_{\Gamma} \Gamma) ; \Delta ; tid ; dc ; t \vdash_{wf} cs : b'$ **and**
 $\Theta ; \Phi ; \mathcal{B} ; G ; \Delta ; tid ; dclist \vdash_{wf} css : b' \implies G = \Gamma' @ (x, b, c) \#_{\Gamma} \Gamma \implies \Theta ; \Phi ; \mathcal{B} ; \Gamma' @ ((x, b, TRUE) \#_{\Gamma} \Gamma) ; \Delta ; tid ; dclist \vdash_{wf} css : b'$ **and**
 $\Theta \vdash_{wf} \Phi \implies True$ **and**
 $\Theta ; \mathcal{B} ; G \vdash_{wf} \Delta \implies G = \Gamma' @ (x, b, c) \#_{\Gamma} \Gamma \implies \Theta ; \mathcal{B} ; \Gamma' @ ((x, b, TRUE) \#_{\Gamma} \Gamma) \vdash_{wf} \Delta$ **and**

```

     $\Theta ; \Phi \vdash_{wf} ftq \implies \text{True}$  and
     $\Theta ; \Phi ; \mathcal{B} \vdash_{wf} ft \implies \text{True}$ 
proof(nominal-induct
      b' and b' and b' and b' and  $\Phi$  and  $\Delta$  and ftq and ft
      arbitrary:  $\Gamma \Gamma'$  and  $\Gamma \Gamma'$  and  $\Gamma \Gamma'$  and  $\Gamma \Gamma'$  and  $\Gamma \Gamma'$  and  $\Gamma \Gamma'$  and  $\Gamma \Gamma'$  and  $\Gamma \Gamma'$  and  $\Gamma \Gamma'$  and  $\Gamma \Gamma'$ 
      and  $\Gamma \Gamma'$  and  $\Gamma \Gamma'$  and  $\Gamma \Gamma'$  and  $\Gamma \Gamma'$  and  $\Gamma \Gamma'$  and  $\Gamma \Gamma'$  and  $\Gamma \Gamma'$  and  $\Gamma \Gamma'$ 
      rule:wfE-wfS-wfCS-wfCSS-wfPhi-wfD-wfFTQ-wfFT.strong-induct)

  case (wfE-valI  $\Theta \Phi \mathcal{B} \Gamma \Delta v b$ )
  then show ?case using wf-intros using wf-intros wf-replace-true1 by metis
next
  case (wfE-plusI  $\Theta \Phi \mathcal{B} \Gamma \Delta v1 v2$ )
  then show ?case using wf-intros wf-replace-true1 by metis
next
  case (wfE-legI  $\Theta \Phi \mathcal{B} \Gamma \Delta v1 v2$ )
  then show ?case using wf-intros wf-replace-true1 by metis
next
  case (wfE-fstI  $\Theta \Phi \mathcal{B} \Gamma \Delta v1 b1 b2$ )
  then show ?case using wf-intros wf-replace-true1 by metis
next
  case (wfE-sndI  $\Theta \Phi \mathcal{B} \Gamma \Delta v1 b1 b2$ )
  then show ?case using wf-intros wf-replace-true1 by metis
next
  case (wfE-concatI  $\Theta \Phi \mathcal{B} \Gamma \Delta v1 v2$ )
  then show ?case using wf-intros wf-replace-true1 by metis
next
  case (wfE-splitI  $\Theta \Phi \mathcal{B} \Gamma \Delta v1 v2$ )
  then show ?case using wf-intros wf-replace-true1 by metis
next
  case (wfE-lenI  $\Theta \Phi \mathcal{B} \Gamma \Delta v1$ )
  then show ?case using wf-intros wf-replace-true1 by metis
next
  case (wfE-appI  $\Theta \Phi \mathcal{B} \Gamma \Delta f x b c \tau s v$ )
  then show ?case using wf-intros wf-replace-true1 by metis
next
  case (wfE-appPI  $\Theta \Phi \mathcal{B} \Gamma'' \Delta b' bv v \tau f x1 b1 c1 s$ )
  show ?case proof
    show  $\langle \Theta \vdash_{wf} \Phi \rangle$  using wfE-appPI wf-replace-true1 by metis
    show  $\langle \Theta ; \mathcal{B} ; \Gamma' @ (x, b, \text{TRUE}) \#_{\Gamma} \Gamma \vdash_{wf} \Delta \rangle$  using wfE-appPI by metis
    show  $\langle \Theta ; \mathcal{B} \vdash_{wf} b' \rangle$  using wfE-appPI by metis
    have atom bv  $\# \Gamma' @ (x, b, \text{TRUE}) \#_{\Gamma} \Gamma$  using fresh-u-replace-true wfE-appPI fresh-prodN by
metis
    thus  $\langle \text{atom bv } \# (\Phi, \Theta, \mathcal{B}, \Gamma' @ (x, b, \text{TRUE}) \#_{\Gamma} \Gamma, \Delta, b', v, (b\text{-of } \tau)[bv::=b]_b) \rangle$ 
      using wfE-appPI fresh-prodN by auto
    show  $\langle \text{Some } (AF\text{-fundef } f (AF\text{-fun-typ-some } bv (AF\text{-fun-typ } x1 b1 c1 \tau s))) = \text{lookup-fun } \Phi f \rangle$ 
using wfE-appPI by metis
    show  $\langle \Theta ; \mathcal{B} ; \Gamma' @ (x, b, \text{TRUE}) \#_{\Gamma} \Gamma \vdash_{wf} v : b1[bv::=b]_b \rangle$  using wfE-appPI wf-replace-true1
by metis
  qed
next
  case (wfE-mvarI  $\Theta \Phi \mathcal{B} \Gamma \Delta u \tau$ )
  then show ?case using wf-intros wf-replace-true1 by metis

```

next

case (wfS-valI $\Theta \Phi \mathcal{B} \Gamma v b \Delta$)
 then show ?case using wf-intros wf-replace-true1 by metis

next

case (wfS-letI $\Theta \Phi \mathcal{B} \Gamma'' \Delta e b' x1 s b1$)

show ?case proof

show $\langle \Theta ; \Phi ; \mathcal{B} ; \Gamma' @ (x, b, TRUE) \#_{\Gamma} \Gamma ; \Delta \vdash_{wf} e : b' \rangle$ using wfS-letI wf-replace-true1 by metis

have $\langle \Theta ; \Phi ; \mathcal{B} ; ((x1, b', TRUE) \#_{\Gamma} \Gamma') @ (x, b, TRUE) \#_{\Gamma} \Gamma ; \Delta \vdash_{wf} s : b1 \rangle$ apply(rule wfS-letI(4))

using wfS-letI append-g.simps by simp

thus $\langle \Theta ; \Phi ; \mathcal{B} ; (x1, b', TRUE) \#_{\Gamma} \Gamma' @ (x, b, TRUE) \#_{\Gamma} \Gamma ; \Delta \vdash_{wf} s : b1 \rangle$ using append-g.simps by auto

show $\langle \Theta ; \mathcal{B} ; \Gamma' @ (x, b, TRUE) \#_{\Gamma} \Gamma \vdash_{wf} \Delta \rangle$ using wfS-letI by metis

show atom x1 $\# (\Phi, \Theta, \mathcal{B}, \Gamma' @ (x, b, TRUE) \#_{\Gamma} \Gamma, \Delta, e, b1)$ using fresh-append-g fresh-GCons fresh-prodN wfS-letI by auto

qed

next

case (wfS-assertI $\Theta \Phi \mathcal{B} x' c \Gamma'' \Delta s b'$)

show ?case proof

show $\langle \Theta ; \Phi ; \mathcal{B} ; (x', B\text{-bool}, c) \#_{\Gamma} \Gamma' @ (x, b, TRUE) \#_{\Gamma} \Gamma ; \Delta \vdash_{wf} s : b' \rangle$

using wfS-assertI (2)[of $(x', B\text{-bool}, c) \#_{\Gamma} \Gamma' \Gamma$] wfS-assertI by simp

show $\langle \Theta ; \mathcal{B} ; \Gamma' @ (x, b, TRUE) \#_{\Gamma} \Gamma \vdash_{wf} c \rangle$ using wfS-assertI wf-replace-true1 by metis

show $\langle \Theta ; \mathcal{B} ; \Gamma' @ (x, b, TRUE) \#_{\Gamma} \Gamma \vdash_{wf} \Delta \rangle$ using wfS-assertI by metis

show $\langle \text{atom } x' \# (\Phi, \Theta, \mathcal{B}, \Gamma' @ (x, b, TRUE) \#_{\Gamma} \Gamma, \Delta, c, b', s) \rangle$ using wfS-assertI fresh-prodN

by simp

qed

next

case (wfS-let2I $\Theta \Phi \mathcal{B} \Gamma'' \Delta s1 \tau x' s2 ba'$)

show ?case proof

show $\langle \Theta ; \Phi ; \mathcal{B} ; \Gamma' @ (x, b, TRUE) \#_{\Gamma} \Gamma ; \Delta \vdash_{wf} s1 : b\text{-of } \tau \rangle$ using wfS-let2I wf-replace-true1

by metis

show $\langle \Theta ; \mathcal{B} ; \Gamma' @ (x, b, TRUE) \#_{\Gamma} \Gamma \vdash_{wf} \tau \rangle$ using wfS-let2I wf-replace-true1 by metis

have $\langle \Theta ; \Phi ; \mathcal{B} ; ((x', b\text{-of } \tau, TRUE) \#_{\Gamma} \Gamma') @ (x, b, TRUE) \#_{\Gamma} \Gamma ; \Delta \vdash_{wf} s2 : ba' \rangle$

apply(rule wfS-let2I(5))

using wfS-let2I append-g.simps by auto

thus $\langle \Theta ; \Phi ; \mathcal{B} ; (x', b\text{-of } \tau, TRUE) \#_{\Gamma} \Gamma' @ (x, b, TRUE) \#_{\Gamma} \Gamma ; \Delta \vdash_{wf} s2 : ba' \rangle$ using

wfS-let2I append-g.simps by auto

show $\langle \text{atom } x' \# (\Phi, \Theta, \mathcal{B}, \Gamma' @ (x, b, TRUE) \#_{\Gamma} \Gamma, \Delta, s1, ba', \tau) \rangle$ using fresh-append-g fresh-GCons fresh-prodN wfS-let2I by auto

qed

next

case (wfS-ifI $\Theta \mathcal{B} \Gamma v \Phi \Delta s1 b s2$)

then show ?case using wf-intros wf-replace-true1 by metis

next

case (wfS-varI $\Theta \mathcal{B} \Gamma'' \tau v u \Phi \Delta b' s$)

show ?case proof

show $\langle \Theta ; \mathcal{B} ; \Gamma' @ (x, b, TRUE) \#_{\Gamma} \Gamma \vdash_{wf} \tau \rangle$ using wfS-varI wf-replace-true1 by metis

show $\langle \Theta ; \mathcal{B} ; \Gamma' @ (x, b, TRUE) \#_{\Gamma} \Gamma \vdash_{wf} v : b\text{-of } \tau \rangle$ using wfS-varI wf-replace-true1 by metis

show $\langle \text{atom } u \# (\Phi, \Theta, \mathcal{B}, \Gamma' @ (x, b, TRUE) \#_{\Gamma} \Gamma, \Delta, \tau, v, b') \rangle$ using wfS-varI u-fresh-g fresh-prodN

by auto

```

  show  $\langle \Theta ; \Phi ; \mathcal{B} ; \Gamma' @ (x, b, TRUE) \#_{\Gamma} \Gamma ; (u, \tau) \#_{\Delta} \Delta \vdash_{wf} s : b' \rangle$  using wfS-varI by metis
qed

next
  case (wfS-assignI  $u \tau \Delta \Theta \mathcal{B} \Gamma \Phi v$ )
  then show ?case using wf-intros wf-replace-true1 by metis
next
  case (wfS-whileI  $\Theta \Phi \mathcal{B} \Gamma \Delta s1 s2 b$ )
  then show ?case using wf-intros wf-replace-true1 by metis
next
  case (wfS-seqI  $\Theta \Phi \mathcal{B} \Gamma \Delta s1 s2 b$ )
  then show ?case using wf-intros by metis
next
  case (wfS-matchI  $\Theta \mathcal{B} \Gamma'' v tid dclist \Delta \Phi cs b'$ )
  show ?case proof
  show  $\langle \Theta ; \mathcal{B} ; \Gamma' @ (x, b, TRUE) \#_{\Gamma} \Gamma \vdash_{wf} v : B-id tid \rangle$  using wfS-matchI wf-replace-true1 by auto
  show  $\langle AF-typedef tid dclist \in set \Theta \rangle$  using wfS-matchI by auto
  show  $\langle \Theta ; \mathcal{B} ; \Gamma' @ (x, b, TRUE) \#_{\Gamma} \Gamma \vdash_{wf} \Delta \rangle$  using wfS-matchI by auto
  show  $\langle \Theta \vdash_{wf} \Phi \rangle$  using wfS-matchI by auto
  show  $\langle \Theta ; \Phi ; \mathcal{B} ; \Gamma' @ (x, b, TRUE) \#_{\Gamma} \Gamma ; \Delta ; tid ; dclist \vdash_{wf} cs : b' \rangle$  using wfS-matchI by auto
qed
next
  case (wfS-branchI  $\Theta \Phi \mathcal{B} x' \tau \Gamma'' \Delta s b' tid dc$ )
  show ?case proof
  have  $\langle \Theta ; \Phi ; \mathcal{B} ; ((x', b-of \tau, TRUE) \#_{\Gamma} \Gamma') @ (x, b, TRUE) \#_{\Gamma} \Gamma ; \Delta \vdash_{wf} s : b' \rangle$  using
wfS-branchI append-g.simps by metis
  thus  $\langle \Theta ; \Phi ; \mathcal{B} ; (x', b-of \tau, TRUE) \#_{\Gamma} \Gamma' @ (x, b, TRUE) \#_{\Gamma} \Gamma ; \Delta \vdash_{wf} s : b' \rangle$  using wfS-branchI
append-g.simps append-g.simps by metis
  show  $\langle atom x' \# (\Phi, \Theta, \mathcal{B}, \Gamma' @ (x, b, TRUE) \#_{\Gamma} \Gamma, \Delta, \Gamma' @ (x, b, TRUE) \#_{\Gamma} \Gamma, \tau) \rangle$  using
wfS-branchI by auto
  show  $\langle \Theta ; \mathcal{B} ; \Gamma' @ (x, b, TRUE) \#_{\Gamma} \Gamma \vdash_{wf} \Delta \rangle$  using wfS-branchI by auto
qed
next
  case (wfS-finalI  $\Theta \Phi \mathcal{B} \Gamma \Delta tid dc t cs b$ )
  then show ?case using wf-intros by metis
next
  case (wfS-cons  $\Theta \Phi \mathcal{B} \Gamma \Delta tid dc t cs b dclist css$ )
  then show ?case using wf-intros by metis
next
  case (wfD-emptyI  $\Theta \mathcal{B} \Gamma$ )
  then show ?case using wf-intros wf-replace-true1 by metis
next
  case (wfD-cons  $\Theta \mathcal{B} \Gamma \Delta \tau u$ )
  then show ?case using wf-intros wf-replace-true1 by metis
next
  case (wfPhi-emptyI  $\Theta$ )
  then show ?case using wf-intros by metis
next
  case (wfPhi-consI  $f \Theta \Phi ft$ )
  then show ?case using wf-intros by metis
next
  case (wfFTNone  $\Theta \Phi ft$ )

```

```

  then show ?case using wf-intros by metis
next
  case (wfFTSome  $\Theta \Phi bv ft$ )
  then show ?case using wf-intros by metis
next
  case (wfFTI  $\Theta B b \Phi x c s \tau$ )
  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  $css::branch-s$  and  $\Theta::\Theta$  and  $\Phi::\Phi$  and  $\Gamma::\Gamma$  and  $\Delta::\Delta$  and  $v::v$  and  $\tau::\tau$  and  $css::branch-list$ 
  shows  $\Theta ; \Phi ; B ; \Gamma ; \Delta \vdash s \Leftarrow \tau \implies \Theta ; \Phi ; B ; \Gamma ; \Delta \vdash_{wf} s : b\text{-of } \tau$  and
     $check\text{-branch}\text{-}s \Theta \Phi B \Gamma \Delta \text{ tid cons const } v \text{ cs } \tau \implies wfCS \Theta \Phi B \Gamma \Delta \text{ tid cons const cs } (b\text{-of } \tau)$ 
     $check\text{-branch}\text{-}list \Theta \Phi B \Gamma \Delta \text{ tid dclist } v \text{ css } \tau \implies wfCSS \Theta \Phi B \Gamma \Delta \text{ tid dclist css } (b\text{-of } \tau)$ 
proof(induct rule: check-s-check-branch-s-check-branch-list.inducts)
case (check-valI  $\Theta \mathcal{B} \Gamma \Delta \Phi v \tau' \tau$ )
  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  $x \Theta \Phi \mathcal{B} \Gamma \Delta e \tau z s b c$ )
  show ?case proof
    show  $\langle \Theta ; \Phi ; \mathcal{B} ; \Gamma ; \Delta \vdash_{wf} e : b \rangle$  using infer-e-wf check-letI b-of.simps by metis
    show  $\langle \Theta ; \Phi ; \mathcal{B} ; (x, b, TRUE) \#_{\Gamma} \Gamma ; \Delta \vdash_{wf} s : b\text{-of } \tau \rangle$ 
      using check-letI b-of.simps wf-replace-true2(2)[OF check-letI(5)] append-g.simps by metis
    show  $\langle \Theta ; \mathcal{B} ; \Gamma \vdash_{wf} \Delta \rangle$  using infer-e-wf check-letI b-of.simps by metis
    show  $\langle atom\ x \# (\Phi, \Theta, \mathcal{B}, \Gamma, \Delta, e, b\text{-of } \tau) \rangle$ 
      apply(simp add: fresh-prodN, intro conjI)
      using check-letI(1) fresh-prod7 by simp+
  qed
next
  case (check-assertI  $x \Theta \Phi \mathcal{B} \Gamma \Delta c \tau s$ )
  show ?case proof

    show  $\langle \Theta ; \Phi ; \mathcal{B} ; (x, B\text{-bool}, c) \#_{\Gamma} \Gamma ; \Delta \vdash_{wf} s : b\text{-of } \tau \rangle$  using check-assertI by auto
  next
    show  $\langle \Theta ; \mathcal{B} ; \Gamma \vdash_{wf} c \rangle$  using check-assertI by auto
  next
    show  $\langle \Theta ; \mathcal{B} ; \Gamma \vdash_{wf} \Delta \rangle$  using check-assertI by auto
  next
    show  $\langle atom\ x \# (\Phi, \Theta, \mathcal{B}, \Gamma, \Delta, c, b\text{-of } \tau, s) \rangle$  using check-assertI by auto
  qed
next
  case (check-branch-s-branchI  $\Theta \mathcal{B} \Gamma \Delta \tau \text{ const } x \Phi \text{ tid cons } v s$ )
  show ?case proof
    show  $\langle \Theta ; \Phi ; \mathcal{B} ; (x, b\text{-of } const, TRUE) \#_{\Gamma} \Gamma ; \Delta \vdash_{wf} s : b\text{-of } \tau \rangle$ 
      using wf-replace-true append-g.simps check-branch-s-branchI by metis
    show  $\langle atom\ x \# (\Phi, \Theta, \mathcal{B}, \Gamma, \Delta, \Gamma, const) \rangle$ 
      using wf-replace-true append-g.simps check-branch-s-branchI fresh-prodN by metis
    show  $\langle \Theta ; \mathcal{B} ; \Gamma \vdash_{wf} \Delta \rangle$  using wf-replace-true append-g.simps check-branch-s-branchI by metis
  qed

```



```

qed
next
case (check-branch-list-consI  $\Theta \Phi \mathcal{B} \Gamma \Delta tid cons const v cs \tau dclist css$ )
then show ?case using wf-intros by metis
next
case (check-branch-list-finallI  $\Theta \Phi \mathcal{B} \Gamma \Delta tid cons const v cs \tau$ )
then show ?case using wf-intros by metis
next
case (check-ifi  $z \Theta \Phi \mathcal{B} \Gamma \Delta v s1 s2 \tau$ )
show ?case using wfS-ifi check-v-wf check-ifi b-of.simps by metis
next
case (check-let2I  $x \Theta \Phi \mathcal{B} G \Delta t s1 \tau s2$ )
show ?case proof
  show  $\langle \Theta ; \Phi ; \mathcal{B} ; G ; \Delta \vdash_{wf} s1 : b\text{-of } t \rangle$  using check-let2I b-of.simps by metis
  show  $\langle \Theta ; \mathcal{B} ; G \vdash_{wf} t \rangle$  using check-let2I check-s-check-branch-s-wf by metis
  show  $\langle \Theta ; \Phi ; \mathcal{B} ; (x, b\text{-of } t, TRUE) \#_{\Gamma} G ; \Delta \vdash_{wf} s2 : b\text{-of } \tau \rangle$ 
  using check-let2I(5) wf-replace-true2(2) append-g.simps check-let2I by metis
  show  $\langle atom\ x \# (\Phi, \Theta, \mathcal{B}, G, \Delta, s1, b\text{-of } \tau, t) \rangle$ 
  apply (simp add: fresh-prodN, intro conjI)
  using check-let2I(1) fresh-prod7 by simp+
qed
next
case (check-varI  $u \Theta \Phi \mathcal{B} \Gamma \Delta \tau' v \tau s$ )
show ?case proof
  show  $\langle \Theta ; \mathcal{B} ; \Gamma \vdash_{wf} \tau' \rangle$  using check-v-wf check-varI by metis
  show  $\langle \Theta ; \mathcal{B} ; \Gamma \vdash_{wf} v : b\text{-of } \tau' \rangle$  using check-v-wf check-varI by metis
  show  $\langle atom\ u \# (\Phi, \Theta, \mathcal{B}, \Gamma, \Delta, \tau', v, b\text{-of } \tau) \rangle$  using check-varI fresh-prodN u-fresh-b by metis
  show  $\langle \Theta ; \Phi ; \mathcal{B} ; \Gamma ; (u, \tau') \#_{\Delta} \Delta \vdash_{wf} s : b\text{-of } \tau \rangle$  using check-varI by metis
qed
next
case (check-assignI  $\Theta \Phi \mathcal{B} \Gamma \Delta u \tau v z \tau'$ )
then show ?case using wf-intros check-v-wf subtype-eq-base2 b-of.simps by metis
next
case (check-whileI  $\Theta \Phi \mathcal{B} \Gamma \Delta s1 z s2 \tau'$ )
thus ?case using wf-intros b-of.simps check-v-wf subtype-eq-base2 b-of.simps by metis
next
case (check-seqI  $\Theta \Phi \mathcal{B} \Gamma \Delta s1 z s2 \tau$ )
thus ?case using wf-intros b-of.simps by metis
next
case (check-caseI  $\Theta \Phi \mathcal{B} \Gamma \Delta tid dclist v cs \tau z$ )
show ?case proof
  show  $\langle \Theta ; \mathcal{B} ; \Gamma \vdash_{wf} v : B\text{-id } tid \rangle$  using check-caseI check-v-wf b-of.simps by metis
  show  $\langle AF\text{-typedef } tid\ dclist \in set \Theta \rangle$  using check-caseI by metis
  show  $\langle \Theta ; \mathcal{B} ; \Gamma \vdash_{wf} \Delta \rangle$  using check-caseI check-s-check-branch-s-wf by metis
  show  $\langle \Theta \vdash_{wf} \Phi \rangle$  using check-caseI check-s-check-branch-s-wf by metis
  show  $\langle \Theta ; \Phi ; \mathcal{B} ; \Gamma ; \Delta ; tid ; dclist \vdash_{wf} cs : b\text{-of } \tau \rangle$  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 $\Theta ; B \vdash_{wf} \Gamma \wedge wfT \Theta B \Gamma \tau \wedge wfPhi \Theta \Phi \wedge wfTh \Theta \wedge wfD \Theta B \Gamma \Delta \wedge wfS \Theta \Phi B \Gamma \Delta s$
(b-of τ)
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** $atom\ u \not\# \Delta$ **and** $atom\ u' \not\# \Delta$

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 $\dots = (u', (u' \leftrightarrow u) \cdot \tau') \#_{\Delta} \Delta$

using *assms flip-fresh-fresh* **by** *auto*

also have $\dots = (u', \tau') \#_{\Delta} \Delta$ **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* \Rightarrow *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 *v1::v* **and** *v2::v*

assumes $\Theta; \mathcal{B}; \Gamma \vdash v_1 \Rightarrow \tau_1$ **and** $\Theta; \mathcal{B}; \Gamma \vdash v_2 \Rightarrow \tau_2$

shows $\exists \tau. \Theta; \mathcal{B}; \Gamma \vdash V\text{-pair } v_1 \ v_2 \Rightarrow \tau \wedge b\text{-of } \tau = B\text{-pair } (b\text{-of } \tau_1) (b\text{-of } \tau_2)$

proof –

obtain *z1* **and** *b1* **and** *c1* **and** *z2* **and** *b2* **and** *c2* **where** *zbc*: $\tau_1 = (\llbracket z1 : b1 \mid c1 \rrbracket) \wedge \tau_2 = (\llbracket z2 : b2 \mid c2 \rrbracket)$

using *τ.exhaust* **by** *meson*

obtain *z::x* **where** *atom z* $\# (v_1, v_2, \Theta, \mathcal{B}, \Gamma)$ **using** *obtain-fresh*

by *blast*

hence *atom z* $\# (v_1, v_2) \wedge \text{atom } z \# (\Theta, \mathcal{B}, \Gamma)$ **using** *fresh-prodN* **by** *metis*

hence $\Theta; \mathcal{B}; \Gamma \vdash V\text{-pair } v_1 \ v_2 \Rightarrow \llbracket z : [b\text{-of } \tau_1, b\text{-of } \tau_2]^b \mid CE\text{-val } (V\text{-var } z) == CE\text{-val } (V\text{-pair } v_1 \ v_2) \rrbracket$

using *assms infer-v-pairI zbc* **by** *metis*

moreover obtain τ **where** $\tau = (\llbracket z : B\text{-pair } b1 \ b2 \mid CE\text{-val } (V\text{-var } z) == CE\text{-val } (V\text{-pair } v_1 \ v_2) \rrbracket)$ **by** *blast*

moreover hence $b\text{-of } \tau = B\text{-pair } (b\text{-of } \tau_1) (b\text{-of } \tau_2)$ **using** *b-of.simps zbc* **by** *presburger*

ultimately show *?thesis* **using** *b-of.simps* **by** *metis*

qed

lemma *infer-v-pair2I-zbc*:

fixes *v1::v* **and** *v2::v*

assumes $\Theta; \mathcal{B}; \Gamma \vdash v_1 \Rightarrow \tau_1$ **and** $\Theta; \mathcal{B}; \Gamma \vdash v_2 \Rightarrow \tau_2$

shows $\exists z \tau. \Theta; \mathcal{B}; \Gamma \vdash V\text{-pair } v_1 \ v_2 \Rightarrow \tau \wedge \tau = (\llbracket z : B\text{-pair } (b\text{-of } \tau_1) (b\text{-of } \tau_2) \mid C\text{-eq } (CE\text{-val } (V\text{-var } z)) (CE\text{-val } (V\text{-pair } v_1 \ v_2)) \rrbracket) \wedge \text{atom } z \# (v_1, v_2) \wedge \text{atom } z \# \Gamma$

proof –

obtain *z1* **and** *b1* **and** *c1* **and** *z2* **and** *b2* **and** *c2* **where** *zbc*: $\tau_1 = (\llbracket z1 : b1 \mid c1 \rrbracket) \wedge \tau_2 = (\llbracket z2 : b2 \mid c2 \rrbracket)$

using *τ.exhaust* **by** *meson*

obtain *z::x* **where** $\text{atom } z \# (v_1, v_2, \Gamma, \Theta, \mathcal{B})$ **using** *obtain-fresh*

by *blast*

hence *vinf*: $\Theta; \mathcal{B}; \Gamma \vdash V\text{-pair } v_1 \ v_2 \Rightarrow \llbracket z : [b\text{-of } \tau_1, b\text{-of } \tau_2]^b \mid CE\text{-val } (V\text{-var } z) == CE\text{-val } (V\text{-pair } v_1 \ v_2) \rrbracket$

using *assms infer-v-pairI* **by** *simp*

moreover obtain τ **where** $\tau = (\llbracket z : B\text{-pair } b1 \ b2 \mid CE\text{-val } (V\text{-var } z) == CE\text{-val } (V\text{-pair } v_1 \ v_2) \rrbracket)$ **by** *blast*

moreover have $b\text{-of } \tau_1 = b1 \wedge b\text{-of } \tau_2 = b2$ **using** *zbc b-of.simps* **by** *auto*

ultimately have $\Theta; \mathcal{B}; \Gamma \vdash V\text{-pair } v_1 \ v_2 \Rightarrow \tau \wedge \tau = (\llbracket z : B\text{-pair } (b\text{-of } \tau_1) (b\text{-of } \tau_2) \mid CE\text{-val } (V\text{-var } z) == CE\text{-val } (V\text{-pair } v_1 \ v_2) \rrbracket)$ **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 $\exists \tau_1 \ \tau_2 \ z . \ \Theta; \mathcal{B}; \Gamma \vdash v_1 \Rightarrow \tau_1 \wedge \Theta; \mathcal{B}; \Gamma \vdash v_2 \Rightarrow \tau_2 \wedge$

$\tau = (\llbracket z : B\text{-pair } (b\text{-of } \tau_1) \ (b\text{-of } \tau_2) \mid C\text{-eq } (CE\text{-val } (V\text{-var } z)) \ (CE\text{-val } (V\text{-pair } v_1 \ v_2)) \rrbracket) \wedge$

$atom \ z \ \# \ (v_1, v_2)$

proof –

obtain z **and** $t1$ **and** $t2$ **where**

$\tau = (\llbracket z : B\text{-pair } (b\text{-of } t1) \ (b\text{-of } t2) \mid CE\text{-val } (V\text{-var } z) == CE\text{-val } (V\text{-pair } v_1 \ v_2) \rrbracket) \wedge$

$atom \ z \ \# \ (v_1, v_2) \wedge \ \Theta; \mathcal{B}; \Gamma \vdash v_1 \Rightarrow t1 \wedge \ \Theta; \mathcal{B}; \Gamma \vdash v_2 \Rightarrow t2$ **using** *infer-v-elim3*[*OF assms*

] by metis

thus *?thesis* **using** *b-of.simps* **by metis**

qed

12.7.2 Expressions

lemma *infer-e-app2E*:

fixes $\Phi::\Phi$ **and** $\Theta::\Theta$

assumes $\Theta; \Phi; \mathcal{B}; \Gamma; \Delta \vdash AE\text{-app } f \ v \Rightarrow \tau$

shows $\exists x \ b \ c \ s' \ \tau'. \ wfd \ \Theta \ \mathcal{B} \ \Gamma \ \Delta \wedge Some \ (AF\text{-fundef } f \ (AF\text{-fun-typ-none } (AF\text{-fun-typ } x \ b \ c \ \tau' \ s')))$
 $= lookup\text{-fun } \Phi \ f \wedge \ \Theta \vdash_{wf} \Phi \wedge$

$\Theta; \mathcal{B}; \Gamma \vdash v \Leftarrow \llbracket x : b \mid c \rrbracket \wedge \tau = \tau'[x::=v]_{\tau v} \wedge atom \ x \ \# \ (\Theta, \Phi, \mathcal{B}, \Gamma, \Delta, v, \tau)$

using *infer-e-elim6*[*OF assms*] *b-of.simps subst-defs* **by metis**

lemma *infer-e-appP2E*:

fixes $\Phi::\Phi$ **and** $\Theta::\Theta$

assumes $\Theta; \Phi; \mathcal{B}; \Gamma; \Delta \vdash AE\text{-appP } f \ b \ v \Rightarrow \tau$

shows $\exists bv \ x \ ba \ c \ s' \ \tau'. \ wfd \ \Theta \ \mathcal{B} \ \Gamma \ \Delta \wedge Some \ (AF\text{-fundef } f \ (AF\text{-fun-typ-some } bv \ (AF\text{-fun-typ } x \ ba \ c \ \tau' \ s')))$
 $= lookup\text{-fun } \Phi \ f \wedge \ \Theta \vdash_{wf} \Phi \wedge \ \Theta; \mathcal{B} \vdash_{wf} b \wedge$

$(\Theta; \mathcal{B}; \Gamma \vdash v \Leftarrow \llbracket x : ba[bv::=b]_{bb} \mid c[bv::=b]_{cb} \rrbracket) \wedge (\tau = \tau'[bv::=b]_{\tau b}[x::=v]_{\tau v}) \wedge atom \ x \ \# \ \Gamma \wedge$

$atom \ bv \ \# \ v$

proof –

obtain $bv \ x \ ba \ c \ s' \ \tau'$ **where** $*:wfd \ \Theta \ \mathcal{B} \ \Gamma \ \Delta \wedge Some \ (AF\text{-fundef } f \ (AF\text{-fun-typ-some } bv \ (AF\text{-fun-typ } x \ ba \ c \ \tau' \ s')))$
 $= lookup\text{-fun } \Phi \ f \wedge \ \Theta \vdash_{wf} \Phi \wedge \ \Theta; \mathcal{B} \vdash_{wf} b \wedge$

$(\Theta; \mathcal{B}; \Gamma \vdash v \Leftarrow \llbracket x : ba[bv::=b]_{bb} \mid c[bv::=b]_{cb} \rrbracket) \wedge (\tau = \tau'[bv::=b]_{\tau b}[x::=v]_{\tau v}) \wedge atom \ x \ \# \ \Gamma \wedge$

$atom \ bv \ \# \ (\Theta, \Phi, \mathcal{B}, \Gamma, \Delta, b, v, \tau)$

using *infer-e-elim21*[*OF assms*] *subst-defs* **by metis**

moreover then have $atom \ bv \ \# \ v$ **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 $\Theta; \mathcal{B}; \Gamma \vdash \tau 1 \lesssim \tau 2$ **and** $toSet \ \Gamma \subseteq toSet \ \Gamma'$ **and** $\Theta; \mathcal{B} \vdash_{wf} \Gamma'$

shows $\Theta; \mathcal{B}; \Gamma' \vdash \tau 1 \lesssim \tau 2$

using *assms proof*(*nominal-induct* $\tau 2$ *avoiding*: Γ' *rule*: *subtype.strong-induct*)

```

case (subtype-baseI x  $\Theta$   $\mathcal{B}$   $\Gamma$  z c z' c' b)
show ?case proof
  show *: $\Theta$ ;  $\mathcal{B}$ ;  $\Gamma'$   $\vdash_{wf}$   $\langle z : b \mid c \rangle$  using wfT-weakening subtype-baseI by metis
  show  $\Theta$ ;  $\mathcal{B}$ ;  $\Gamma'$   $\vdash_{wf}$   $\langle z' : b \mid c' \rangle$  using wfT-weakening subtype-baseI by metis
  show atom x  $\#$  ( $\Theta$ ,  $\mathcal{B}$ ,  $\Gamma'$ , z, c, z', c') using subtype-baseI fresh-Pair by metis
  have toSet ((x, b, c[z::=V-var x]v)  $\#_{\Gamma}$   $\Gamma$ )  $\subseteq$  toSet ((x, b, c[z::=V-var x]v)  $\#_{\Gamma}$   $\Gamma'$ ) using subtype-baseI
  toSet.simps by blast
  moreover have  $\Theta$  ;  $\mathcal{B} \vdash_{wf}$  (x, b, c[z::=V-var x]v)  $\#_{\Gamma}$   $\Gamma'$  using wfT-wf-cons3[OF *, of x]
  subtype-baseI fresh-Pair subst-defs by metis
  ultimately show  $\Theta$ ;  $\mathcal{B}$ ; (x, b, c[z::=V-var x]v)  $\#_{\Gamma}$   $\Gamma'$   $\models$  c'[z':=V-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  $\Theta$ ;  $\mathcal{B}$  ;  $\Gamma \vdash v \Rightarrow \tau$  and toSet  $\Gamma \subseteq$  toSet  $\Gamma'$  and  $\Theta$  ;  $\mathcal{B} \vdash_{wf}$   $\Gamma'$ 
shows  $\Theta$ ;  $\mathcal{B}$  ;  $\Gamma' \vdash v \Rightarrow \tau$ 
using assms proof(nominal-induct avoiding:  $\Gamma'$  rule: infer-v.strong-induct)
  case (infer-v-varI  $\Theta$   $\mathcal{B}$   $\Gamma$  b c x' z)
  show ?case proof
    show  $\langle \Theta ; \mathcal{B} \vdash_{wf} \Gamma' \rangle$  using infer-v-varI by auto
    show  $\langle \text{Some } (b, c) = \text{lookup } \Gamma' \ x' \rangle$  using infer-v-varI lookup-weakening by metis
    show  $\langle \text{atom } z \# x' \rangle$  using infer-v-varI by auto
    show  $\langle \text{atom } z \# (\Theta, \mathcal{B}, \Gamma') \rangle$  using infer-v-varI by auto
  qed
next
  case (infer-v-litI  $\Theta$   $\mathcal{B}$   $\Gamma$  l  $\tau$ )
  then show ?case using infer-v.intros by simp
next
  case (infer-v-pairI z v1 v2  $\Theta$   $\mathcal{B}$   $\Gamma$  t1 t2)
  then show ?case using infer-v.intros by simp
next
  case (infer-v-consI s dclist  $\Theta$  dc tc  $\mathcal{B}$   $\Gamma$  v tv z)
  show ?case proof

    show  $\langle \text{AF-typedef } s \text{ dclist} \in \text{set } \Theta \rangle$  using infer-v-consI by auto

    show  $\langle (dc, tc) \in \text{set } \text{dclist} \rangle$  using infer-v-consI by auto

    show  $\langle \Theta ; \mathcal{B} ; \Gamma' \vdash v \Rightarrow tv \rangle$  using infer-v-consI by auto

    show  $\langle \Theta ; \mathcal{B} ; \Gamma' \vdash tv \lesssim tc \rangle$  using infer-v-consI subtype-weakening by auto

    show  $\langle \text{atom } z \# v \rangle$  using infer-v-consI by auto

    show  $\langle \text{atom } z \# (\Theta, \mathcal{B}, \Gamma') \rangle$  using infer-v-consI by auto
  qed
next

```

```

case (infer-v-conspI s bv dclist  $\Theta$  dc tc  $\mathcal{B}$   $\Gamma$  v tv b z)
show ?case proof

show  $\langle AF\text{-typedef-poly } s \text{ bv dclist} \in \text{set } \Theta \rangle$  using infer-v-conspI by auto

show  $\langle (dc, tc) \in \text{set dclist} \rangle$  using infer-v-conspI by auto

show  $\langle \Theta; \mathcal{B}; \Gamma' \vdash v \Rightarrow tv \rangle$  using infer-v-conspI by auto

show  $\langle \Theta; \mathcal{B}; \Gamma' \vdash tv \lesssim tc[bv::=b]_{\tau b} \rangle$  using infer-v-conspI subtype-weakening by auto

show  $\langle atom \ z \ \# \ (\Theta, \mathcal{B}, \Gamma', v, b) \rangle$  using infer-v-conspI by auto

show  $\langle atom \ bv \ \# \ (\Theta, \mathcal{B}, \Gamma', v, b) \rangle$  using infer-v-conspI by auto

show  $\langle \Theta; \mathcal{B} \vdash_{wf} b \rangle$  using infer-v-conspI by auto
qed
qed

lemma check-v-g-weakening:
  fixes  $e::e$  and  $\Gamma'::\Gamma$ 
  assumes  $\Theta; \mathcal{B}; \Gamma \vdash v \Leftarrow \tau$  and  $toSet \ \Gamma \subseteq toSet \ \Gamma'$  and  $\Theta; \mathcal{B} \vdash_{wf} \Gamma'$ 
  shows  $\Theta; \mathcal{B}; \Gamma' \vdash v \Leftarrow \tau$ 
  using subtype-weakening infer-v-g-weakening check-v-elim check-v-subtypeI assms by metis

lemma infer-e-g-weakening:
  fixes  $e::e$  and  $\Gamma'::\Gamma$ 
  assumes  $\Theta; \Phi; \mathcal{B}; \Gamma; \Delta \vdash e \Rightarrow \tau$  and  $toSet \ \Gamma \subseteq toSet \ \Gamma'$  and  $\Theta; \mathcal{B} \vdash_{wf} \Gamma'$ 
  shows  $\Theta; \Phi; \mathcal{B}; \Gamma'; \Delta \vdash e \Rightarrow \tau$ 
  using assms proof (nominal-induct  $\tau$  avoiding:  $\Gamma'$  rule: infer-e.strong-induct)
  case (infer-e-valI  $\Theta \mathcal{B} \Gamma \Delta' \Phi v \tau$ )
  then show ?case using infer-v-g-weakening wf-weakening infer-e.intros by metis
next
case (infer-e-plusI  $\Theta \mathcal{B} \Gamma \Delta \Phi v1 z1 c1 v2 z2 c2 z3$ )

  obtain  $z'::x$  where  $z': atom \ z' \ \# \ v1 \wedge atom \ z' \ \# \ v2 \wedge atom \ z' \ \# \ \Gamma'$  using obtain-fresh fresh-prod3 by metis
  moreover hence  $*(\llbracket z3 : B\text{-int} \mid CE\text{-val} \ (V\text{-var } z3) \rrbracket == CE\text{-op Plus } [v1]^{ce} [v2]^{ce} \rrbracket = (\llbracket z' : B\text{-int} \mid CE\text{-val} \ (V\text{-var } z') \rrbracket == CE\text{-op Plus } [v1]^{ce} [v2]^{ce} \rrbracket)$ 
  using infer-e-plusI type-e-eq ce.fresh fresh-e-opp by auto

  have  $\Theta; \Phi; \mathcal{B}; \Gamma'; \Delta \vdash AE\text{-op Plus } v1 \ v2 \Rightarrow \llbracket z' : B\text{-int} \mid CE\text{-val} \ (V\text{-var } z') \rrbracket == CE\text{-op Plus } [v1]^{ce} [v2]^{ce} \rrbracket$  proof
    show  $\langle \Theta; \mathcal{B}; \Gamma' \vdash_{wf} \Delta \rangle$  using wf-weakening infer-e-plusI by auto
    show  $\langle \Theta \vdash_{wf} \Phi \rangle$  using infer-e-plusI by auto
    show  $\langle \Theta; \mathcal{B}; \Gamma' \vdash v1 \Rightarrow \llbracket z1 : B\text{-int} \mid c1 \rrbracket \rangle$  using infer-v-g-weakening infer-e-plusI by auto
    show  $\langle \Theta; \mathcal{B}; \Gamma' \vdash v2 \Rightarrow \llbracket z2 : B\text{-int} \mid c2 \rrbracket \rangle$  using infer-v-g-weakening infer-e-plusI by auto
    show  $\langle atom \ z' \ \# \ AE\text{-op Plus } v1 \ v2 \rangle$  using  $z'$  by auto
    show  $\langle atom \ z' \ \# \ \Gamma' \rangle$  using  $z'$  by auto
  qed
  thus ?case using * by metis

```

next

case (*infer-e-leqI* $\Theta \mathcal{B} \Gamma \Delta \Phi \ v1 \ z1 \ c1 \ v2 \ z2 \ c2 \ z3$)
 obtain $z'::x$ where $z': \text{atom } z' \# v1 \wedge \text{atom } z' \# v2 \wedge \text{atom } z' \# \Gamma'$ using *obtain-fresh fresh-prod3* by *metis*

moreover hence $\ast: \llbracket z3 : B\text{-bool} \mid CE\text{-val } (V\text{-var } z3) \rrbracket == CE\text{-op } LEq \ [v1]^{ce} \ [v2]^{ce} \ \rrbracket = (\llbracket z' : B\text{-bool} \mid CE\text{-val } (V\text{-var } z') \rrbracket == CE\text{-op } LEq \ [v1]^{ce} \ [v2]^{ce} \ \rrbracket)$
 using *infer-e-leqI type-e-eq ce.fresh fresh-e-opp* by *auto*

have $\Theta ; \Phi ; \mathcal{B} ; \Gamma' ; \Delta \vdash AE\text{-op } LEq \ v1 \ v2 \Rightarrow \llbracket z' : B\text{-bool} \mid CE\text{-val } (V\text{-var } z') \rrbracket == CE\text{-op } LEq \ [v1]^{ce} \ [v2]^{ce} \ \rrbracket$ **proof**

show $\langle \Theta ; \mathcal{B} ; \Gamma' \vdash_{wf} \Delta \rangle$ using *wf-weakening infer-e-leqI* by *auto*

show $\langle \Theta \vdash_{wf} \Phi \rangle$ using *infer-e-leqI* by *auto*

show $\langle \Theta ; \mathcal{B} ; \Gamma' \vdash v1 \Rightarrow \llbracket z1 : B\text{-int} \mid c1 \rrbracket \rangle$ using *infer-v-g-weakening infer-e-leqI* by *auto*

show $\langle \Theta ; \mathcal{B} ; \Gamma' \vdash v2 \Rightarrow \llbracket z2 : B\text{-int} \mid c2 \rrbracket \rangle$ using *infer-v-g-weakening infer-e-leqI* by *auto*

show $\langle \text{atom } z' \# AE\text{-op } LEq \ v1 \ v2 \rangle$ using z' by *auto*

show $\langle \text{atom } z' \# \Gamma' \rangle$ using z' by *auto*

qed

thus *?case* using \ast by *metis*

next

case (*infer-e-appI* $\Theta \mathcal{B} \Gamma \Delta \Phi \ f \ x \ b \ c \ \tau' \ s' \ v \ \tau$)

show *?case* **proof**

show $\Theta ; \mathcal{B} ; \Gamma' \vdash_{wf} \Delta$ using *wf-weakening infer-e-appI* by *auto*

show $\Theta \vdash_{wf} \Phi$ using *wf-weakening infer-e-appI* by *auto*

show *Some* (*AF-fundef* f (*AF-fun-typ-none* (*AF-fun-typ* $x \ b \ c \ \tau' \ s'$))) = *lookup-fun* $\Phi \ f$ using *wf-weakening infer-e-appI* by *auto*

show $\Theta ; \mathcal{B} ; \Gamma' \vdash v \Leftarrow \llbracket x : b \mid c \rrbracket$ using *wf-weakening infer-e-appI check-v-g-weakening* by *auto*

show $\text{atom } x \# (\Theta, \Phi, \mathcal{B}, \Gamma', \Delta, v, \tau)$ using *wf-weakening infer-e-appI* by *auto*

show $\tau'[x::v]_v = \tau$ using *wf-weakening infer-e-appI* by *auto*

qed

next

case (*infer-e-appPI* $\Theta \mathcal{B} \Gamma \Delta \Phi \ b' \ f \ bv \ x \ b \ c \ \tau' \ s' \ v \ \tau$)

hence $\ast: \Theta ; \Phi ; \mathcal{B} ; \Gamma ; \Delta \vdash AE\text{-appP } f \ b' \ v \Rightarrow \tau$ using *Typing.infer-e-appPI* by *auto*

obtain $x'::x$ where $x': \text{atom } x' \# (s', c, \tau', (\Gamma', v, \tau)) \wedge (AF\text{-fundef } f \ (AF\text{-fun-typ-some } bv \ (AF\text{-fun-typ } x \ b \ c \ \tau' \ s')))) = (AF\text{-fundef } f \ (AF\text{-fun-typ-some } bv \ (AF\text{-fun-typ } x' \ b \ ((x' \leftrightarrow x) \cdot c) \ ((x' \leftrightarrow x) \cdot \tau') \ ((x' \leftrightarrow x) \cdot s'))))$

using *obtain-fresh-fun-def[of s' c \tau' (\Gamma', v, \tau) f x b]*

by (*metis fun-def.eq-iff fun-typ-q.eq-iff(2)*)

hence $\ast: \llbracket x : b \mid c \rrbracket = \llbracket x' : b \mid (x' \leftrightarrow x) \cdot c \rrbracket$

using *fresh-PairD(1) fresh-PairD(2) type-eq-flip* by *blast*

have $\text{atom } x' \# \Gamma$ 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* $x' \ b \ ((x' \leftrightarrow x) \cdot c) \ ((x' \leftrightarrow x) \cdot \tau') \ ((x' \leftrightarrow x) \cdot s')$)))

$\leftrightarrow x) \cdot s')))) = \text{lookup-fun } \Phi f \text{ using } x' \text{ infer-e-appPI by argo}$
show $\Theta; \mathcal{B}; \Gamma' \vdash v \Leftarrow \llbracket x' : b[bv::=b]_b \mid ((x' \leftrightarrow x) \cdot c)[bv::=b]_b \rrbracket$ **using** **
 $\tau.\text{eq-iff check-v-g-weakening infer-e-appPI.hyps infer-e-appPI.premis}(1) \text{ infer-e-appPI.premis subst-defs}$
 subst-tb.simps **by** *metis*
show $\text{atom } x' \# \Gamma'$ **using** $x' \text{ fresh-prodN}$ **by** *metis*

have $\text{atom } x \# (v, \tau) \wedge \text{atom } x' \# (v, \tau)$ **using** $x' \text{ infer-e-fresh}[OF \ *] \text{ e.fresh}(2) \text{ fresh-Pair}$
 $\text{infer-e-appPI } \langle \text{atom } x' \# \Gamma \rangle \text{ e.fresh}$ **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 $(\text{metis subst-b-}\tau\text{-def})$
ultimately show $((x' \leftrightarrow x) \cdot \tau')[bv::=b]_b[x'::=v]_v = \tau$
using $\text{infer-e-appPI subst-tv-flip subst-defs}$ **by** *metis*

show $\text{atom } bv \# (\Theta, \Phi, \mathcal{B}, \Gamma', \Delta, b', v, \tau)$ **using** $\text{infer-e-appPI fresh-prodN}$ **by** *metis*
qed

next
case $(\text{infer-e-fstI } \Theta \ \mathcal{B} \ \Gamma \ \Delta \ \Phi \ v \ z'' \ b1 \ b2 \ c \ z)$

obtain $z'::x$ **where** $*$: $\text{atom } z' \# \Gamma' \wedge \text{atom } z' \# v \wedge \text{atom } z' \# c$ **using** $\text{obtain-fresh-z fresh-Pair}$ **by** *metis*
hence $**::\llbracket z : b1 \mid \text{CE-val } (V\text{-var } z) == \text{CE-fst } [v]^{ce} \rrbracket = \llbracket z' : b1 \mid \text{CE-val } (V\text{-var } z') == \text{CE-fst } [v]^{ce} \rrbracket$
using $\text{type-e-eq infer-e-fstI v.fresh e.fresh ce.fresh obtain-fresh-z fresh-Pair}$ **by** *metis*

have $\Theta; \Phi; \mathcal{B}; \Gamma'; \Delta \vdash \text{AE-fst } v \Rightarrow \llbracket z' : b1 \mid \text{CE-val } (V\text{-var } z') == \text{CE-fst } [v]^{ce} \rrbracket$ **proof**
show $\langle \Theta; \mathcal{B}; \Gamma' \vdash_{wf} \Delta \rangle$ **using** $\text{wf-weakening infer-e-fstI}$ **by** *auto*
show $\langle \Theta \vdash_{wf} \Phi \rangle$ **using** $\text{wf-weakening infer-e-fstI}$ **by** *auto*
show $\Theta; \mathcal{B}; \Gamma' \vdash v \Rightarrow \llbracket z'' : B\text{-pair } b1 \ b2 \mid c \rrbracket$ **using** $\text{infer-v-g-weakening infer-e-fstI}$ **by** *metis*
show $\text{atom } z' \# \text{AE-fst } v$ **using** $*$ **by** *auto*
show $\text{atom } z' \# \Gamma'$ **using** $*$ **by** *auto*
qed
thus $?case$ **using** $*$ **by** *metis*

next
case $(\text{infer-e-sndI } \Theta \ \mathcal{B} \ \Gamma \ \Delta \ \Phi \ v \ z'' \ b1 \ b2 \ c \ z)$

obtain $z'::x$ **where** $*$: $\text{atom } z' \# \Gamma' \wedge \text{atom } z' \# v \wedge \text{atom } z' \# c$ **using** $\text{obtain-fresh-z fresh-Pair}$ **by** *metis*
hence $**::\llbracket z : b2 \mid \text{CE-val } (V\text{-var } z) == \text{CE-snd } [v]^{ce} \rrbracket = \llbracket z' : b2 \mid \text{CE-val } (V\text{-var } z') == \text{CE-snd } [v]^{ce} \rrbracket$
using $\text{type-e-eq infer-e-sndI e.fresh ce.fresh obtain-fresh-z fresh-Pair}$ **by** *metis*

have $\Theta; \Phi; \mathcal{B}; \Gamma'; \Delta \vdash \text{AE-snd } v \Rightarrow \llbracket z' : b2 \mid \text{CE-val } (V\text{-var } z') == \text{CE-snd } [v]^{ce} \rrbracket$ **proof**
show $\langle \Theta; \mathcal{B}; \Gamma' \vdash_{wf} \Delta \rangle$ **using** $\text{wf-weakening infer-e-sndI}$ **by** *auto*
show $\langle \Theta \vdash_{wf} \Phi \rangle$ **using** $\text{wf-weakening infer-e-sndI}$ **by** *auto*
show $\Theta; \mathcal{B}; \Gamma' \vdash v \Rightarrow \llbracket z'' : B\text{-pair } b1 \ b2 \mid c \rrbracket$ **using** $\text{infer-v-g-weakening infer-e-sndI}$ **by** *metis*
show $\text{atom } z' \# \text{AE-snd } v$ **using** $*$ **by** *auto*
show $\text{atom } z' \# \Gamma'$ **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** $*$: $\text{atom } z' \# \Gamma' \wedge \text{atom } z' \# v \wedge \text{atom } z' \# c$ **using** *obtain-fresh-z fresh-Pair* **by** *metis*

hence $**:\llbracket z : B\text{-int} \mid \text{CE-val } (V\text{-var } z) \rrbracket = \llbracket z' : B\text{-int} \mid \text{CE-val } (V\text{-var } z') \rrbracket$
 $= \llbracket \text{CE-len } [v]^{ce} \rrbracket$

using *type-e-eq infer-e-lenI e.fresh ce.fresh obtain-fresh-z fresh-Pair* **by** *metis*

have $\Theta ; \Phi ; \mathcal{B} ; \Gamma' ; \Delta \vdash \text{AE-len } v \Rightarrow \llbracket z' : B\text{-int} \mid \text{CE-val } (V\text{-var } z') \rrbracket = \llbracket \text{CE-len } [v]^{ce} \rrbracket$ **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 $\Theta ; \mathcal{B} ; \Gamma' \vdash v \Rightarrow \llbracket z'' : B\text{-bitvec} \mid c \rrbracket$ **using** *infer-v-g-weakening infer-e-lenI* **by** *metis*

show $\text{atom } z' \# \text{AE-len } v$ **using** $* \text{e.supp}$ **by** *auto*

show $\text{atom } z' \# \Gamma'$ **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** $*$: $\text{atom } z' \# \Gamma' \wedge \text{atom } z' \# v1 \wedge \text{atom } z' \# v2$ **using** *obtain-fresh-z fresh-Pair* **by** *metis*

hence $**:\llbracket z3 : B\text{-bitvec} \mid \text{CE-val } (V\text{-var } z3) \rrbracket = \llbracket \text{CE-concat } [v1]^{ce} [v2]^{ce} \rrbracket = \llbracket z' : B\text{-bitvec} \mid \text{CE-val } (V\text{-var } z') \rrbracket$
 $= \llbracket \text{CE-concat } [v1]^{ce} [v2]^{ce} \rrbracket$

using *type-e-eq infer-e-concatI e.fresh ce.fresh obtain-fresh-z fresh-Pair* **by** *metis*

have $\Theta ; \Phi ; \mathcal{B} ; \Gamma' ; \Delta \vdash \text{AE-concat } v1 \ v2 \Rightarrow \llbracket z' : B\text{-bitvec} \mid \text{CE-val } (V\text{-var } z') \rrbracket = \llbracket \text{CE-concat } [v1]^{ce} [v2]^{ce} \rrbracket$ **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 $\Theta ; \mathcal{B} ; \Gamma' \vdash v1 \Rightarrow \llbracket z1 : B\text{-bitvec} \mid c1 \rrbracket$ **using** *infer-v-g-weakening infer-e-concatI* **by** *metis*

show $\Theta ; \mathcal{B} ; \Gamma' \vdash v2 \Rightarrow \llbracket z2 : B\text{-bitvec} \mid c2 \rrbracket$ **using** *infer-v-g-weakening infer-e-concatI* **by** *metis*

show $\text{atom } z' \# \text{AE-concat } v1 \ v2$ **using** $* \text{e.supp}$ **by** *auto*

show $\text{atom } z' \# \Gamma'$ **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 $\Theta ; \mathcal{B} ; \Gamma' \vdash_{wf} \Delta$ **using** *infer-e-splitI wf-weakening* **by** *auto*

show $\Theta \vdash_{wf} \Phi$ **using** *infer-e-splitI wf-weakening* **by** *auto*

show $\Theta ; \mathcal{B} ; \Gamma' \vdash v1 \Rightarrow \llbracket z1 : B\text{-bitvec} \mid c1 \rrbracket$ **using** *infer-v-g-weakening infer-e-splitI* **by** *metis*

show $\Theta ; \mathcal{B} ; \Gamma' \vdash v2 \Leftarrow \llbracket z2 : B\text{-int} \mid \llbracket \text{leq } [\llbracket L\text{-num } 0 \rrbracket^v]^{ce} [\llbracket z2 \rrbracket^v]^{ce} \rrbracket^{ce} = \llbracket \llbracket L\text{-true} \rrbracket^v \rrbracket^{ce}$
 $\text{AND } \llbracket \text{leq } [\llbracket z2 \rrbracket^v]^{ce} [\llbracket v1 \rrbracket^{ce}]^{ce} \rrbracket^{ce} = \llbracket \llbracket L\text{-true} \rrbracket^v \rrbracket^{ce} \rrbracket$

using *check-v-g-weakening infer-e-splitI* **by** *metis*

show $\text{atom } z1 \# \text{AE-split } v1 \ v2$ **using** *infer-e-splitI* **by** *auto*

show $\text{atom } z1 \# \Gamma'$ **using** *infer-e-splitI* **by** *auto*

show $\text{atom } z2 \# \text{AE-split } v1 \ v2$ **using** *infer-e-splitI* **by** *auto*

```

  show atom z2 #  $\Gamma'$  using infer-e-splitI by auto
  show atom z3 # AE-split v1 v2 using infer-e-splitI by auto
  show atom z3 #  $\Gamma'$  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  $\Theta ; \Phi ; \mathcal{B} ; \Gamma ; \Delta \vdash e \Rightarrow \tau$  and setD  $\Delta \subseteq \text{setD } \Delta'$  and wfD  $\Theta \ \mathcal{B} \ \Gamma \ \Delta'$ 
  shows  $\Theta ; \Phi ; \mathcal{B} ; \Gamma ; \Delta' \vdash e \Rightarrow \tau$ 
  using assms by(nominal-induct  $\tau$  avoiding:  $\Delta'$  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  $\Theta ; \mathcal{B} ; \Gamma \vdash v \Rightarrow \tau$  and atom x #  $\Gamma$ 
  shows atom x # 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  $\Gamma'::\Gamma$  and  $\Theta::\Theta$  and css::branch-list
  shows check-s  $\Theta \ \Phi \ \mathcal{B} \ \Gamma \ \Delta \ s \ t \Longrightarrow \text{toSet } \Gamma \subseteq \text{toSet } \Gamma' \Longrightarrow \Theta ; \mathcal{B} \vdash_{wf} \Gamma' \Longrightarrow \text{check-s } \Theta \ \Phi \ \mathcal{B} \ \Gamma' \ \Delta \ s \ t$  and
    check-branch-s  $\Theta \ \Phi \ \mathcal{B} \ \Gamma \ \Delta \ \text{tid cons const } v \ cs \ t \Longrightarrow \text{toSet } \Gamma \subseteq \text{toSet } \Gamma' \Longrightarrow \Theta ; \mathcal{B} \vdash_{wf} \Gamma' \Longrightarrow \text{check-branch-s } \Theta \ \Phi \ \mathcal{B} \ \Gamma' \ \Delta \ \text{tid cons const } v \ cs \ t$  and
    check-branch-list  $\Theta \ \Phi \ \mathcal{B} \ \Gamma \ \Delta \ \text{tid dclist } v \ css \ t \Longrightarrow \text{toSet } \Gamma \subseteq \text{toSet } \Gamma' \Longrightarrow \Theta ; \mathcal{B} \vdash_{wf} \Gamma' \Longrightarrow \text{check-branch-list } \Theta \ \Phi \ \mathcal{B} \ \Gamma' \ \Delta \ \text{tid dclist } v \ css \ t$ 

```

proof(nominal-induct t and t and t avoiding: Γ' rule: check-s-check-branch-s-check-branch-list.strong-induct)

case (check-valI $\Theta \ \mathcal{B} \ \Gamma \ \Delta' \ \Phi \ v \ \tau' \ \tau$)

then show ?case using Typing.check-valI infer-v-g-weakening wf-weakening subtype-weakening by metis

next

case (check-letI $x \ \Theta \ \Phi \ \mathcal{B} \ \Gamma \ \Delta \ e \ \tau \ z \ s \ b \ c$)

hence xf:atom x # Γ' by metis

show ?case proof

show atom x # $(\Theta, \Phi, \mathcal{B}, \Gamma', \Delta, e, \tau)$ using check-letI using fresh-prod4 xf by metis

show $\Theta ; \Phi ; \mathcal{B} ; \Gamma' ; \Delta \vdash e \Rightarrow \llbracket z : b \mid c \rrbracket$ using infer-e-g-weakening check-letI by metis

show atom z # $(x, \Theta, \Phi, \mathcal{B}, \Gamma', \Delta, e, \tau, s)$

by(unfold fresh-prodN, auto simp add: check-letI fresh-prodN)

have toSet $((x, b, c[z::=V\text{-var } x]_v) \#_{\Gamma} \Gamma) \subseteq \text{toSet } ((x, b, c[z::=V\text{-var } x]_v) \#_{\Gamma} \Gamma')$ using check-letI toSet.simps

by (metis Un-commute Un-empty-right Un-insert-right insert-mono)

moreover hence $\Theta ; \mathcal{B} \vdash_{wf} ((x, b, c[z::=V\text{-var } x]_v) \#_{\Gamma} \Gamma')$ using check-letI wfG-cons-weakening check-s-wf by metis

ultimately show $\Theta ; \Phi ; \mathcal{B} ; (x, b, c[z::=V\text{-var } x]_v) \#_{\Gamma} \Gamma' ; \Delta \vdash s \Leftarrow \tau$ using check-letI by metis

qed

next

case (check-let2I $x \ \Theta \ \Phi \ \mathcal{B} \ G \ \Delta \ t \ s1 \ \tau \ s2$)

show ?case proof

show atom x # $(\Theta, \Phi, \mathcal{B}, \Gamma', \Delta, t, s1, \tau)$ using check-let2I using fresh-prod4 by auto

show $\Theta ; \Phi ; \mathcal{B} ; \Gamma' ; \Delta \vdash s1 \Leftarrow t$ using check-let2I by metis

have toSet $((x, b\text{-of } t, c\text{-of } t \ x) \#_{\Gamma} G) \subseteq \text{toSet } ((x, b\text{-of } t, c\text{-of } t \ x) \#_{\Gamma} \Gamma')$ using check-let2I by

```

auto
  moreover hence  $\Theta ; \mathcal{B} \vdash_{wf} ((x, b\text{-of } t, c\text{-of } t \ x) \#_{\Gamma} \Gamma')$  using check-let2I wfG-cons-weakening
check-s-wf by metis
  ultimately show  $\Theta ; \Phi ; \mathcal{B} ; (x, b\text{-of } t, c\text{-of } t \ x) \#_{\Gamma} \Gamma' ; \Delta \vdash s2 \Leftarrow \tau$  using check-let2I by metis
qed
next
case (check-branch-list-consI  $\Theta \ \Phi \ \mathcal{B} \ \Gamma \ \Delta \ tid \ dclist' \ v \ cs \ \tau \ css \ dclist$ )
  thus ?case using Typing.check-branch-list-consI by metis
next
case (check-branch-list-finalI  $\Theta \ \Phi \ \mathcal{B} \ \Gamma \ \Delta \ tid \ dclist' \ v \ cs \ \tau \ dclist$ )
  thus ?case using Typing.check-branch-list-finalI by metis
next
case (check-branch-s-branchI  $\Theta \ \mathcal{B} \ \Gamma \ \Delta \ \tau \ const \ x \ \Phi \ tid \ cons \ v \ s$ )
  show ?case proof
    show  $\Theta ; \mathcal{B} ; \Gamma' \vdash_{wf} \Delta$  using wf-weakening2(6) check-branch-s-branchI by metis
    show  $\vdash_{wf} \Theta$  using check-branch-s-branchI by auto
    show  $\Theta ; \mathcal{B} ; \Gamma' \vdash_{wf} \tau$  using check-branch-s-branchI wfT-weakening (wfG \Theta \mathcal{B} \Gamma') by presburger

    show  $\Theta ; \{\}\} ; GNil \vdash_{wf} const$  using check-branch-s-branchI by auto
    show atom  $x \# (\Theta, \Phi, \mathcal{B}, \Gamma', \Delta, tid, cons, const, v, \tau)$  using check-branch-s-branchI by auto
    have toSet  $((x, b\text{-of } const, CE\text{-val } v == CE\text{-val } (V\text{-cons } tid \ cons \ (V\text{-var } x)) \ \text{AND } c\text{-of } const \ x)$ 
 $\#_{\Gamma} \Gamma) \subseteq \text{toSet } ((x, b\text{-of } const, CE\text{-val } v == CE\text{-val } (V\text{-cons } tid \ cons \ (V\text{-var } x)) \ \text{AND } c\text{-of } const \ x)$ 
 $\#_{\Gamma} \Gamma')$ 
    using check-branch-s-branchI by auto
    moreover hence  $\Theta ; \mathcal{B} \vdash_{wf} ((x, b\text{-of } const, CE\text{-val } v == CE\text{-val } (V\text{-cons } tid \ cons \ (V\text{-var } x))$ 
AND  $c\text{-of } const \ x) \#_{\Gamma} \Gamma')$ 
    using check-branch-s-branchI wfG-cons-weakening check-s-wf by metis
    ultimately show  $\Theta ; \Phi ; \mathcal{B} ; (x, b\text{-of } const, CE\text{-val } v == CE\text{-val } (V\text{-cons } tid \ cons \ (V\text{-var } x))$ 
AND  $c\text{-of } const \ x) \#_{\Gamma} \Gamma' ; \Delta \vdash s \Leftarrow \tau$ 
    using check-branch-s-branchI using fresh-dom-free by auto
  qed
next
case (check-ifI  $z \ \Theta \ \Phi \ \mathcal{B} \ \Gamma \ \Delta \ v \ s1 \ s2 \ \tau$ )
  show ?case proof
    show  $\langle atom \ z \# (\Theta, \Phi, \mathcal{B}, \Gamma', \Delta, v, s1, s2, \tau) \rangle$  using fresh-prodN check-ifI by auto
    show  $\langle \Theta ; \mathcal{B} ; \Gamma' \vdash v \Leftarrow \llbracket z : B\text{-bool} \mid TRUE \rrbracket \rangle$  using check-v-g-weakening check-ifI by auto
    show  $\langle \Theta ; \Phi ; \mathcal{B} ; \Gamma' ; \Delta \vdash s1 \Leftarrow \llbracket z : b\text{-of } \tau \mid CE\text{-val } v == CE\text{-val } (V\text{-lit } L\text{-true}) \text{ IMP } c\text{-of } \tau \text{ } z \rrbracket \rangle$  using check-ifI by auto
    show  $\langle \Theta ; \Phi ; \mathcal{B} ; \Gamma' ; \Delta \vdash s2 \Leftarrow \llbracket z : b\text{-of } \tau \mid CE\text{-val } v == CE\text{-val } (V\text{-lit } L\text{-false}) \text{ IMP } c\text{-of } \tau \text{ } z \rrbracket \rangle$  using check-ifI by auto
  qed
next
case (check-whileI  $\Delta \ G \ P \ s1 \ z \ s2 \ \tau'$ )
  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  $\Delta \ G \ P \ s1 \ z \ s2 \ \tau$ )
  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  $\Theta$   $\Phi$   $\mathcal{B}$   $\Gamma$   $\Delta$   $\tau'$  v  $\tau$  s)
    thus ?case using check-v-g-weakening check-s-check-branch-s-check-branch-list.intros by auto
next
    case (check-assignI  $\Theta$   $\Phi$   $\mathcal{B}$   $\Gamma$   $\Delta$  u  $\tau$  v z  $\tau'$ )
    show ?case proof
      show  $\langle \Theta \vdash_{wf} \Phi \rangle$  using check-assignI by auto
      show  $\langle \Theta; \mathcal{B}; \Gamma' \vdash_{wf} \Delta \rangle$  using check-assignI wf-weakening by auto
      show  $\langle (u, \tau) \in \text{setD } \Delta \rangle$  using check-assignI by auto
      show  $\langle \Theta; \mathcal{B}; \Gamma' \vdash v \Leftarrow \tau \rangle$  using check-assignI check-v-g-weakening by auto
      show  $\langle \Theta; \mathcal{B}; \Gamma' \vdash \llbracket z : B\text{-unit} \mid \text{TRUE} \rrbracket \lesssim \tau' \rangle$  using subtype-weakening check-assignI by auto
    qed
next
    case (check-caseI  $\Delta$   $\Gamma$   $\Theta$  dclist cs  $\tau$  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  $\Theta$   $\Phi$   $\mathcal{B}$   $\Gamma$   $\Delta$  c  $\tau$  s)
    show ?case proof
      show  $\langle \text{atom } x \# (\Theta, \Phi, \mathcal{B}, \Gamma', \Delta, c, \tau, s) \rangle$  using check-assertI by auto

      have  $\Theta; \mathcal{B} \vdash_{wf} (x, B\text{-bool}, c) \#_{\Gamma} \Gamma$  using check-assertI check-s-wf by metis
      hence *:  $\Theta; \mathcal{B} \vdash_{wf} (x, B\text{-bool}, c) \#_{\Gamma} \Gamma'$  using wfG-cons-weakening check-assertI by metis
      moreover have  $\text{toSet } ((x, B\text{-bool}, c) \#_{\Gamma} \Gamma) \subseteq \text{toSet } ((x, B\text{-bool}, c) \#_{\Gamma} \Gamma')$  using check-assertI by
    auto
      thus  $\langle \Theta; \Phi; \mathcal{B}; (x, B\text{-bool}, c) \#_{\Gamma} \Gamma'; \Delta \vdash s \Leftarrow \tau \rangle$  using check-assertI(11) [OF - *] by auto

      show  $\langle \Theta; \mathcal{B}; \Gamma' \models c \rangle$  using check-assertI valid-weakening by metis
      show  $\langle \Theta; \mathcal{B}; \Gamma' \vdash_{wf} \Delta \rangle$  using check-assertI wf-weakening by metis
    qed
  qed

lemma wfG-xa-fresh-in-v:
  fixes c::c and  $\Gamma::\Gamma$  and  $G::\Gamma$  and v::v and xa::x
  assumes  $\Theta; \mathcal{B}; \Gamma \vdash v \Rightarrow \tau$  and  $G = (\Gamma' @ (x, b, c[z ::= V\text{-var } x]_v) \#_{\Gamma} \Gamma)$  and  $\text{atom } xa \# G$  and  $\Theta; \mathcal{B} \vdash_{wf} G$ 
  shows  $\text{atom } xa \# v$ 
proof -
  have  $\Theta; \mathcal{B}; \Gamma \vdash_{wf} v : b\text{-of } \tau$  using infer-v-wf assms by metis
  hence  $\text{supp } v \subseteq \text{atom-dom } \Gamma \cup \text{supp } \mathcal{B}$  using wfV-supp by simp
  moreover have  $\text{atom } xa \notin \text{atom-dom } G$ 
    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  $G::\Gamma$ 

```

assumes $\text{atom } z' \# (x, v)$ **and** $\langle \text{atom } z' \# G \rangle$
shows $\text{atom } z' \# G[x ::= v]_{\Gamma v}$
proof –
have $\text{atom } z' \# v$ **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** $\Gamma::\Gamma$ **and** $G::\Gamma$ **and** $xa::x$
assumes $\Theta; \mathcal{B}; \Gamma \vdash v \Rightarrow \tau$ **and** $G = (\Gamma' @ (x, b, c[z ::= V\text{-var } x]_v) \#_{\Gamma} \Gamma)$ **and** $\text{atom } xa \# G$ **and** Θ ;
 $\mathcal{B} \vdash_{wf} G$
shows $\text{atom } xa \# (\text{subst-gv } G \ x \ v)$
proof –
have $\text{atom } xa \# v$ **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** $cs::\text{branch-s}$ **and** $css::\text{branch-list}$
shows $\Theta; \Phi; \mathcal{B}; \Gamma; \Delta \vdash s \Leftarrow \tau \Longrightarrow \text{setD } \Delta \subseteq \text{setD } \Delta' \Longrightarrow \text{wfD } \Theta \ \mathcal{B} \ \Gamma \ \Delta' \Longrightarrow \Theta; \Phi; \mathcal{B}; \Gamma; \Delta' \vdash s \Leftarrow \tau$ **and**
 $\text{check-branch-s } \Theta \ \Phi \ \mathcal{B} \ \Gamma \ \Delta \ \text{tid cons const } v \ cs \ \tau \Longrightarrow \text{setD } \Delta \subseteq \text{setD } \Delta' \Longrightarrow \text{wfD } \Theta \ \mathcal{B} \ \Gamma \ \Delta' \Longrightarrow \text{check-branch-s } \Theta \ \Phi \ \mathcal{B} \ \Gamma \ \Delta' \ \text{tid cons const } v \ cs \ \tau$ **and**
 $\text{check-branch-list } \Theta \ \Phi \ \mathcal{B} \ \Gamma \ \Delta \ \text{tid dclist } v \ css \ \tau \Longrightarrow \text{setD } \Delta \subseteq \text{setD } \Delta' \Longrightarrow \text{wfD } \Theta \ \mathcal{B} \ \Gamma \ \Delta' \Longrightarrow \text{check-branch-list } \Theta \ \Phi \ \mathcal{B} \ \Gamma \ \Delta' \ \text{tid dclist } v \ css \ \tau$
proof(*nominal-induct* τ **and** τ **and** τ *avoiding: Δ' arbitrary: v rule: *check-s-check-branch-s-check-branch-list.strong-ind**)
case (*check-valI* $\Theta \ \mathcal{B} \ \Gamma \ \Delta \ \Phi \ v \ \tau' \ \tau$)
then show *?case* **using** *check-s-check-branch-s-check-branch-list.intros* **by** *blast*
next
case (*check-letI* $x \ \Theta \ \Phi \ \mathcal{B} \ \Gamma \ \Delta \ e \ \tau \ z \ s \ b \ c$)
show *?case* **proof**
show $\text{atom } x \# (\Theta, \Phi, \mathcal{B}, \Gamma, \Delta', e, \tau)$ **using** *check-letI* **by** *auto*
show $\text{atom } z \# (x, \Theta, \Phi, \mathcal{B}, \Gamma, \Delta', e, \tau, s)$ **using** *check-letI* **by** *auto*
show $\Theta; \Phi; \mathcal{B}; \Gamma; \Delta' \vdash e \Rightarrow \llbracket z : b \mid c \rrbracket$ **using** *check-letI infer-e-d-weakening* **by** *auto*
have $\Theta; \mathcal{B} \vdash_{wf} (x, b, c[z ::= V\text{-var } x]_v) \#_{\Gamma} \Gamma$ **using** *check-letI check-s-wf* **by** *metis*
moreover have $\Theta; \mathcal{B}; \Gamma \vdash_{wf} \Delta'$ **using** *check-letI check-s-wf* **by** *metis*
ultimately have $\Theta; \mathcal{B}; (x, b, c[z ::= V\text{-var } x]_v) \#_{\Gamma} \Gamma \vdash_{wf} \Delta'$ **using** *wf-weakening2(6)* *toSet.simps*
by *fast*
thus $\Theta; \Phi; \mathcal{B}; (x, b, c[z ::= V\text{-var } x]_v) \#_{\Gamma} \Gamma; \Delta' \vdash s \Leftarrow \tau$ **using** *check-letI* **by** *simp*
qed
next
case (*check-branch-s-branchI* $\Theta \ \mathcal{B} \ \Gamma \ \Delta \ \tau \ \text{const } x \ \Phi \ \text{tid cons } v \ s$)
moreover have $\Theta; \mathcal{B} \vdash_{wf} (x, b\text{-of const, CE-val } v == \text{CE-val } (V\text{-cons tid cons } (V\text{-var } x)))$ **AND**
 $c\text{-of const } x) \#_{\Gamma} \Gamma$
using *check-s-wf[OF check-branch-s-branchI(16)]* **by** *metis*
moreover hence $\Theta; \mathcal{B}; (x, b\text{-of const, CE-val } v == \text{CE-val } (V\text{-cons tid cons } (V\text{-var } x)))$ **AND**
 $c\text{-of const } x) \#_{\Gamma} \Gamma \vdash_{wf} \Delta'$
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  $\Theta \Phi \mathcal{B} \Gamma \Delta tid dclist v cs \tau css$ )
then show ?case using check-s-check-branch-s-check-branch-list.intros by meson
next
case (check-branch-list-finalI  $\Theta \Phi \mathcal{B} \Gamma \Delta tid dclist v cs \tau$ )
then show ?case using check-s-check-branch-s-check-branch-list.intros by meson
next
case (check-ifI  $z \Theta \Phi \mathcal{B} \Gamma \Delta v s1 s2 \tau$ )
show ?case proof
  show  $\langle atom\ z \# (\Theta, \Phi, \mathcal{B}, \Gamma, \Delta', v, s1, s2, \tau) \rangle$  using fresh-prodN check-ifI by auto
  show  $\langle \Theta; \mathcal{B}; \Gamma \vdash v \Leftarrow \llbracket z : B\text{-}bool \mid TRUE \rrbracket \rangle$  using check-ifI by auto
  show  $\langle \Theta; \Phi; \mathcal{B}; \Gamma; \Delta' \vdash s1 \Leftarrow \llbracket z : b\text{-}of\ \tau \mid CE\text{-}val\ v == CE\text{-}val\ (V\text{-}lit\ L\text{-}true) \text{ IMP } c\text{-}of\ \tau\ z \rrbracket \rangle$  using check-ifI by auto
  show  $\langle \Theta; \Phi; \mathcal{B}; \Gamma; \Delta' \vdash s2 \Leftarrow \llbracket z : b\text{-}of\ \tau \mid CE\text{-}val\ v == CE\text{-}val\ (V\text{-}lit\ L\text{-}false) \text{ IMP } c\text{-}of\ \tau\ z \rrbracket \rangle$  using check-ifI by auto
qed
next
case (check-assertI  $x \Theta \Phi \mathcal{B} \Gamma \Delta c \tau s$ )
show ?case proof
  show  $atom\ x \# (\Theta, \Phi, \mathcal{B}, \Gamma, \Delta', c, \tau, s)$  using fresh-prodN check-assertI by auto
  show  $*: \Theta; \mathcal{B}; \Gamma \vdash_{wf} \Delta'$  using check-assertI by auto
  hence  $\Theta; \mathcal{B}; (x, B\text{-}bool, c) \#_{\Gamma} \Gamma \vdash_{wf} \Delta'$  using wf-weakening2(6)[OF *, of  $(x, B\text{-}bool, c) \#_{\Gamma} \Gamma$ ]
check-assertI check-s-wf toSet.simps by auto
  thus  $\Theta; \Phi; \mathcal{B}; (x, B\text{-}bool, c) \#_{\Gamma} \Gamma; \Delta' \vdash s \Leftarrow \tau$ 
  using check-assertI(11)[OF  $\langle setD\ \Delta \subseteq setD\ \Delta' \rangle$ ] by simp

  show  $\Theta; \mathcal{B}; \Gamma \models c$  using fresh-prodN check-assertI by auto

qed
next
case (check-let2I  $x \Theta \Phi \mathcal{B} G \Delta t s1 \tau s2$ )
show ?case proof
  show  $atom\ x \# (\Theta, \Phi, \mathcal{B}, G, \Delta', t, s1, \tau)$  using check-let2I by auto

  show  $\Theta; \Phi; \mathcal{B}; G; \Delta' \vdash s1 \Leftarrow t$  using check-let2I infer-e-d-weakening by auto
  have  $\Theta; \mathcal{B}; (x, b\text{-}of\ t, c\text{-}of\ t\ x) \#_{\Gamma} G \vdash_{wf} \Delta'$  using check-let2I wf-weakening2(6) check-s-wf by
fastforce
  thus  $\Theta; \Phi; \mathcal{B}; (x, b\text{-}of\ t, c\text{-}of\ t\ x) \#_{\Gamma} G; \Delta' \vdash s2 \Leftarrow \tau$  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  $atom\ u \# (\Theta, \Phi, \mathcal{B}, \Gamma, \Delta', \tau', v, \tau)$  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  $u \notin fst\ 'setD\ \Delta'$  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  $(u, \tau) \in \text{setD } \Delta'$  by auto
ultimately show ?case using Typing.check-assignI by simp
next
case (check-whileI  $\Theta \Phi \mathcal{B} \Gamma \Delta s1 z s2 \tau'$ )
then show ?case using check-s-check-branch-s-check-branch-list.intros by meson
next
case (check-seqI  $\Theta \Phi \mathcal{B} \Gamma \Delta s1 z s2 \tau$ )
then show ?case using check-s-check-branch-s-check-branch-list.intros by meson
next
case (check-caseI  $\Theta \Phi \mathcal{B} \Gamma \Delta tid \text{dclist } v \text{cs } \tau z$ )
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  $wfV \Theta \mathcal{B} GNil v b$  and  $wfCE \Theta \mathcal{B} ((x, b, TRUE) \#_{\Gamma} GNil)$ 
   $ce2 b'$  and  $wfCE \Theta \mathcal{B} GNil ce1 b'$ 
  shows  $\langle \Theta; \mathcal{B}; (x, b, ([x]^v)^{ce} == [v]^{ce}) \rangle \#_{\Gamma} GNil \models ce1 == ce2 \rangle$ 
  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  $wf: \langle \Theta; \mathcal{B}; (x, b, ([x]^v)^{ce} == [v]^{ce}) \rangle \#_{\Gamma} GNil \vdash_{wf} ce1 == ce2 \rangle$ 
    apply (rule wfC-eqI[where  $b=b'$ ])
    using wfg toSet.simps assms wfCE-weakening apply simp

    using wfg assms wf-replace-inside1(8) assms
    using wfC-trueI wf-trans(8) by auto

  show  $\forall i. ((\Theta; (x, b, ([x]^v)^{ce} == [v]^{ce}) \#_{\Gamma} GNil \vdash i) \wedge (i \models (x, b, ([x]^v)^{ce} == [v]^{ce}) \#_{\Gamma} GNil)) \longrightarrow (i \models (ce1 == ce2)))$  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  $1: wfV \Theta \mathcal{B} ((x, b, ([x]^v)^{ce} == [v]^{ce}) \#_{\Gamma} GNil) v b$ 
      using wf-weakening assms append-g.simps toSet.simps wf wfX-wfY
      by (metis empty-subsetI)
    hence  $\exists s. i \Vdash v \sim s$  using eval-v-exist[OF 1] as by auto
    then obtain s where  $iv: i \Vdash v \sim s ..$ 

    hence  $ix: i x = \text{Some } s$  proof -
      have  $i \models ([x]^v)^{ce} == [v]^{ce}$  using is-satis-g.simps as by auto
      hence  $i \Vdash ([x]^v)^{ce} == [v]^{ce}$  using is-satis.simps by auto
      hence  $i \Vdash ([x]^v)^{ce} \sim s$  using

```

iv *eval-e-elim*s
by (*metis eval-c-elim*s(7) *eval-e-uniqueness eval-e-valI*)
thus *?thesis* **using** *eval-v-elim*s(2) *eval-e-elim*s(1) **by** *metis*
qed

have $1: wfCE \ \Theta \ \mathcal{B} \ ((x, b, [[x]^v]^{ce} == [v]^{ce}) \#_{\Gamma} GNil) \ ce1 \ b'$
using *wfCE-weakening* *assms append-g.simps toSet.simps wf wfX-wfY*
by (*metis empty-subsetI*)
hence $\exists s1. i \llbracket ce1 \rrbracket \sim s1$ **using** *eval-e-exist* *assms* **as** **by** *auto*
then obtain $s1$ **where** $s1: i \llbracket ce1 \rrbracket \sim s1$..

moreover have $i \llbracket ce2 \rrbracket \sim s1$ **proof** –
have $i \llbracket ce2[x::=v]_{cev} \rrbracket \sim s1$ **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 $i(x \mapsto s) \llbracket ce2 \rrbracket \sim s1$
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 $i \llbracket ce1 == ce2 \rrbracket \sim True$ **using** *eval-c-eqI* **by** *metis*
qed
qed

lemma *check-v-top*:
fixes $v::v$
assumes $\Theta; \mathcal{B}; GNil \vdash v \Leftarrow \tau$ **and** $ce1 = ce2[z::=v]_{cev}$ **and** $\Theta; \mathcal{B}; GNil \vdash_{wf} \{ z : b\text{-of } \tau \mid ce1 == ce2 \}$
and $supp \ ce1 \subseteq supp \ \mathcal{B}$
shows $\Theta; \mathcal{B}; GNil \vdash v \Leftarrow \{ z : b\text{-of } \tau \mid ce1 == ce2 \}$
proof –
obtain t **where** $t: \Theta; \mathcal{B}; GNil \vdash v \Rightarrow t \wedge \Theta; \mathcal{B}; GNil \vdash t \lesssim \tau$
using *assms check-v-elim*s **by** *metis*

then obtain z' **and** b' **where** $*:t = \{ z' : b' \mid [[z']^v]^{ce} == [v]^{ce} \} \wedge atom \ z' \# v \wedge atom \ z' \#$
 $(\Theta, \mathcal{B}, GNil)$
using *assms infer-v-form* **by** *metis*
have $beq: b\text{-of } t = b\text{-of } \tau$ **using** *subtype-eq-base2 b-of.simps t* **by** *auto*
obtain $x::x$ **where** $xf: \langle atom \ x \# (\Theta, \mathcal{B}, GNil, z', [[z']^v]^{ce} == [v]^{ce}, z, ce1 == ce2) \rangle$
using *obtain-fresh* **by** *metis*

have $\Theta; \mathcal{B}; (x, b\text{-of } \tau, TRUE) \#_{\Gamma} GNil \vdash_{wf} (ce1[z::=[x]^v]_v == ce2[z::=[x]^v]_v)$
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** $b2: \Theta; \mathcal{B}; (x, b\text{-of } t, TRUE) \#_{\Gamma} GNil \vdash_{wf} ce1[z::=[x]^v]_v : b2 \wedge$
 $\Theta; \mathcal{B}; (x, b\text{-of } t, TRUE) \#_{\Gamma} GNil \vdash_{wf} ce2[z::=[x]^v]_v : b2$ **using** *wfC-elim*s(3)
beq **by** *metis*

from xf **have** $\Theta; \mathcal{B}; GNil \vdash \{ z' : b\text{-of } t \mid [[z']^v]^{ce} == [v]^{ce} \} \lesssim \{ z : b\text{-of } t \mid ce1 == ce2 \}$
proof
show $\langle \Theta; \mathcal{B}; GNil \vdash_{wf} \{ z' : b\text{-of } t \mid [[z']^v]^{ce} == [v]^{ce} \} \rangle$ **using** *b-of.simps* *assms infer-v-wf*


```

t * by auto
show ⟨ Θ; B; GNil ⊢wf { z : b-of t | ce1 == ce2 } ⟩ using beq assms by auto
have ⟨ Θ; B; (x, b-of t, ([ [ x ]v]ce == [ v ]ce)) #Γ GNil ⊨ (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 # 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 # 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; B; GNil ⊢wf v : b-of t ⟩ using infer-v-wf t by simp
show ⟨ Θ; B; (x, b-of t, TRUE) #Γ GNil ⊢wf ce2[z::=[ x ]v]v : b2 ⟩ using b2 by auto

have Θ; B; (x, b-of t, TRUE) #Γ GNil ⊢wf ce1[z::=[ x ]v]v : b2 using b2 by auto
moreover have atom x # ce1[z::=[ x ]v]v
  using fresh-subst-v-if assms fresh-def
  using ⟨ Θ; B; GNil ⊢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; B; GNil ⊢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; B; (x, b-of t, ([ [ z' ]v]ce == [ v ]ce)[z'::=[ x ]v]v) #Γ GNil ⊨ (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

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. Sometimes 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* :: $\Gamma \Rightarrow (x*c) \text{ list} \Rightarrow \Gamma$ **where**
replace-in-g-many G xcs = *List.foldr* ($\lambda(x,c) \ G. \ G[x \mapsto c]$) 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* :: $\Theta \Rightarrow \mathcal{B} \Rightarrow \Gamma \Rightarrow (x*c) \text{ list} \Rightarrow \Gamma \Rightarrow \text{bool}$ (- ; - \vdash - \langle - $\rangle \rightsquigarrow$ - [100,50,50] 50) **where**

replace-in-g-subtyped-nilI: $\Theta; \mathcal{B} \vdash G \langle \ [] \rangle \rightsquigarrow G$
| *replace-in-g-subtyped-consI*: \llbracket
Some (b,c') = *lookup* G x ;
 $\Theta; \mathcal{B}; G \vdash_{wf} c$;
 $\Theta; \mathcal{B}; G[x \mapsto c] \models c'$;
 $\Theta; \mathcal{B} \vdash G[x \mapsto c] \langle \ xcs \rangle \rightsquigarrow G'; x \notin \text{fst ' set } xcs \rrbracket \implies$
 $\Theta; \mathcal{B} \vdash G \langle \ (x,c)\#xcs \rangle \rightsquigarrow G'$

equivariance *replace-in-g-subtyped*

nominal-inductive *replace-in-g-subtyped* .

inductive-cases *replace-in-g-subtyped-elim*[*elim!*]:

$\Theta; \mathcal{B} \vdash G \langle \ [] \rangle \rightsquigarrow G'$
 $\Theta; \mathcal{B} \vdash ((x,b,c)\#_{\Gamma} G) \langle \ acs \rangle \rightsquigarrow ((x,b,c)\#_{\Gamma} G')$
 $\Theta; \mathcal{B} \vdash G' \langle \ (x,c)\# acs \rangle \rightsquigarrow G$

lemma *rigs-atom-dom-eq*:

assumes $\Theta; \mathcal{B} \vdash G \langle \ xcs \rangle \rightsquigarrow G'$

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  $\Theta$  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  $\Theta$ ; B  $\vdash$  G  $\langle$  xcs  $\rangle \rightsquigarrow$  G' and wfG  $\Theta$  B G
  shows wfG  $\Theta$  B G'
  using assms proof(induct rule: replace-in-g-subtyped.induct)
  case (replace-in-g-subtyped-nilI  $\Theta$  G)
  then show ?case by auto
next
  case (replace-in-g-subtyped-consI b c' G x  $\Theta$  c xcs G')
  then show ?case using valid-g-wf by auto
qed

```

```

lemma wfD-rig-single:
  fixes  $\Delta::\Delta$  and  $x::x$  and  $c::c$  and  $G::\Gamma$ 
  assumes  $\Theta$ ; B; G  $\vdash_{wf}$   $\Delta$  and wfG  $\Theta$  B (G[x $\mapsto$ c])
  shows  $\Theta$ ; B; G[x $\mapsto$ c]  $\vdash_{wf}$   $\Delta$ 
proof(cases atom x  $\in$  atom-dom G)
  case False
  hence (G[x $\mapsto$ 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 $\mapsto$ c]) = G1@ $(x,b,c)$ # $_{\Gamma}$ G2 using replace-in-g-inside wfD-wf assms wfD-wf by metis

  hence wfG  $\Theta$  B ((x,b,c)# $_{\Gamma}$ G2) using wfG-suffix assms by auto
  hence  $\Theta$ ; B; (x, b, TRUE) # $_{\Gamma}$  G2  $\vdash_{wf}$  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  $\Theta$ ; B  $\vdash$  G  $\langle$  xcs  $\rangle \rightsquigarrow$  G' and wfD  $\Theta$  B G  $\Delta$ 
  shows wfD  $\Theta$  B G'  $\Delta$ 
using assms proof(induct rule: replace-in-g-subtyped.induct)
  case (replace-in-g-subtyped-nilI  $\Theta$  G)
  then show ?case by auto
next
  case (replace-in-g-subtyped-consI b c' G x  $\Theta$  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 $\Theta; \mathcal{B} \vdash \Gamma \langle xcs \rangle \rightsquigarrow \Gamma'$ and $wfG \Theta \mathcal{B} \Gamma$ and $wfG \Theta \mathcal{B} \Gamma'$ and $atom\ x \# \Gamma$
 shows $atom\ x \# \Gamma'$
 using *wfG-dom-supp* *assms* *fresh-def* *rigs-atom-dom-eq* by *metis*

lemma *replace-in-g-fresh1*:

fixes $x::x$

assumes $\Theta; \mathcal{B} \vdash \Gamma \langle xcs \rangle \rightsquigarrow \Gamma'$ and $wfG \Theta \mathcal{B} \Gamma$ and $atom\ x \# \Gamma$

shows $atom\ x \# \Gamma'$

proof –

have $wfG \Theta \mathcal{B} \Gamma'$ **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 $wsX::\Gamma \Rightarrow (x*c)\ list \Rightarrow bool$ **where**

wsX-NilI: $wsX\ G\ []$

| *wsX-ConsI*: $\llbracket wsX\ G\ xcs ; atom\ x \in atom-dom\ G ; x \notin fst\ 'set\ xcs \rrbracket \Longrightarrow wsX\ G\ ((x,c)\#xcs)$

equivariance wsX

nominal-inductive wsX .

lemma *wsX-if1*:

assumes $wsX\ G\ xcs$

shows $((atom\ 'fst\ 'set\ xcs) \subseteq atom-dom\ G) \wedge List.distinct\ (List.map\ fst\ xcs)$

using *assms* **by**(*induct* *rule*: $wsX.induct$,*force*+)

lemma *wsX-if2*:

assumes $((atom\ 'fst\ 'set\ xcs) \subseteq atom-dom\ G) \wedge List.distinct\ (List.map\ fst\ xcs)$

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 $atom\ 'fst\ 'set\ xcs \subseteq atom-dom\ G$ **using** *Cons* **by** *simp*

ultimately show *?thesis* **using** *Cons* **by** *fast*

qed

moreover have $atom\ x \in atom-dom\ G$ **using** *Cons* *xc*

by *simp*

moreover have $x \notin fst\ 'set\ xcs$ **using** *Cons* *xc*

by *simp*

ultimately show *?case* **using** *wsX-ConsI* *xc* **by** *blast*

qed

lemma *wsX-iff*:

$wsX\ G\ xcs = (((atom\ 'fst\ 'set\ xcs) \subseteq atom-dom\ G) \wedge List.distinct\ (List.map\ fst\ xcs))$

using *wsX-if1* *wsX-if2* **by** *meson*

inductive-cases $wsX-elim$ [*elim*!]:

$wsX\ G\ []$

$wsX\ G\ ((x,c)\#xcs)$

lemma *wsX-cons:*

assumes $wsX\ \Gamma\ xcs$ **and** $x \notin fst\ 'set\ xcs$
shows $wsX\ ((x, b, c1) \#_{\Gamma} \Gamma)\ ((x, c2) \# xcs)$

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 $atom\ 'fst\ 'set\ xcs \subseteq atom-dom\ (xbc \#_{\Gamma} \Gamma) \wedge distinct\ (map\ fst\ xcs)$

using *GCons.prem1* *wsX-iff* **by** *blast*

then have $wsX\ ((x, b, c1) \#_{\Gamma} xbc \#_{\Gamma} \Gamma)\ xcs$

by (*simp add: Un-commute subset-Un-eq wsX-if2*)

then show *?case* **by** (*simp add: GCons.prem2 wsX-ConsI*)

qed

lemma *wsX-cons2:*

assumes $wsX\ \Gamma\ xcs$ **and** $x \notin fst\ 'set\ xcs$

shows $wsX\ ((x, b, c1) \#_{\Gamma} \Gamma)\ xcs$

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 $atom\ 'fst\ 'set\ xcs \subseteq atom-dom\ (xbc \#_{\Gamma} \Gamma) \wedge distinct\ (map\ fst\ xcs)$

using *GCons.prem1* *wsX-iff* **by** *blast* **then show** *?case* **by** (*simp add: Un-commute subset-Un-eq wsX-if2*)

qed

lemma *wsX-cons3:*

assumes $wsX\ \Gamma\ xcs$

shows $wsX\ ((x, b, c1) \#_{\Gamma} \Gamma)\ xcs$

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 $atom\ 'fst\ 'set\ xcs \subseteq atom-dom\ (xbc \#_{\Gamma} \Gamma) \wedge distinct\ (map\ fst\ xcs)$

using *GCons.prem1* *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** $atom\ x \not\# G$ **and** $wfG\ \Theta\ \mathcal{B}\ G$

shows $x \notin fst\ 'set\ xcs$

proof –

have $atom\ x \notin atom-dom\ G$ **using** *assms*

using *fresh-def wfG-dom-suppl* **by** *auto*

thus ?thesis using wsX-iff assms by blast
qed

lemma replace-in-g-dist:

assumes $x' \neq x$

shows $\text{replace-in-g } ((x, b, c) \#_{\Gamma} G) x' c'' = ((x, b, c) \#_{\Gamma} (\text{replace-in-g } G x' c'))$ using replace-in-g.simps
assms by presburger

lemma wfG-replace-inside-rig:

fixes $c'' :: c$

assumes $\langle \Theta; \mathcal{B} \vdash_{wf} G[x' \mapsto c'] \rangle \langle \Theta; \mathcal{B} \vdash_{wf} (x, b, c) \#_{\Gamma} G \rangle$

shows $\Theta; \mathcal{B} \vdash_{wf} (x, b, c) \#_{\Gamma} G[x' \mapsto c']$

proof(rule wfG-consI)

have $wfG \ \Theta \ \mathcal{B} \ G$ using wfG-cons assms by auto

show $*: \Theta; \mathcal{B} \vdash_{wf} G[x' \mapsto c']$ using assms by auto

show $\text{atom } x \nmid G[x' \mapsto c']$ using replace-in-g-fresh-single[OF *] assms wfG-elim2 assms by metis

show $*: \Theta; \mathcal{B} \vdash_{wf} b$ using wfG-elim2 assms by auto

show $\Theta; \mathcal{B}; (x, b, \text{TRUE}) \#_{\Gamma} G[x' \mapsto c'] \vdash_{wf} c$

proof(cases $\text{atom } x' \notin \text{atom-dom } G$)

case True

hence $G = G[x' \mapsto c']$ using replace-in-g-forget $\langle wfG \ \Theta \ \mathcal{B} \ G \rangle$ 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 $***: (G[x' \mapsto c']) = G1 @ (x', b', c'') \#_{\Gamma} G2$

using replace-in-g-inside $\langle wfG \ \Theta \ \mathcal{B} \ G \rangle$ by metis

hence $\Theta; \mathcal{B}; (x, b, \text{TRUE}) \#_{\Gamma} G1 @ (x', b', c') \#_{\Gamma} G2 \vdash_{wf} c$ using * ** assms wfG-wfC by auto

hence $\Theta; \mathcal{B}; (x, b, \text{TRUE}) \#_{\Gamma} G1 @ (x', b', c'') \#_{\Gamma} G2 \vdash_{wf} c$ 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 $\Theta; \mathcal{B}; \Gamma[x' \mapsto c'] \models c'$ and $x' \neq x$ and $\Theta; \mathcal{B} \vdash_{wf} (x, b, c) \#_{\Gamma} \Gamma[x' \mapsto c']$

shows $\Theta; \mathcal{B}; ((x, b, c) \#_{\Gamma} \Gamma)[x' \mapsto c'] \models c'$

apply(subst replace-in-g-dist, simp add: assms, rule valid-weakening)

using assms by auto+

lemma replace-in-g-subtyped-cons:

assumes $\text{replace-in-g-subtyped } \Theta \ \mathcal{B} \ G \ \text{xs } G'$ and $wfG \ \Theta \ \mathcal{B} \ ((x, b, c) \#_{\Gamma} G)$

shows $x \notin \text{fst ' set xs} \implies \text{replace-in-g-subtyped } \Theta \ \mathcal{B} \ ((x, b, c) \#_{\Gamma} G) \ \text{xs } ((x, b, c) \#_{\Gamma} G')$

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 $b' \ c' \ G \ x' \ \Theta \ \mathcal{B} \ c'' \ \text{xs}' \ G'$)

hence $\Theta; \mathcal{B} \vdash_{wf} G[x' \mapsto c'']$ using *valid.simps wfC-wf* by *auto*

show *?case* **proof**(*rule* *replace-in-g-subtyped.replace-in-g-subtyped-consI*)
show *Some* $(b', c') = \text{lookup } ((x, b, c) \#_{\Gamma} G) x'$ using *lookup.simps*
fst-conv image-iff Γ -set-intros surj-pair *replace-in-g-subtyped-consI* by *force*
show *wbc*: $\Theta; \mathcal{B}; (x, b, c) \#_{\Gamma} G \vdash_{wf} c''$ using *wf-weakening* $\langle \Theta; \mathcal{B}; G \vdash_{wf} c'' \rangle \langle \Theta; \mathcal{B} \vdash_{wf} (x, b, c) \#_{\Gamma} G \rangle$ by *fastforce*
have $x' \neq x$ using *replace-in-g-subtyped-consI* by *auto*
have *wbc1*: $\Theta; \mathcal{B} \vdash_{wf} (x, b, c) \#_{\Gamma} G[x' \mapsto c'']$ **proof** –
have $(x, b, c) \#_{\Gamma} G[x' \mapsto c''] = ((x, b, c) \#_{\Gamma} G)[x' \mapsto c'']$ using $\langle x' \neq x \rangle$ using *replace-in-g.simps*
by *auto*
thus *?thesis* using *wfG-replace-inside-rig* $\langle \Theta; \mathcal{B} \vdash_{wf} G[x' \mapsto c''] \rangle \langle \Theta; \mathcal{B} \vdash_{wf} (x, b, c) \#_{\Gamma} G \rangle$
by *fastforce*
qed
show $*$: $\Theta; \mathcal{B}; \text{replace-in-g } ((x, b, c) \#_{\Gamma} G) x' c'' \models c'$
proof –
have $\Theta; \mathcal{B}; G[x' \mapsto c''] \models c'$ using *replace-in-g-subtyped-consI* by *auto*
thus *?thesis* using *replace-in-g-valid-weakening wbc1* $\langle x' \neq x \rangle$ by *auto*
qed

show *replace-in-g-subtyped* $\Theta \mathcal{B} (\text{replace-in-g } ((x, b, c) \#_{\Gamma} G) x' c'') \text{ xcs}' ((x, b, c) \#_{\Gamma} G')$
using *replace-in-g-subtyped-consI wbc1* by *auto*
show $x' \notin \text{fst 'set xcs'}$
using *replace-in-g-subtyped-consI* by *linarith*
qed
qed

lemma *replace-in-g-split*:

fixes $G::\Gamma$
assumes $\Gamma = \text{replace-in-g } \Gamma' x c$ and $\Gamma' = G'@(x, b, c') \#_{\Gamma} G$ and $wfG \Theta \mathcal{B} \Gamma'$
shows $\Gamma = G'@(x, b, c') \#_{\Gamma} G$
using *assms* **proof**(*induct* G' *arbitrary*: $G \Gamma \Gamma'$ *rule*: Γ -*induct*)
case *GNil*
then show *?case* by *simp*
next
case (*GCons* $x1 b1 c1 \Gamma1$)
hence $x1 \neq x$
using *wfG-cons-fresh2*[*of* $\Theta \mathcal{B} x1 b1 c1 \Gamma1 x b$]
using *GCons.prem*s(2) *GCons.prem*s(3) *append-g.simps*(2) by *auto*
moreover **hence** $*$: $\Theta; \mathcal{B} \vdash_{wf} (\Gamma1 @ (x, b, c') \#_{\Gamma} G)$ using *GCons append-g.simps wfG-elim*s by *metis*
moreover **hence** *replace-in-g* $(\Gamma1 @ (x, b, c') \#_{\Gamma} G) x c = \Gamma1 @ (x, b, c) \#_{\Gamma} G$ using *GCons replace-in-g-inside*[*OF* $*$, *of* c] by *auto*

ultimately **show** *?case* using *replace-in-g.simps*(2)[*of* $x1 b1 c1 \Gamma1 @ (x, b, c') \#_{\Gamma} G x c$] *GCons*
by (*simp add*: *GCons.prem*s(1) *GCons.prem*s(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 $\Gamma = \text{replace-in-g } \Gamma' x c$ **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** $\Theta; \mathcal{B}; \text{replace-in-g } G x c \models c'$ **and** $\text{wfG } \Theta \mathcal{B} G$
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** $\text{replace-in-g } G x c = \Gamma' @ (x, b, c) \#_{\Gamma} \Gamma$ **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 $\Theta; \mathcal{B}; \Gamma' @ (x, b, c0') \#_{\Gamma} \Gamma \vdash_{\text{wf}} c$ **and** $\Theta; \mathcal{B}; (x, b, \text{TRUE}) \#_{\Gamma} \Gamma \vdash_{\text{wf}} c0$
shows $\Theta; \mathcal{B}; \Gamma' @ (x, b, c0) \#_{\Gamma} \Gamma \vdash_{\text{wf}} c$
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 $\Theta; \mathcal{B}; \Gamma' @ (x, b, c0) \#_{\Gamma} \Gamma \vdash_{\text{wf}} c$ **proof** –
have $\Theta; \mathcal{B}; \Gamma' @ (x, b, c0') \#_{\Gamma} \Gamma \vdash_{\text{wf}} c$ **using** *valid.simps* *assms* **by** *auto*
moreover **have** $\Theta; \mathcal{B}; (x, b, \text{TRUE}) \#_{\Gamma} \Gamma \vdash_{\text{wf}} c0$ **proof** –
have $\text{wfG } \Theta \mathcal{B} (\Gamma' @ (x, b, c0) \#_{\Gamma} \Gamma)$ **using** *assms valid.simps wfC-wf* **by** *auto*
hence $\text{wfG } \Theta \mathcal{B} ((x, b, c0) \#_{\Gamma} \Gamma)$ **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 $\forall i. \text{wfI } \Theta (\Gamma' @ (x, b, c0) \#_{\Gamma} \Gamma) i \wedge \text{is-satis-g } i (\Gamma' @ (x, b, c0) \#_{\Gamma} \Gamma) \longrightarrow \text{is-satis } i c$
proof(*rule, rule*)
fix i
assume $*$: $\text{wfI } \Theta (\Gamma' @ (x, b, c0) \#_{\Gamma} \Gamma) i \wedge \text{is-satis-g } i (\Gamma' @ (x, b, c0) \#_{\Gamma} \Gamma)$

hence $\text{is-satis-g } i (\Gamma' @ (x, b, c0) \#_{\Gamma} \Gamma) \wedge \text{wfI } \Theta (\Gamma' @ (x, b, c0) \#_{\Gamma} \Gamma) i$ **using** *is-satis-g-append*
wfI-suffix **by** *metis*
moreover **hence** $\text{is-satis } i c0'$ **using** *valid.simps* *assms* **by** *presburger*

moreover **have** $\text{is-satis-g } i \Gamma'$ **using** *is-satis-g-append ** **by** *simp*
ultimately **have** $\text{is-satis-g } i (\Gamma' @ (x, b, c0') \#_{\Gamma} \Gamma)$ **using** *is-satis-g-append* **by** *simp*
moreover **have** $\text{wfI } \Theta (\Gamma' @ (x, b, c0') \#_{\Gamma} \Gamma) i$ **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 $\Gamma :: \Gamma$

shows $\Theta; \mathcal{B}; G \vdash t1 \lesssim t2 \implies G = \Gamma' @ (x, b0, c0') \#_{\Gamma} \Gamma \implies \Theta; \mathcal{B}; \Gamma' @ (x, b0, c0) \#_{\Gamma} \Gamma \models c0' \implies \Theta; \mathcal{B}; \Gamma' @ (x, b0, c0) \#_{\Gamma} \Gamma \vdash t1 \lesssim t2$

proof (nominal-induct avoiding: *c0* rule: *subtype.strong-induct*)

case (*subtype-baseI* $x' \Theta \mathcal{B} \Gamma'' z c z' c' b$)

let $? \Gamma c0 = \Gamma' @ (x, b0, c0) \#_{\Gamma} \Gamma$

have *wb1*: $wfG \Theta \mathcal{B} ? \Gamma c0$ using *valid.simps wfC-wf subtype-baseI* by *metis*

show *?case* proof

show $\langle \Theta; \mathcal{B}; \Gamma' @ (x, b0, c0) \#_{\Gamma} \Gamma \vdash_{wf} \{ z : b \mid c \} \rangle$ using *wfT-replace-inside2[OF - wb1]* *subtype-baseI* by *metis*

show $\langle \Theta; \mathcal{B}; \Gamma' @ (x, b0, c0) \#_{\Gamma} \Gamma \vdash_{wf} \{ z' : b \mid c' \} \rangle$ using *wfT-replace-inside2[OF - wb1]* *subtype-baseI* by *metis*

have *atom* $x' \# \Gamma' @ (x, b0, c0) \#_{\Gamma} \Gamma$ using *fresh-prodN subtype-baseI fresh-replace-inside wb1 subtype-wf wfX-wfY* by *metis*

thus $\langle \text{atom } x' \# (\Theta, \mathcal{B}, \Gamma' @ (x, b0, c0) \#_{\Gamma} \Gamma, z, c, z', c') \rangle$ using *subtype-baseI fresh-prodN* by *metis*

have $\Theta; \mathcal{B}; ((x', b, c[z ::= V\text{-var } x]_v) \#_{\Gamma} \Gamma') @ (x, b0, c0) \#_{\Gamma} \Gamma \models c'[z ::= V\text{-var } x]_v$ proof (rule *ctx-subtype-valid*)

show 1: $\langle \Theta; \mathcal{B}; ((x', b, c[z ::= V\text{-var } x]_v) \#_{\Gamma} \Gamma') @ (x, b0, c0') \#_{\Gamma} \Gamma \models c'[z' ::= V\text{-var } x]_v \rangle$

using *subtype-baseI append-g.simps subst-defs* by *metis*

have $* \Theta; \mathcal{B} \vdash_{wf} ((x', b, c[z ::= V\text{-var } x]_v) \#_{\Gamma} \Gamma') @ (x, b0, c0) \#_{\Gamma} \Gamma$ proof (rule *wfG-replace-inside2*)

show $\Theta; \mathcal{B} \vdash_{wf} ((x', b, c[z ::= V\text{-var } x]_v) \#_{\Gamma} \Gamma') @ (x, b0, c0') \#_{\Gamma} \Gamma$

using $* \text{valid-wf-all } wfC\text{-wf } 1 \text{ append-g.simps}$ by *metis*

show $\Theta; \mathcal{B} \vdash_{wf} (x, b0, c0) \#_{\Gamma} \Gamma$ using *wfG-suffix wb1* by *auto*

qed

moreover have $\text{toSet } (\Gamma' @ (x, b0, c0) \#_{\Gamma} \Gamma) \subseteq \text{toSet } (((x', b, c[z ::= V\text{-var } x]_v) \#_{\Gamma} \Gamma') @ (x, b0, c0) \#_{\Gamma} \Gamma)$ using *toSet.simps append-g.simps* by *auto*

ultimately show $\langle \Theta; \mathcal{B}; ((x', b, c[z ::= V\text{-var } x]_v) \#_{\Gamma} \Gamma') @ (x, b0, c0) \#_{\Gamma} \Gamma \models c0' \rangle$ using *valid-weakening subtype-baseI ** by *blast*

qed

thus $\langle \Theta; \mathcal{B}; (x', b, c[z ::= V\text{-var } x]_v) \#_{\Gamma} \Gamma' @ (x, b0, c0) \#_{\Gamma} \Gamma \models c'[z' ::= V\text{-var } x]_v \rangle$ using *append-g.simps subst-defs* by *simp*

qed

qed

lemma *ctx-subtype-subtype-rig*:

assumes *replace-in-g-subtyped* $\Theta \mathcal{B} \Gamma' [(x, c0)] \Gamma$ and $\Theta; \mathcal{B}; \Gamma' \vdash t1 \lesssim t2$

shows $\Theta; \mathcal{B}; \Gamma \vdash t1 \lesssim t2$

proof –

have *wf*: $wfG \Theta \mathcal{B} \Gamma'$ using *subtype-g-wf assms* by *auto*

obtain *b* and *c0'* where $\text{Some } (b, c0') = \text{lookup } \Gamma' x \wedge (\Theta; \mathcal{B}; \text{replace-in-g } \Gamma' x c0 \models c0')$ using *replace-in-g-subtyped.simps[of $\Theta \mathcal{B} \Gamma' [(x, c0)] \Gamma$ 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* $b \ c0' \Gamma' x \Theta \mathcal{B} \ c0$] *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* $\Theta \mathcal{B} \Gamma' \ xcs \ \Gamma$ **and** $\Theta; \mathcal{B}; \Gamma' \vdash t1 \lesssim t2$

shows $\Theta; \mathcal{B}; \Gamma \vdash t1 \lesssim t2$

using *assms* **proof**(*induct xcs arbitrary: $\Gamma \Gamma'$*)

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* $(b, c') = \text{lookup } \Gamma' x \wedge$

replace-in-g-subtyped $\Theta \mathcal{B} (\text{replace-in-g } \Gamma' x c) \ xcs \ \Gamma \wedge \Theta; \mathcal{B}; \Gamma' \vdash_{wf} c \wedge$

$x \notin \text{fst } \text{'set } xcs \wedge \Theta; \mathcal{B}; (\text{replace-in-g } \Gamma' x c) \models c'$ **using** *replace-in-g-subtyped-elim3*(*3*)[*of* $\Theta \mathcal{B} \Gamma' x c \ xcs \ \Gamma$] *Cons*

by (*metis valid.simps*)

hence $*$: *replace-in-g-subtyped* $\Theta \mathcal{B} \Gamma' [(x,c)] (\text{replace-in-g } \Gamma' x c)$ **using** *replace-in-g-subtyped-consI*
by (*meson image-iff list.distinct(1) list.set-cases replace-in-g-subtyped-nilI*)

hence $\Theta; \mathcal{B}; (\text{replace-in-g } \Gamma' x c) \vdash t1 \lesssim t2$

using *ctx-subtype-subtype-rig * assms Cons.prem2* **by** *auto*

moreover have *replace-in-g-subtyped* $\Theta \mathcal{B} (\text{replace-in-g } \Gamma' x c) \ xcs \ \Gamma$ **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* $\Theta \mathcal{B} \Gamma' [(x, c0)] \ \Gamma$ **and** $wfG \ \Theta \mathcal{B} \Gamma'$

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* $(b, c0') = \text{lookup } \Gamma' x \wedge \Theta; \mathcal{B}; \text{replace-in-g } \Gamma' x c0 \models c0'$ **using**

replace-in-g-subtyped.simps[*of* $\Theta \mathcal{B} \Gamma' [(x, c0)] \ \Gamma$] *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** $*$: $\Gamma' = G'@ (x, b, c0') \#_{\Gamma} G \wedge \Theta; \mathcal{B}; G'@ (x, b, c0') \#_{\Gamma} G \models c0'$ **using**
replace-in-g-subtyped-split[*of b c0' $\Gamma' x \Theta \mathcal{B} c0$*] *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 $\Theta; \mathcal{B} \vdash G \langle xcs \rangle \rightsquigarrow G'$ **and** $\Theta; \mathcal{B}; G \models c$
shows $\langle \Theta; \mathcal{B}; G' \models c \rangle$
using *assms* **proof**(*induct rule: replace-in-g-subtyped.inducts*)
case (*replace-in-g-subtyped-nilI $\Theta \mathcal{B} G$*)
then show *?case* **by** *auto*
next
case (*replace-in-g-subtyped-consI b c1 G x $\Theta \mathcal{B} c2 xcs G'$*)
hence $\Theta; \mathcal{B}; G[x \mapsto c2] \models c$
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 $\Theta; B \vdash_{wf} (\Gamma'@ (x, b0, c0) \#_{\Gamma} \Gamma)$ **and** $\Theta; B \vdash_{wf} (\Gamma'@ (x, b0, c0') \#_{\Gamma} \Gamma)$
and *Some (b, c) = lookup ($\Gamma'@ (x, b0, c0') \#_{\Gamma} \Gamma$) y* **and** *Some (b0, c0) = lookup ($\Gamma'@ ((x, b0, c0)) \#_{\Gamma} \Gamma$)*
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*)

thm *infer-v-form2*

lemma *ctx-subtype-v-aux*:

fixes $v::v$
assumes $\Theta; \mathcal{B}; \Gamma'@ ((x, b0, c0') \#_{\Gamma} \Gamma) \vdash v \Rightarrow t1$ **and** $\Theta; \mathcal{B}; \Gamma'@ (x, b0, c0) \#_{\Gamma} \Gamma \models c0'$
shows $\Theta; \mathcal{B}; \Gamma'@ ((x, b0, c0) \#_{\Gamma} \Gamma) \vdash v \Rightarrow t1$
using *assms* **proof**(*nominal-induct $\Gamma'@ ((x, b0, c0') \#_{\Gamma} \Gamma) v t1$ avoiding: c0 rule: infer-v.strong-induct*)
case (*infer-v-varI $\Theta \mathcal{B} b c xa z$*)
have $wf: \langle \Theta; \mathcal{B} \vdash_{wf} \Gamma'@ (x, b0, c0) \#_{\Gamma} \Gamma \rangle$ **using** *wfG-inside-valid2 infer-v-varI* **by** *metis*
have $xf1: \langle atom\ z \# xa \rangle$ **using** *infer-v-varI* **by** *metis*
have $xf2: \langle atom\ z \# (\Theta, \mathcal{B}, \Gamma'@ (x, b0, c0) \#_{\Gamma} \Gamma) \rangle$ **apply**(*fresh-mth add: infer-v-varI*)
using *fresh-def infer-v-varI wfG-supp fresh-append-g fresh-GCons fresh-prodN* **by** *metis+*
show *?case* **proof** (*cases x=xa*)
case *True*
moreover **have** $b = b0$ **using** *infer-v-varI True* **by** *simp*
moreover **hence** $\langle Some (b, c0) = lookup (\Gamma'@ (x, b0, c0) \#_{\Gamma} \Gamma) xa \rangle$ **using** *lookup-inside-wf[OF wf] infer-v-varI True* **by** *auto*
ultimately show *?thesis* **using** *wf xf1 xf2 Typing.infer-v-varI* **by** *metis*
next
case *False*

moreover hence $\langle \text{Some } (b, c) = \text{lookup } (\Gamma' @ (x, b0, c0) \#_{\Gamma} \Gamma) \text{ } xa \rangle$ using *lookup-inside2*
infer-v-varI by *metis*
 ultimately show *?thesis* using *wf xf1 xf2 Typing.infer-v-varI* by *simp*
 qed
 next
 case (*infer-v-litI* $\Theta \mathcal{B} \text{ l } \tau$)
 thus *?case* using *Typing.infer-v-litI wfG-inside-valid2* by *simp*
 next
 case (*infer-v-pairI* $z \text{ v1 v2 } \Theta \mathcal{B} \text{ t1' t2' c0}$)
 show *?case* proof
 show *atom* $z \# (v1, v2)$ using *infer-v-pairI fresh-Pair* by *simp*
 show *atom* $z \# (\Theta, \mathcal{B}, \Gamma' @ (x, b0, c0) \#_{\Gamma} \Gamma)$ apply (*fresh-mth add: infer-v-pairI*)
 using *fresh-def infer-v-pairI wfG-supp fresh-append-g fresh-GCons fresh-prodN* by *metis+*
 show $\Theta; \mathcal{B}; \Gamma' @ (x, b0, c0) \#_{\Gamma} \Gamma \vdash v1 \Rightarrow t1'$ using *infer-v-pairI* by *simp*
 show $\Theta; \mathcal{B}; \Gamma' @ (x, b0, c0) \#_{\Gamma} \Gamma \vdash v2 \Rightarrow t2'$ using *infer-v-pairI* by *simp*
 qed
 next
 case (*infer-v-consI* $s \text{ dclist } \Theta \text{ dc tc } \mathcal{B} \text{ v tv z}$)
 show *?case* proof
 show $\langle \text{AF-typedef } s \text{ dclist} \in \text{set } \Theta \rangle$ using *infer-v-consI* by *auto*
 show $\langle (dc, tc) \in \text{set dclist} \rangle$ using *infer-v-consI* by *auto*
 show $\langle \Theta; \mathcal{B}; \Gamma' @ (x, b0, c0) \#_{\Gamma} \Gamma \vdash v \Rightarrow tv \rangle$ using *infer-v-consI* by *auto*
 show $\langle \Theta; \mathcal{B}; \Gamma' @ (x, b0, c0) \#_{\Gamma} \Gamma \vdash tv \lesssim tc \rangle$ using *infer-v-consI ctx-subtype-subtype* by *auto*
 show *atom* $z \# v$ using *infer-v-consI* by *auto*
 show *atom* $z \# (\Theta, \mathcal{B}, \Gamma' @ (x, b0, c0) \#_{\Gamma} \Gamma)$ apply (*fresh-mth add: infer-v-consI*)
 using *fresh-def infer-v-consI wfG-supp fresh-append-g fresh-GCons fresh-prodN* by *metis+*
 qed
 next
 case (*infer-v-conspI* $s \text{ bv dclist } \Theta \text{ dc tc } \mathcal{B} \text{ v tv b z}$)
 show *?case* proof
 show $\langle \text{AF-typedef-poly } s \text{ bv dclist} \in \text{set } \Theta \rangle$ using *infer-v-conspI* by *auto*
 show $\langle (dc, tc) \in \text{set dclist} \rangle$ using *infer-v-conspI* by *auto*
 show $\langle \Theta; \mathcal{B}; \Gamma' @ (x, b0, c0) \#_{\Gamma} \Gamma \vdash v \Rightarrow tv \rangle$ using *infer-v-conspI* by *auto*
 show $\langle \Theta; \mathcal{B}; \Gamma' @ (x, b0, c0) \#_{\Gamma} \Gamma \vdash tv \lesssim tc[bv::=b]_{\tau b} \rangle$ using *infer-v-conspI ctx-subtype-subtype*
 by *auto*
 show *atom* $z \# (\Theta, \mathcal{B}, \Gamma' @ (x, b0, c0) \#_{\Gamma} \Gamma, v, b)$ apply (*fresh-mth add: infer-v-conspI*)
 using *fresh-def infer-v-conspI wfG-supp fresh-append-g fresh-GCons fresh-prodN* by *metis+*
 show *atom* $bv \# (\Theta, \mathcal{B}, \Gamma' @ (x, b0, c0) \#_{\Gamma} \Gamma, v, b)$ apply (*fresh-mth add: infer-v-conspI*)
 using *fresh-def infer-v-conspI wfG-supp fresh-append-g fresh-GCons fresh-prodN* by *metis+*
 show $\langle \Theta; \mathcal{B} \vdash_{wf} b \rangle$ using *infer-v-conspI* by *auto*
 qed
 qed

 lemma *ctx-subtype-v*:
 fixes $v::v$
 assumes $\Theta; \mathcal{B}; \Gamma' @ ((x, b0, c0) \#_{\Gamma} \Gamma) \vdash v \Rightarrow t1$ and $\Theta; \mathcal{B}; \Gamma' @ (x, b0, c0) \#_{\Gamma} \Gamma \models c0'$
 shows $\exists t2. \Theta; \mathcal{B}; \Gamma' @ ((x, b0, c0) \#_{\Gamma} \Gamma) \vdash v \Rightarrow t2 \wedge \Theta; \mathcal{B}; \Gamma' @ ((x, b0, c0) \#_{\Gamma} \Gamma) \vdash t2 \lesssim t1$
 proof –
 have $\Theta; \mathcal{B}; \Gamma' @ ((x, b0, c0) \#_{\Gamma} \Gamma) \vdash v \Rightarrow t1$ using *ctx-subtype-v-aux assms* by *auto*
 moreover hence $\Theta; \mathcal{B}; \Gamma' @ ((x, b0, c0) \#_{\Gamma} \Gamma) \vdash t1 \lesssim t1$ using *subtype-reflI2 infer-v-wf* by *simp*
 ultimately show *?thesis* by *auto*

qed

lemma *ctx-subtype-v-eq*:

fixes $v::v$

assumes

$\Theta; \mathcal{B}; \Gamma' @ ((x, b0, c0') \#_{\Gamma} \Gamma) \vdash v \Rightarrow t1$ **and**

$\Theta; \mathcal{B}; \Gamma' @ (x, b0, c0) \#_{\Gamma} \Gamma \models c0'$

shows $\Theta; \mathcal{B}; \Gamma' @ ((x, b0, c0) \#_{\Gamma} \Gamma) \vdash v \Rightarrow t1$

proof –

obtain $t1'$ **where** $\Theta; \mathcal{B}; \Gamma' @ ((x, b0, c0) \#_{\Gamma} \Gamma) \vdash v \Rightarrow t1'$ **using** *ctx-subtype-v* **assms** **by** *metis*

moreover **have** *replace-in-g* $(\Gamma' @ ((x, b0, c0') \#_{\Gamma} \Gamma)) \ x \ c0 = \Gamma' @ ((x, b0, c0) \#_{\Gamma} \Gamma)$ **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 $\Theta; \mathcal{B}; \Gamma' @ ((x, b0, c0') \#_{\Gamma} \Gamma) \vdash v \Leftarrow t1$ **and** $\Theta; \mathcal{B}; \Gamma' @ (x, b0, c0) \#_{\Gamma} \Gamma \models c0'$

shows $\Theta; \mathcal{B}; \Gamma' @ ((x, b0, c0) \#_{\Gamma} \Gamma) \vdash v \Leftarrow t1$

proof –

obtain $t2$ **where** $t2: \Theta; \mathcal{B}; \Gamma' @ ((x, b0, c0') \#_{\Gamma} \Gamma) \vdash v \Rightarrow t2 \wedge \Theta; \mathcal{B}; \Gamma' @ ((x, b0, c0') \#_{\Gamma} \Gamma) \vdash t2 \lesssim t1$
using *check-v-elim* **assms** **by** *blast*

hence $t3: \Theta; \mathcal{B}; \Gamma' @ ((x, b0, c0) \#_{\Gamma} \Gamma) \vdash v \Rightarrow t2$

using *assms ctx-subtype-v-eq* **by** *blast*

have $\Theta; \mathcal{B}; \Gamma' @ ((x, b0, c0) \#_{\Gamma} \Gamma) \vdash v \Rightarrow t2$ **using** $t3$ **by** *auto*

moreover **have** $\Theta; \mathcal{B}; \Gamma' @ ((x, b0, c0) \#_{\Gamma} \Gamma) \vdash t2 \lesssim t1$ **proof** –

have $\Theta; \mathcal{B}; \Gamma' @ ((x, b0, c0') \#_{\Gamma} \Gamma) \vdash t2 \lesssim t1$ **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* $(b, c') = \text{lookup } \Gamma' x \wedge$
 $\text{replace-in-g-subtyped } \Theta \mathcal{B} (\text{replace-in-g } \Gamma' x c) xcs \Gamma \wedge \Theta; \mathcal{B}; \Gamma' \vdash_{wf} c \wedge$
 $x \notin \text{fst } \text{'set } xcs \wedge \Theta; \mathcal{B}; (\text{replace-in-g } \Gamma' x c) \models c'$
using *replace-in-g-subtyped-elim3[of $\Theta \mathcal{B} \Gamma' x c xcs \Gamma$] Cons* **by** (*metis valid.simps*)

hence *: *replace-in-g-subtyped* $\Theta \mathcal{B} \Gamma' [(x,c)] (\text{replace-in-g } \Gamma' x c)$ **using** *replace-in-g-subtyped-consI*
by (*meson image-iff list.distinct(1) list.set-cases replace-in-g-subtyped-nilI*)
hence $t2$: $\Theta; \mathcal{B}; (\text{replace-in-g } \Gamma' x c) \vdash v \Rightarrow t1$ **using** *ctx-subtype-v-rig-eq[OF * Cons(3)]* **by** *blast*
moreover have *: *replace-in-g-subtyped* $\Theta \mathcal{B} (\text{replace-in-g } \Gamma' x c) xcs \Gamma$ **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-elim 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 $\Theta; \mathcal{B}; \Gamma' @ (x, b0, c0) \#_{\Gamma} \Gamma \models c0'$
shows $\Theta; \mathcal{B}; (x, b0, \text{TRUE}) \#_{\Gamma} \Gamma \vdash_{wf} c0$
using *wfG-elim2 valid.simps wfG-suffix*
using *assms valid-g-wf* **by** *metis*

lemma *ctx-subtype-e-eq*:
fixes $G::\Gamma$
assumes
 $\Theta; \Phi; \mathcal{B}; G; \Delta \vdash e \Rightarrow t1$ **and** $G = \Gamma' @ ((x, b0, c0') \#_{\Gamma} \Gamma)$
 $\Theta; \mathcal{B}; \Gamma' @ (x, b0, c0) \#_{\Gamma} \Gamma \models c0'$
shows $\Theta; \Phi; \mathcal{B}; \Gamma' @ ((x, b0, c0) \#_{\Gamma} \Gamma); \Delta \vdash e \Rightarrow t1$
using *assms proof(nominal-induct t1 avoiding: c0 rule: infer-e.strong-induct)*

```

case (infer-e-valI  $\Theta \mathcal{B} \Gamma'' \Delta \Phi v \tau$ )
show ?case proof
  show  $\langle \Theta; \mathcal{B}; \Gamma' @ (x, b0, c0) \#_{\Gamma} \Gamma \vdash_{wf} \Delta \rangle$  using wf-replace-inside2(6) valid-wfC infer-e-valI by
auto
  show  $\langle \Theta \vdash_{wf} \Phi \rangle$  using infer-e-valI by auto
  show  $\langle \Theta; \mathcal{B}; \Gamma' @ (x, b0, c0) \#_{\Gamma} \Gamma \vdash v \Rightarrow \tau \rangle$  using infer-e-valI ctx-subtype-v-eq by auto
qed
next
case (infer-e-plusI  $\Theta \mathcal{B} \Gamma'' \Delta \Phi v1 z1 c1 v2 z2 c2 z3$ )
show ?case proof
  show  $\langle \Theta; \mathcal{B}; \Gamma' @ (x, b0, c0) \#_{\Gamma} \Gamma \vdash_{wf} \Delta \rangle$  using wf-replace-inside2(6) valid-wfC infer-e-plusI
by auto
  show  $\langle \Theta \vdash_{wf} \Phi \rangle$  using infer-e-plusI by auto
  show *:  $\langle \Theta; \mathcal{B}; \Gamma' @ (x, b0, c0) \#_{\Gamma} \Gamma \vdash v1 \Rightarrow \{ z1 : B\text{-int} \mid c1 \} \rangle$  using infer-e-plusI ctx-subtype-v-eq
by auto
  show  $\langle \Theta; \mathcal{B}; \Gamma' @ (x, b0, c0) \#_{\Gamma} \Gamma \vdash v2 \Rightarrow \{ z2 : B\text{-int} \mid c2 \} \rangle$  using infer-e-plusI ctx-subtype-v-eq
by auto
  show  $\langle atom\ z3 \# AE\text{-op}\ Plus\ v1\ v2 \rangle$  using infer-e-plusI by auto
  show  $\langle atom\ z3 \# \Gamma' @ (x, b0, c0) \#_{\Gamma} \Gamma \rangle$  using * infer-e-plusI fresh-replace-inside infer-v-wf by
metis
qed
next
case (infer-e-leqI  $\Theta \mathcal{B} \Gamma'' \Delta \Phi v1 z1 c1 v2 z2 c2 z3$ )
show ?case proof
  show  $\langle \Theta; \mathcal{B}; \Gamma' @ (x, b0, c0) \#_{\Gamma} \Gamma \vdash_{wf} \Delta \rangle$  using wf-replace-inside2(6) valid-wfC infer-e-leqI by
auto
  show  $\langle \Theta \vdash_{wf} \Phi \rangle$  using infer-e-leqI by auto
  show *:  $\langle \Theta; \mathcal{B}; \Gamma' @ (x, b0, c0) \#_{\Gamma} \Gamma \vdash v1 \Rightarrow \{ z1 : B\text{-int} \mid c1 \} \rangle$  using infer-e-leqI ctx-subtype-v-eq
by auto
  show  $\langle \Theta; \mathcal{B}; \Gamma' @ (x, b0, c0) \#_{\Gamma} \Gamma \vdash v2 \Rightarrow \{ z2 : B\text{-int} \mid c2 \} \rangle$  using infer-e-leqI ctx-subtype-v-eq
by auto
  show  $\langle atom\ z3 \# AE\text{-op}\ LEq\ v1\ v2 \rangle$  using infer-e-leqI by auto
  show  $\langle atom\ z3 \# \Gamma' @ (x, b0, c0) \#_{\Gamma} \Gamma \rangle$  using * infer-e-leqI fresh-replace-inside infer-v-wf by
metis
qed
next
case (infer-e-appI  $\Theta \mathcal{B} \Gamma'' \Delta \Phi f x' b c \tau' s' v \tau$ )
show ?case proof
  show  $\langle \Theta; \mathcal{B}; \Gamma' @ (x, b0, c0) \#_{\Gamma} \Gamma \vdash_{wf} \Delta \rangle$  using wf-replace-inside2(6) valid-wfC infer-e-appI
by auto
  show  $\langle \Theta \vdash_{wf} \Phi \rangle$  using infer-e-appI by auto
  show  $\langle Some\ (AF\text{-fundef}\ f\ (AF\text{-fun-typ-none}\ (AF\text{-fun-typ}\ x'\ b\ c\ \tau'\ s')) = lookup\text{-fun}\ \Phi\ f) \rangle$  using
infer-e-appI by auto
  show  $\langle \Theta; \mathcal{B}; \Gamma' @ (x, b0, c0) \#_{\Gamma} \Gamma \vdash v \Leftarrow \{ x' : b \mid c \} \rangle$  using infer-e-appI ctx-subtype-check-v-eq
by auto
  thus  $\langle atom\ x' \# (\Theta, \Phi, \mathcal{B}, \Gamma' @ (x, b0, c0) \#_{\Gamma} \Gamma, \Delta, v, \tau) \rangle$ 
    using infer-e-appI fresh-replace-inside[of  $\Theta \mathcal{B} \Gamma' x\ b0\ c0' \Gamma\ c0\ x$ ] infer-v-wf by auto
  show  $\langle \tau[x ::= v]_v = \tau \rangle$  using infer-e-appI by auto
qed
next
case (infer-e-appPI  $\Theta \mathcal{B} \Gamma1 \Delta \Phi b' f bv x1 b c \tau' s' v \tau$ )
show ?case proof

```

```

  show ⟨Θ; B; Γ' @ (x, b0, c0) #Γ Γ ⊢wf Δ⟩ using wf-replace-inside2(6) valid-wfC infer-e-appPI
by auto
  show ⟨Θ ⊢wf Φ⟩ using infer-e-appPI by auto
  show ⟨Θ; B ⊢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; B; Γ' @ (x, b0, c0) #Γ Γ ⊢ 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) #Γ Γ⟩ using fresh-replace-inside[of Θ 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') #Γ Γ using infer-e-appPI by metis
  hence atom bv # Γ' @ (x, b0, c0) #Γ Γ
  unfolding fresh-append-g fresh-GCons fresh-prod3 using ⟨atom bv # c0⟩ fresh-append-g by metis
  thus ⟨atom bv # (Θ, Φ, B, Γ' @ (x, b0, c0) #Γ Γ, Δ, b', v, τ)⟩ using infer-e-appPI by auto
qed
next
case (infer-e-fstI Θ B Γ'' Δ Φ v z' b1 b2 c z)
show ?case proof
  show ⟨Θ; B; Γ' @ (x, b0, c0) #Γ Γ ⊢wf Δ⟩ using wf-replace-inside2(6) valid-wfC infer-e-fstI by
auto
  show ⟨Θ ⊢wf Φ⟩ using infer-e-fstI by auto
  show ⟨Θ; B; Γ' @ (x, b0, c0) #Γ Γ ⊢ v ⇒ { z' : B-pair b1 b2 | c }⟩ using infer-e-fstI ctx-subtype-v-eq
by auto
  thus ⟨atom z # Γ' @ (x, b0, c0) #Γ Γ⟩ using infer-e-fstI fresh-replace-inside[of Θ 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 B Γ'' Δ Φ v z' b1 b2 c z)
show ?case proof
  show ⟨Θ; B; Γ' @ (x, b0, c0) #Γ Γ ⊢wf Δ⟩ using wf-replace-inside2(6) valid-wfC infer-e-sndI
by auto
  show ⟨Θ ⊢wf Φ⟩ using infer-e-sndI by auto
  show ⟨Θ; B; Γ' @ (x, b0, c0) #Γ Γ ⊢ v ⇒ { z' : B-pair b1 b2 | c }⟩ using infer-e-sndI
ctx-subtype-v-eq by auto
  thus ⟨atom z # Γ' @ (x, b0, c0) #Γ Γ⟩ using infer-e-sndI fresh-replace-inside[of Θ 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 B Γ'' Δ Φ v z' c z)
show ?case proof
  show ⟨Θ; B; Γ' @ (x, b0, c0) #Γ Γ ⊢wf Δ⟩ using wf-replace-inside2(6) valid-wfC infer-e-lenI by
auto
  show ⟨Θ ⊢wf Φ⟩ using infer-e-lenI by auto
  show ⟨Θ; B; Γ' @ (x, b0, c0) #Γ Γ ⊢ v ⇒ { z' : B-bitvec | c }⟩ using infer-e-lenI ctx-subtype-v-eq
by auto
  thus ⟨atom z # Γ' @ (x, b0, c0) #Γ Γ⟩ using infer-e-lenI fresh-replace-inside[of Θ 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  $\Theta \mathcal{B} \Gamma'' \Phi \Delta u \tau$ )
  show ?case proof
    show  $\Theta; \mathcal{B}; \Gamma' @ (x, b0, c0) \#_{\Gamma} \Gamma \vdash_{wf} \Delta$  using wf-replace-inside2(6) valid-wfC infer-e-mvarI
  by auto
    thus  $\Theta; \mathcal{B} \vdash_{wf} \Gamma' @ (x, b0, c0) \#_{\Gamma} \Gamma$  using infer-e-mvarI fresh-replace-inside wfD-wf by blast
    show  $\Theta \vdash_{wf} \Phi$  using infer-e-mvarI by auto
    show  $(u, \tau) \in setD \Delta$  using infer-e-mvarI by auto
  qed
next
  case (infer-e-concatI  $\Theta \mathcal{B} \Gamma'' \Delta \Phi v1 z1 c1 v2 z2 c2 z3$ )
  show ?case proof
    show  $\langle \Theta; \mathcal{B}; \Gamma' @ (x, b0, c0) \#_{\Gamma} \Gamma \vdash_{wf} \Delta \rangle$  using wf-replace-inside2(6) valid-wfC infer-e-concatI
  by auto
    thus  $\langle atom\ z3 \# \Gamma' @ (x, b0, c0) \#_{\Gamma} \Gamma \rangle$  using infer-e-concatI fresh-replace-inside[of  $\Theta \mathcal{B} \Gamma' x b0$ 
 $c0' \Gamma c0 z3$ ] infer-v-wf wfX-wfY by metis
    show  $\langle \Theta \vdash_{wf} \Phi \rangle$  using infer-e-concatI by auto
    show  $\langle \Theta; \mathcal{B}; \Gamma' @ (x, b0, c0) \#_{\Gamma} \Gamma \vdash v1 \Rightarrow \llbracket z1 : B-bitvec \mid c1 \rrbracket \rangle$  using infer-e-concatI
    ctx-subtype-v-eq by auto
    show  $\langle \Theta; \mathcal{B}; \Gamma' @ (x, b0, c0) \#_{\Gamma} \Gamma \vdash v2 \Rightarrow \llbracket z2 : B-bitvec \mid c2 \rrbracket \rangle$  using infer-e-concatI
    ctx-subtype-v-eq by auto
    show  $\langle atom\ z3 \# AE-concat\ v1\ v2 \rangle$  using infer-e-concatI by auto
  qed
next
  case (infer-e-splitI  $\Theta \mathcal{B} \Gamma'' \Delta \Phi v1 z1 c1 v2 z2 z3$ )
  show ?case proof
    show  $\langle \Theta; \mathcal{B}; \Gamma' @ (x, b0, c0) \#_{\Gamma} \Gamma \vdash_{wf} \Delta \rangle$  using wf-replace-inside2(6) valid-wfC infer-e-splitI
  by auto
    show  $\langle \Theta \vdash_{wf} \Phi \rangle$  using infer-e-splitI by auto
    show  $\langle \Theta; \mathcal{B}; \Gamma' @ (x, b0, c0) \#_{\Gamma} \Gamma \vdash v1 \Rightarrow \llbracket z1 : B-bitvec \mid c1 \rrbracket \rangle$  using infer-e-splitI ctx-subtype-v-eq
  by auto
    show  $\langle \Theta; \mathcal{B}; \Gamma' @$ 
       $(x, b0, c0) \#_{\Gamma}$ 
       $\Gamma \vdash v2 \Leftarrow \llbracket z2 : B-int \mid [leq\ [ [L-num\ 0]^v ]^{ce} [ [z2]^v ]^{ce} ]^{ce} == [ [L-true]^v ]^{ce} \text{ AND}$ 
       $[leq\ [ [z2]^v ]^{ce} [ [v1]^v ]^{ce} ]^{ce} == [ [L-true]^v ]^{ce} ] \rrbracket \rangle$ 
      using infer-e-splitI ctx-subtype-check-v-eq by auto

    show  $\langle atom\ z1 \# \Gamma' @ (x, b0, c0) \#_{\Gamma} \Gamma \rangle$  using fresh-replace-inside[of  $\Theta \mathcal{B} \Gamma' x b0 c0' \Gamma c0 z1$ ]
    infer-e-splitI infer-v-wf wfX-wfY * by metis
    show  $\langle atom\ z2 \# \Gamma' @ (x, b0, c0) \#_{\Gamma} \Gamma \rangle$  using fresh-replace-inside[of  $\Theta \mathcal{B} \Gamma' x b0 c0' \Gamma c0$ ]
    infer-e-splitI infer-v-wf wfX-wfY * by metis
    show  $\langle atom\ z3 \# \Gamma' @ (x, b0, c0) \#_{\Gamma} \Gamma \rangle$  using fresh-replace-inside[of  $\Theta \mathcal{B} \Gamma' x b0 c0' \Gamma c0$ ]
    infer-e-splitI infer-v-wf wfX-wfY * by metis
    show  $\langle atom\ z1 \# AE-split\ v1\ v2 \rangle$  using infer-e-splitI by auto
    show  $\langle atom\ z2 \# AE-split\ v1\ v2 \rangle$  using infer-e-splitI by auto
    show  $\langle atom\ z3 \# AE-split\ v1\ v2 \rangle$  using infer-e-splitI by auto
  qed
qed

```

lemma ctx-subtype-e-rig-eq:

assumes replace-in-g-subtyped $\Theta \mathcal{B} \Gamma' [(x, c0)] \Gamma$ and

$\Theta; \Phi; \mathcal{B}; \Gamma'; \Delta \vdash e \Rightarrow t1$

shows $\Theta ; \Phi ; \mathcal{B} ; \Gamma ; \Delta \vdash e \Rightarrow t1$
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* $\Theta \mathcal{B} \Gamma' xcs \Gamma$ and
 $\Theta ; \Phi ; \mathcal{B} ; \Gamma' ; \Delta \vdash e \Rightarrow t1$
shows $\Theta ; \Phi ; \mathcal{B} ; \Gamma ; \Delta \vdash e \Rightarrow t1$
using *assms proof(induct xcs arbitrary: $\Gamma \Gamma' t1$)*
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 $bc: \text{Some } (b, c') = \text{lookup } \Gamma' x \wedge$
 $\text{replace-in-g-subtyped } \Theta \mathcal{B} (\text{replace-in-g } \Gamma' x c) xcs \Gamma \wedge \Theta ; \mathcal{B} ; \Gamma' \vdash_{wf} c \wedge$
 $x \notin \text{fst 'set } xcs \wedge \Theta ; \mathcal{B} ; (\text{replace-in-g } \Gamma' x c) \models c'$ **using** *replace-in-g-subtyped-elim(3)[of*
 $\Theta \mathcal{B} \Gamma' x c xcs \Gamma]$ *Cons*
by (*metis valid.simps*)

hence *: *replace-in-g-subtyped* $\Theta \mathcal{B} \Gamma' [(x, c)] (\text{replace-in-g } \Gamma' x c)$ **using** *replace-in-g-subtyped-consI*
by (*meson image-iff list.distinct(1) list.set-cases replace-in-g-subtyped-nilI*)
hence $t2: \Theta ; \Phi ; \mathcal{B} ; (\text{replace-in-g } \Gamma' x c) ; \Delta \vdash e \Rightarrow t1$ **using** *ctx-subtype-e-rig-eq[OF * Cons(3)]*
by *blast*
moreover have **: *replace-in-g-subtyped* $\Theta \mathcal{B} (\text{replace-in-g } \Gamma' x c) xcs \Gamma$ **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 $G'::\Gamma$ and $xcs :: (x*c) \text{ list}$ and $css::\text{branch-list}$
shows
 $\text{check-s } \Theta \Phi \mathcal{B} G \Delta \ s \ t1 \Longrightarrow wsX \ G \ xcs \Longrightarrow \text{replace-in-g-subtyped } \Theta \mathcal{B} G \ xcs \ G' \Longrightarrow \text{check-s}$
 $\Theta \Phi \mathcal{B} G' \Delta \ s \ t1$ **and**
 $\text{check-branch-s } \Theta \Phi \mathcal{B} G \Delta \ tid \ cons \ const \ v \ cs \ t1 \Longrightarrow wsX \ G \ xcs \Longrightarrow \text{replace-in-g-subtyped}$
 $\Theta \mathcal{B} G \ xcs \ G' \Longrightarrow \text{check-branch-s } \Theta \Phi \mathcal{B} G' \Delta \ tid \ cons \ const \ v \ cs \ t1$
 $\text{check-branch-list } \Theta \Phi \mathcal{B} G \Delta \ tid \ dclist \ v \ css \ t1 \Longrightarrow wsX \ G \ xcs \Longrightarrow \text{replace-in-g-subtyped } \Theta$
 $\mathcal{B} G \ xcs \ G' \Longrightarrow \text{check-branch-list } \Theta \Phi \mathcal{B} G' \Delta \ tid \ dclist \ v \ css \ t1$
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 $\ast; \Theta; \mathcal{B}; G' \vdash v \Rightarrow \tau' \wedge \Theta; \mathcal{B}; G' \vdash \tau' \lesssim \tau$ **using** *ctx-subtype-v-rigs-eq ctx-subtype-subtype-rigs*

by (*meson check-v.simps*)

show *?case proof*

show $\langle \Theta; \mathcal{B}; G' \vdash_{wf} \Delta \rangle$ **using** *check-valI wfD-rig* **by** *auto*

show $\langle \Theta \vdash_{wf} \Phi \rangle$ **using** *check-valI* **by** *auto*

show $\langle \Theta; \mathcal{B}; G' \vdash v \Rightarrow \tau' \rangle$ **using** \ast **by** *auto*

show $\langle \Theta; \mathcal{B}; G' \vdash \tau' \lesssim \tau \rangle$ **using** \ast **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 $atom\ x \# (\Theta, \Phi, \mathcal{B}, G', \Delta, e, \tau)$ **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 $atom\ z' \# (x, \Theta, \Phi, \mathcal{B}, G', \Delta, e, \tau, s)$ **using** *check-letI fresh-prodN* **by** *metis*

show $\Theta; \Phi; \mathcal{B}; G'; \Delta \vdash e \Rightarrow \{ z' : b' \mid c' \}$

using *check-letI ctx-subtype-e-rigs-eq* **by** *blast*

show $\Theta; \Phi; \mathcal{B}; (x, b', c'[z'::=V-var\ x]_v) \#_{\Gamma} G'; \Delta \vdash s \Leftarrow \tau$

proof(*rule check-letI(5)*)

have $vld: \Theta; \mathcal{B}; ((x, b', c'[z'::=V-var\ x]_v) \#_{\Gamma} \Gamma) \models c'[z'::=V-var\ x]_{cv}$ **proof** –

have $wfG\ \Theta\ \mathcal{B}\ ((x, b', c'[z'::=V-var\ x]_v) \#_{\Gamma} \Gamma)$ **using** *check-letI check-s-wf* **by** *metis*

hence $wfC\ \Theta\ \mathcal{B}\ ((x, b', c'[z'::=V-var\ x]_v) \#_{\Gamma} \Gamma)\ (c'[z'::=V-var\ x]_{cv})$ **using** *wfC-refl subst-defs*

by *auto*

thus *?thesis* **using** *valid-refl[of \Theta \mathcal{B} x b' c'[z'::=V-var\ x]_v \Gamma\ c'[z'::=V-var\ x]_v]* *subst-defs* **by**

auto

qed

have $xf: x \notin fst\ 'set\ xcs$ **proof** –

have $atom\ 'fst\ 'set\ xcs \subseteq atom-dom\ \Gamma$ **using** *check-letI wsX-iff* **by** *meson*

moreover **have** $wfG\ \Theta\ \mathcal{B}\ \Gamma$ **using** *infer-e-wf check-letI* **by** *metis*

ultimately **show** *?thesis* **using** *fresh-def check-letI wfG-dom-sup*

using *wsX-fresh* **by** *auto*

qed

show *replace-in-g-subtyped* $\Theta\ \mathcal{B}\ ((x, b', c'[z'::=V-var\ x]_v) \#_{\Gamma} \Gamma)\ ((x, c'[z'::=V-var\ x]_v) \#_{\Gamma} xcs)$ **proof** –

have $Some\ (b', c'[z'::=V-var\ x]_v) = lookup\ ((x, b', c'[z'::=V-var\ x]_v) \#_{\Gamma} \Gamma)\ x$ **by** *auto*

moreover **have** $\Theta; \mathcal{B}; replace-in-g\ ((x, b', c'[z'::=V-var\ x]_v) \#_{\Gamma} \Gamma)\ x\ (c'[z'::=V-var\ x]_v) \models c'[z'::=V-var\ x]_v$ **proof** –

have *replace-in-g* $((x, b', c'[z'::=V-var\ x]_v) \#_{\Gamma} \Gamma)\ x\ (c'[z'::=V-var\ x]_v) = ((x, b', c'[z'::=V-var\ x]_v) \#_{\Gamma} \Gamma)$

using *replace-in-g.simps* **by** *presburger*

thus *?thesis* **using** *vld subst-defs* **by** *auto*

qed

moreover **have** *replace-in-g-subtyped* $\Theta\ \mathcal{B}\ (replace-in-g\ ((x, b', c'[z'::=V-var\ x]_v) \#_{\Gamma} \Gamma)\ x)$

$(c'[z'::=V\text{-var } x]_v)) \text{ } xcs \text{ } ((x, b', c'[z'::=V\text{-var } x]_v) \#_{\Gamma} G')) \text{ } \mathbf{proof} \text{ } -$
 $\text{have } wfG \text{ } \Theta \text{ } \mathcal{B} \text{ } ((x, b', c'[z'::=V\text{-var } x]_v) \#_{\Gamma} \Gamma)) \text{ } \mathbf{using} \text{ } check\text{-}letI \text{ } check\text{-}s\text{-}wf \text{ } \mathbf{by} \text{ } metis$
 $\text{hence } replace\text{-}in\text{-}g\text{-subtyped} \text{ } \Theta \text{ } \mathcal{B} \text{ } ((x, b', c'[z'::=V\text{-var } x]_v) \#_{\Gamma} \Gamma)) \text{ } xcs \text{ } ((x, b', c'[z'::=V\text{-var } x]_v) \#_{\Gamma} G'))$
 $\text{using } check\text{-}letI \text{ } replace\text{-}in\text{-}g\text{-subtyped}\text{-}cons \text{ } xf \text{ } \mathbf{by} \text{ } meson$
 $\text{moreover have } replace\text{-}in\text{-}g \text{ } ((x, b', c'[z'::=V\text{-var } x]_v) \#_{\Gamma} \Gamma) \text{ } x \text{ } (c'[z'::=V\text{-var } x]_v) = \text{ } ((x, b', c'[z'::=V\text{-var } x]_v) \#_{\Gamma} \Gamma))$
 $\text{using } replace\text{-}in\text{-}g.\text{simps} \text{ } \mathbf{by} \text{ } presburger$
 $\text{ultimately show } ?thesis \text{ } \mathbf{by} \text{ } argo$
 \mathbf{qed}
 $\text{moreover have } \Theta; \mathcal{B}; (x, b', c'[z'::=V\text{-var } x]_v) \#_{\Gamma} \Gamma \text{ } \vdash_{wf} \text{ } c'[z'::=V\text{-var } x]_v \text{ } \mathbf{using} \text{ } vld$
 $subst\text{-}defs \text{ } \mathbf{by} \text{ } auto$
 $\text{ultimately show } ?thesis \text{ } \mathbf{using} \text{ } replace\text{-}in\text{-}g\text{-subtyped}\text{-}consI \text{ } xf \text{ } replace\text{-}in\text{-}g.\text{simps}(2) \text{ } \mathbf{by} \text{ } metis$
 \mathbf{qed}
 $\text{show } wsX \text{ } ((x, b', c'[z'::=V\text{-var } x]_v) \#_{\Gamma} \Gamma) \text{ } ((x, c'[z'::=V\text{-var } x]_v) \# \text{ } xcs)$
 $\text{using } check\text{-}letI \text{ } xf \text{ } subst\text{-}defs \text{ } \mathbf{by} \text{ } (simp \text{ } add: \text{ } wsX\text{-}cons)$
 \mathbf{qed}
 \mathbf{qed}
 \mathbf{next}
 $\text{case } (check\text{-}branch\text{-}list\text{-}consI \text{ } \Theta \text{ } \Phi \text{ } \mathcal{B} \text{ } \Gamma \text{ } \Delta \text{ } tid \text{ } dclist \text{ } v \text{ } cs \text{ } \tau \text{ } css)$
 $\text{then show } ?case \text{ } \mathbf{using} \text{ } Typing.\text{check}\text{-}branch\text{-}list\text{-}consI \text{ } \mathbf{by} \text{ } auto$
 \mathbf{next}
 $\text{case } (check\text{-}branch\text{-}list\text{-}finalI \text{ } \Theta \text{ } \Phi \text{ } \mathcal{B} \text{ } \Gamma \text{ } \Delta \text{ } tid \text{ } dclist \text{ } v \text{ } cs \text{ } \tau)$
 $\text{then show } ?case \text{ } \mathbf{using} \text{ } Typing.\text{check}\text{-}branch\text{-}list\text{-}finalI \text{ } \mathbf{by} \text{ } auto$
 \mathbf{next}
 $\text{case } (check\text{-}branch\text{-}s\text{-}branchI \text{ } \Theta \text{ } \mathcal{B} \text{ } \Gamma \text{ } \Delta \text{ } \tau \text{ } const \text{ } x \text{ } \Phi \text{ } tid \text{ } cons \text{ } v \text{ } s)$
 $\text{have } wfcons: wfG \text{ } \Theta \text{ } \mathcal{B} \text{ } ((x, b\text{-}of \text{ } const, CE\text{-}val \text{ } v == CE\text{-}val \text{ } (V\text{-}cons \text{ } tid \text{ } cons \text{ } (V\text{-}var \text{ } x)) \text{ } AND \text{ } c\text{-}of \text{ } const \text{ } x) \#_{\Gamma} \Gamma) \text{ } \mathbf{using} \text{ } check\text{-}s\text{-}wf \text{ } check\text{-}branch\text{-}s\text{-}branchI$
 $\text{by } meson$
 $\text{hence } wf: wfG \text{ } \Theta \text{ } \mathcal{B} \text{ } \Gamma \text{ } \mathbf{using} \text{ } wfG\text{-}cons \text{ } \mathbf{by} \text{ } metis$
 $\text{moreover have } atom \text{ } x \text{ } \nmid (const, G', v) \text{ } \mathbf{proof} \text{ } -$
 $\text{have } atom \text{ } x \text{ } \nmid G' \text{ } \mathbf{using} \text{ } check\text{-}branch\text{-}s\text{-}branchI \text{ } wf \text{ } replace\text{-}in\text{-}g\text{-}fresh$
 $wfG\text{-}dom\text{-}supp \text{ } replace\text{-}in\text{-}g\text{-}wfG \text{ } \mathbf{by} \text{ } simp$
 $\text{thus } ?thesis \text{ } \mathbf{using} \text{ } check\text{-}branch\text{-}s\text{-}branchI \text{ } fresh\text{-}prodN \text{ } \mathbf{by} \text{ } simp$
 \mathbf{qed}
 $\text{moreover have } st: \Theta; \Phi; \mathcal{B}; (x, b\text{-}of \text{ } const, CE\text{-}val \text{ } v == CE\text{-}val(V\text{-}cons \text{ } tid \text{ } cons \text{ } (V\text{-}var \text{ } x)) \text{ } AND \text{ } c\text{-}of \text{ } const \text{ } x) \#_{\Gamma} G'; \Delta \text{ } \vdash s \Leftarrow \tau \text{ } \mathbf{proof} \text{ } -$
 $\text{have } wsX \text{ } ((x, b\text{-}of \text{ } const, CE\text{-}val \text{ } v == CE\text{-}val(V\text{-}cons \text{ } tid \text{ } cons \text{ } (V\text{-}var \text{ } x)) \text{ } AND \text{ } c\text{-}of \text{ } const \text{ } x) \#_{\Gamma} \Gamma) \text{ } xcs \text{ } \mathbf{using} \text{ } check\text{-}branch\text{-}s\text{-}branchI \text{ } wsX\text{-}cons2 \text{ } wsX\text{-}fresh \text{ } wf \text{ } \mathbf{by} \text{ } force$
 $\text{moreover have } replace\text{-}in\text{-}g\text{-subtyped} \text{ } \Theta \text{ } \mathcal{B} \text{ } ((x, b\text{-}of \text{ } const, CE\text{-}val \text{ } v == CE\text{-}val(V\text{-}cons \text{ } tid \text{ } cons \text{ } (V\text{-}var \text{ } x)) \text{ } AND \text{ } c\text{-}of \text{ } const \text{ } x) \#_{\Gamma} \Gamma) \text{ } xcs \text{ } ((x, b\text{-}of \text{ } const, CE\text{-}val \text{ } v == CE\text{-}val(V\text{-}cons \text{ } tid \text{ } cons \text{ } (V\text{-}var \text{ } x)) \text{ } AND \text{ } c\text{-}of \text{ } const \text{ } x) \#_{\Gamma} G')$
 $\text{using } replace\text{-}in\text{-}g\text{-subtyped}\text{-}cons \text{ } wsX\text{-}fresh \text{ } wf \text{ } check\text{-}branch\text{-}s\text{-}branchI \text{ } wfcons \text{ } \mathbf{by} \text{ } auto$
 $\text{thus } ?thesis \text{ } \mathbf{using} \text{ } check\text{-}branch\text{-}s\text{-}branchI \text{ } calculation \text{ } \mathbf{by} \text{ } meson$
 \mathbf{qed}
 $\text{moreover have } wft: wfT \text{ } \Theta \text{ } \mathcal{B} \text{ } G' \text{ } \tau \text{ } \mathbf{using}$
 $check\text{-}branch\text{-}s\text{-}branchI \text{ } ctr\text{-}subtype\text{-}subtype\text{-}rigs \text{ } subtype\text{-}reflI2 \text{ } subtype\text{-}wf \text{ } \mathbf{by} \text{ } metis$

moreover have $wfD \Theta \Phi \mathcal{B} G' \Delta$ 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* $z \Theta \Phi \mathcal{B} \Gamma \Delta v s1 s2 \tau$)
 hence $wf:wfG \Theta \Phi \mathcal{B} \Gamma$ using *check-s-wf* by *presburger*
 show $?case$ proof(*rule check-s-check-branch-s-check-branch-list.check-ifI*)
 show $\langle atom\ z \# (\Theta, \Phi, \mathcal{B}, G', \Delta, v, s1, s2, \tau) \rangle$ using *fresh-prodN replace-in-g-fresh1 wf check-ifI*
 by *auto*
 show $\langle \Theta; \mathcal{B}; G' \vdash v \Leftarrow \{ z : B\text{-bool} \mid TRUE \} \rangle$ using *ctx-subtype-check-v-rigs-eq check-ifI* by
presburger
 show $\langle \Theta; \Phi; \mathcal{B}; G'; \Delta \vdash s1 \Leftarrow \{ z : b\text{-of } \tau \mid CE\text{-val } v == CE\text{-val } (V\text{-lit } L\text{-true}) \} \text{ IMP } c\text{-of } \tau \text{ } z \} \rangle$ using *check-ifI* by *auto*
 show $\langle \Theta; \Phi; \mathcal{B}; G'; \Delta \vdash s2 \Leftarrow \{ z : b\text{-of } \tau \mid CE\text{-val } v == CE\text{-val } (V\text{-lit } L\text{-false}) \} \text{ IMP } c\text{-of } \tau \text{ } z \} \rangle$ using *check-ifI* by *auto*
 qed
 next

case (*check-let2I* $x P \Phi \mathcal{B} G \Delta t s1 \tau s2$)
 show $?case$ proof
 have $wfG P \mathcal{B} G$ using *check-let2I check-s-wf* by *metis*
 show $*: P; \Phi; \mathcal{B}; G'; \Delta \vdash s1 \Leftarrow t$ using *check-let2I* by *blast*
 show $atom\ x \# (P, \Phi, \mathcal{B}, G', \Delta, t, s1, \tau)$ proof –
 have $wfG P \mathcal{B} G'$ using *check-s-wf ** by *blast*
 hence $atom\text{-}dom\ G = atom\text{-}dom\ G'$ 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-suppl fresh-def wfG P B G'* by *metis*
 thus $?thesis$ using *check-let2I* by *auto*
 qed
 show $P; \Phi; \mathcal{B}; (x, b\text{-of } t, c\text{-of } t\ x) \#_{\Gamma} G'; \Delta \vdash s2 \Leftarrow \tau$ proof –
 have $wsX ((x, b\text{-of } t, c\text{-of } t\ x) \#_{\Gamma} G) \text{ } xcs$ using *check-let2I wsX-cons2 wsX-fresh wfG P B G*
 by *simp*
 moreover have *replace-in-g-subtyped* $P \mathcal{B} ((x, b\text{-of } t, c\text{-of } t\ x) \#_{\Gamma} G) \text{ } xcs ((x, b\text{-of } t, c\text{-of } t\ x) \#_{\Gamma} G')$ proof(*rule replace-in-g-subtyped-cons*)
 show *replace-in-g-subtyped* $P \mathcal{B} G \text{ } xcs G'$ using *check-let2I* by *auto*
 have $atom\ x \# G$ using *check-let2I* by *auto*
 moreover have $wfT P \mathcal{B} G t$ using *check-let2I check-s-wf* by *metis*
 moreover have $atom\ x \# t$ using *check-let2I check-s-wf wfT-suppl* 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 $x \notin fst\ \text{'set } xcs$ using *check-let2I wsX-fresh wfG P B G* by *simp*
 qed
 ultimately show $?thesis$ using *check-let2I* by *presburger*
 qed
 qed
 next

case (*check-varI* $u \Theta \Phi \mathcal{B} \Gamma \Delta \tau' v \tau s$)
 show $?case$ proof

```

have atom u # 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 # (Θ, Φ, B, G', Δ, τ', v, τ)⟩ unfolding fresh-prodN using check-varI by simp
show ⟨Θ; B; G' ⊢ v ⇐ τ'⟩ using ctx-subtype-check-v-rigs-eq check-varI by auto
show ⟨Θ; Φ; B; G'; (u, τ') #Δ Δ ⊢ s ⇐ τ⟩ using check-varI by auto
qed
next
case (check-assignI P Φ B G Δ u τ v z τ')
show ?case proof
  show ⟨P ⊢wf Φ⟩ using check-assignI by auto
  show ⟨P; B; G' ⊢wf Δ⟩ using check-assignI wfD-rig by auto
  show ⟨(u, τ) ∈ setD Δ⟩ using check-assignI by auto
  show ⟨P; B; G' ⊢ v ⇐ τ⟩ using ctx-subtype-check-v-rigs-eq check-assignI by auto
  show ⟨P; B; G' ⊢ ⋈ z : B-unit | TRUE ⋈ ≲ 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 B Γ Δ tid dclist v cs τ z)
show ?case proof
  show Θ; Φ; B; G'; Δ; tid; dclist; v ⊢ cs ⇐ τ using check-caseI ctx-subtype-check-v-rigs-eq
by auto
  show AF-typedef tid dclist ∈ set Θ using check-caseI by auto
  show Θ; B; G' ⊢ v ⇐ ⋈ z : B-id tid | TRUE ⋈ using check-caseI ctx-subtype-check-v-rigs-eq by
auto
  show ⊢wf Θ using check-caseI by auto
qed
next
case (check-assertI x Θ Φ B Γ Δ c τ s)
show ?case proof
  have wfG: Θ; B ⊢wf Γ ∧ Θ; B ⊢wf G' using check-s-wf check-assertI replace-in-g-wfG wfX-wfY by
metis
  hence atom x # G' using check-assertI replace-in-g-fresh replace-in-g-wfG by auto
  thus ⟨atom x # (Θ, Φ, B, G', Δ, c, τ, s)⟩ using check-assertI fresh-prodN by auto
  show ⟨Θ; Φ; B; (x, B-bool, c) #Γ G'; Δ ⊢ s ⇐ τ⟩ proof(rule check-assertI(5))
    show wsX ((x, B-bool, c) #Γ Γ) xcs using check-assertI wsX-cons3 by simp
  show Θ; B ⊢ (x, B-bool, c) #Γ Γ ⟨xcs⟩ ∼ (x, B-bool, c) #Γ G' proof(rule replace-in-g-subtyped-cons)
    show ⟨Θ; B ⊢ Γ ⟨xcs⟩ ∼ G'⟩ using check-assertI by auto
    show ⟨Θ; B ⊢wf (x, B-bool, c) #Γ Γ⟩ using check-assertI check-s-wf by metis
    thus ⟨x ∉ fst 'set xcs'⟩ using check-assertI wsX-fresh wfG-elim wsX-wfY by metis
  qed
qed
show ⟨Θ; B; G' ⊢ c⟩ using check-assertI replace-in-g-valid by auto
show ⟨Θ; B; G' ⊢wf Δ⟩ 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 $replace\text{-in-g-subtyped} \ \Theta \ \mathcal{B} \ (replace\text{-in-g} \ (\Gamma' @ (x, b, c[z ::= V\text{-var } x]_{cv}) \#_{\Gamma} \Gamma) \ x \ (c'[z' ::= V\text{-var } x]_{cv})) \ [] \ (\Gamma' @ (x, b, c'[z' ::= V\text{-var } x]_{cv}) \#_{\Gamma} \Gamma)$
proof –
have $replace\text{-in-g} \ (\Gamma' @ (x, b, c[z ::= V\text{-var } x]_{cv}) \#_{\Gamma} \Gamma) \ x \ (c'[z' ::= V\text{-var } x]_{cv}) = (\Gamma' @ (x, b, c'[z' ::= V\text{-var } x]_{cv}) \#_{\Gamma} \Gamma)$
using *assms proof(induct Γ' rule: Γ -induct)*
case *GNil*
then show *?case using replace-in-g.simps by auto*
next
case $(GCons \ x1 \ b1 \ c1 \ \Gamma1)$
have $x \notin fst \ ' \ toSet \ ((x1, b1, c1) \#_{\Gamma} \Gamma1) \ \text{using } GCons \ wfG\text{-inside-fresh} \ atom\text{-dom.simps} \ dom.simps \ toSet.simps \ append\text{-g.simps} \ \text{by fast}$
hence $x1 \neq x \ \text{using } assms \ wfG\text{-inside-fresh} \ GCons \ \text{by force}$
hence $((x1, b1, c1) \#_{\Gamma} (\Gamma1 @ (x, b, c[z ::= V\text{-var } x]_{cv}) \#_{\Gamma} \Gamma)) [x \mapsto c'[z' ::= V\text{-var } x]_{cv}] = (x1, b1, c1) \#_{\Gamma} (\Gamma1 @ (x, b, c'[z' ::= V\text{-var } x]_{cv}) \#_{\Gamma} \Gamma)$
using *replace-in-g.simps GCons wfG-elim 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**
 $\Theta ; \mathcal{B} ; \Gamma \vdash \llbracket z' : b \mid c' \rrbracket \lesssim \llbracket z : b \mid c \rrbracket$ **and**
 $atom \ x \ \# \ (z, z', c, c')$
shows $\Theta ; \Phi ; \mathcal{B} ; \Gamma' @ (x, b, c'[z' ::= V\text{-var } x]_{cv}) \#_{\Gamma} \Gamma ; \Delta \vdash s \Leftarrow \tau$
proof –

have $wf : wfG \ \Theta \ \mathcal{B} \ (\Gamma' @ ((x, b, c[z ::= V\text{-var } x]_{cv}) \#_{\Gamma} \Gamma)) \ \text{using } check\text{-s-wf} \ assms \ \text{by meson}$
hence $* : x \notin fst \ ' \ toSet \ \Gamma' \ \text{using } wfG\text{-inside-fresh} \ \text{by force}$
have $wfG \ \Theta \ \mathcal{B} \ ((x, b, c[z ::= V\text{-var } x]_{cv}) \#_{\Gamma} \Gamma) \ \text{using } wf \ wfG\text{-suffix} \ \text{by metis}$
hence $xfg : atom \ x \ \# \ \Gamma \ \text{using } wfG\text{-elim} \ \text{by metis}$
have $x \neq z' \ \text{using } assms \ fresh\text{-at-base} \ fresh\text{-prod4} \ \text{by metis}$
hence $a2 : atom \ x \ \# \ c' \ \text{using } assms \ fresh\text{-prod4} \ \text{by metis}$

have $atom \ x \ \# \ (z', c', z, c, \Gamma) \ \text{proof} \ -$

have $x \neq z \ \text{using } assms \ \text{using } assms \ fresh\text{-at-base} \ fresh\text{-prod4} \ \text{by metis}$
hence $a1 : atom \ x \ \# \ c \ \text{using } assms \ subtype\text{-wf} \ subtype\text{-wf} \ assms \ wfT\text{-fresh-c} \ xfg \ \text{by meson}$
thus *?thesis using a1 a2 (atom x # (z, z', c, c')) fresh-prod4 fresh-Pair xfg by simp*
qed

hence $wc1 : \Theta ; \mathcal{B} ; (x, b, c'[z' ::= V\text{-var } x]_v) \#_{\Gamma} \Gamma \models c[z ::= V\text{-var } x]_v$
using *subtype-valid assms fresh-prodN by metis*

have $vld : \Theta ; \mathcal{B} ; (\Gamma' @ (x, b, c'[z' ::= V\text{-var } x]_{cv}) \#_{\Gamma} \Gamma) \models c[z ::= V\text{-var } x]_{cv} \ \text{proof} \ -$

```

have toSet ((x, b, c'[z'::=V-var x]cv) #Γ Γ) ⊆ toSet (Γ'@ (x, b, c'[z'::=V-var x]cv) #Γ Γ) by auto
moreover have wfG Θ B (Γ'@ (x, b, c'[z'::=V-var x]cv) #Γ Γ) proof –
  have *:wfT Θ B Γ (⊥ z' : b | c' ⊥) using subtype-wf-assms by meson
  moreover have atom x ⊥ (c', Γ) using xfg a2 by simp
  ultimately have wfG Θ B ((x, b, c'[z'::=V-var x]cv) #Γ Γ) 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; B; Γ' @ (x, b, c'[z'::=V-var x]cv) #Γ Γ ⊢wf c[z::=V-var x]cv using valid.simps by
auto
have wbc1: Θ; B; (x, b, c'[z'::=V-var x]cv) #Γ Γ ⊢wf c[z::=V-var x]cv using wc1 valid.simps
subst-defs by auto
have wsX (Γ'@ ((x, b, c[z::=V-var x]cv) #Γ Γ)) [(x, c'[z'::=V-var x]cv)] proof
  show wsX (Γ' @ (x, b, c[z::=V-var x]cv) #Γ Γ) [] using wsX-NilI by auto
  show atom x ∈ atom-dom (Γ' @ (x, b, c[z::=V-var x]cv) #Γ Γ) by simp
  show x ∉ fst ' set [] by auto
qed
moreover have replace-in-g-subtyped Θ B (Γ'@ ((x, b, c[z::=V-var x]cv) #Γ Γ)) [(x, c'[z'::=V-var x]cv)]
(Γ'@ (x, b, c'[z'::=V-var x]cv) #Γ Γ) proof
  show Some (b, c[z::=V-var x]cv) = lookup (Γ' @ (x, b, c[z::=V-var x]cv) #Γ Γ) x using lookup-inside*
by auto
  show Θ; B; replace-in-g (Γ' @ (x, b, c[z::=V-var x]cv) #Γ Γ) x (c'[z'::=V-var x]cv) ⊢ c[z::=V-var
x]cv using vld replace-in-g-split wf by metis
  show replace-in-g-subtyped Θ B (replace-in-g (Γ' @ (x, b, c[z::=V-var x]cv) #Γ Γ) x (c'[z'::=V-var
x]cv)) [] (Γ' @ (x, b, c'[z'::=V-var x]cv) #Γ Γ)
    using replace-in-g-subtyped-empty wf by presburger
  show x ∉ fst ' set [] by auto
  show Θ; B; Γ' @ (x, b, c[z::=V-var x]cv) #Γ Γ ⊢wf c'[z'::=V-var x]cv
proof(rule wf-weakening)
    show Θ; B; (x, b, c[z::=V-var x]cv) #Γ Γ ⊢wf c'[z'::=[ x ]cv] using wfC-cons-switch[OF
wbc1] wf-weakening(6) check-s-wf-assms toSet.simps by metis
    show Θ; B ⊢wf Γ' @ (x, b, c[z::=[ x ]cv] #Γ Γ) using wfC-cons-switch[OF wbc1] wf-weakening(6)
check-s-wf-assms toSet.simps by metis
    show ⟨toSet ((x, b, c[z::=V-var x]cv) #Γ Γ) ⊆ toSet (Γ' @ (x, b, c[z::=[ x ]cv] #Γ Γ)⟩ using
append-g.simps toSet.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 Proof Methods

method *subst-mth* = (*metis subst-g-inside infer-e-wf infer-v-wf infer-v-wf*)

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*))

14.2 Misc

lemma *subst-top-eq*:

$\llbracket z : b \mid TRUE \rrbracket = \llbracket z : b \mid TRUE \rrbracket[x::=v]_{\tau v}$

proof –

obtain $z'::x$ **and** c' **where** *zeq*: $\llbracket z : b \mid TRUE \rrbracket = \llbracket z' : b \mid c' \rrbracket \wedge \text{atom } z' \nmid (x,v)$ **using** *obtain-fresh-z2 b-of.simps by metis*

hence $\llbracket z' : b \mid TRUE \rrbracket[x::=v]_{\tau v} = \llbracket z' : b \mid TRUE \rrbracket$ **using** *subst-tv.simps subst-cv.simps by metis*

moreover have $c' = C\text{-true}$ **using** $\tau.\text{eq-iff Abs1-eq-iff}(\exists) c.\text{fresh flip-fresh-fresh}$ **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 # (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 $c\text{-of } \llbracket z' : b' \mid c' \rrbracket[x::=v]_{\tau v} xa = c\text{-of } \llbracket z' : b' \mid c'[x::=v]_{cv} \rrbracket xa$
using *subst-tv.simps fresh-Pair* by *metis*
also have $\dots = c'[x::=v]_{cv} [z'::=V\text{-var } xa]_{cv}$ using *c-of.simps T-refined-type* by *metis*
also have $\dots = c' [z'::=V\text{-var } xa]_{cv}[x::=v]_{cv}$
using *subst-cv-commute-full[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

14.3 Context

lemma *subst-lookup*:

assumes *Some (b,c) = lookup (Γ'@((x,b₁,c₁)#_ΓΓ)) y* and $x \neq y$ and $wfG \Theta \mathcal{B} (\Gamma'@((x,b_1,c_1)\#_{\Gamma}\Gamma))$
shows $\exists d. \text{Some } (b,d) = \text{lookup } ((\Gamma'[x::=v]_{\Gamma v})@_{\Gamma}) y$
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 Γ1*)
show *?case* proof(*cases x1 = x*)
case *True*
hence *atom x # (Γ1 @ (x, b₁, c₁) #_Γ Γ)* using *GCons wfG-elim(2)*
append-g.simps by *metis*
moreover have *atom x ∈ atom-dom (Γ1 @ (x, b₁, c₁) #_Γ Γ)* by *simp*
ultimately show *?thesis*
using *forget-subst-gv not-GCons-self2 subst-gv.simps append-g.simps*
by (*metis GCons.prem(3) True wfG-cons-fresh2*)
next
case *False*
hence $((x1,b1,c1) \#_{\Gamma} \Gamma1)[x::=v]_{\Gamma v} = (x1,b1,c1[x::=v]_{cv}) \#_{\Gamma} \Gamma1[x::=v]_{\Gamma v}$ 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) #_Γ Γ1)[x::=v]_{\Gamma v} = (x1, b1, c1[x::=v]_{cv}) #_Γ Γ1[x::=v]_{\Gamma v}*) *append-g.simps fst-conv option.inject*)
next
case *False*
then show *?thesis* using *GCons* using *lookup.simps*
using $((x1, b1, c1) \#_{\Gamma} \Gamma1)[x::=v]_{\Gamma v} = (x1, b1, c1[x::=v]_{cv}) \#_{\Gamma} \Gamma1[x::=v]_{\Gamma v}$ *append-g.simps*
Γ.distinct Γ.inject wfG.simps wfG-elim by *metis*

qed

qed
qed

14.4 Satisfiability

lemma *is-satis-g-i-upd2*:

assumes *eval-v i v s* **and** *is-satis ((i (x ↦ s))) c0* **and** *atom x # G* **and** *wfG Θ B (G3@((x,b,c0)#Γ G))*
and *wfV Θ B G v b* **and** *wfI Θ (G3[x::=v]_{Γv}@G) i*
and *is-satis-g i (G3[x::=v]_{Γv}@G)*
shows *is-satis-g (i (x ↦ s)) (G3@((x,b,c0)#Γ G))*
using *assms* **proof**(*induct G3 rule: Γ-induct*)
case *GNil*
hence *is-satis-g (i(x ↦ 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 ≠ x'* **using** *wfG-cons-append* **by** *metis*
hence *is-satis-g i (((x', b', c'[x::=v]_{cv}) #_Γ (Γ'[x::=v]_{Γv}) @ G))* **using** *subst-gv.simps GCons* **by** *auto*
hence **:is-satis i c'[x::=v]_{cv} ∧ is-satis-g i ((Γ'[x::=v]_{Γv}) @ G)* **using** *subst-gv.simps* **by** *auto*

have *is-satis-g (i(x ↦ s)) ((x', b', c') #_Γ (Γ' @ (x, b, c0) #_Γ G))* **proof**(*subst is-satis-g.simps,rule*)
show *is-satis (i(x ↦ s)) c'* **proof**(*subst subst-c-satis-full[symmetric]*)
show *⟨eval-v i v s⟩* **using** *GCons* **by** *auto*
show *⟨Θ ; B ; ((x', b', c') #_Γ Γ') @ (x, b, c0) #_Γ G ⊢_{wf} c'⟩* **using** *GCons wfC-refl* **by** *auto*
show *⟨wfI Θ (((x', b', c') #_Γ Γ')[x::=v]_{Γv}) @ G i⟩* **using** *GCons* **by** *auto*
show *⟨Θ ; 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 ↦ s)) (Γ' @ (x, b, c0) #_Γ G)* **proof**(*rule GCons(1)*)

show *⟨eval-v i v s⟩* **using** *GCons* **by** *auto*
show *⟨is-satis (i(x ↦ s)) c0⟩* **using** *GCons* **by** *metis*
show *⟨atom x # G⟩* **using** *GCons* **by** *auto*
show *⟨Θ ; B ⊢_{wf} Γ' @ (x, b, c0) #_Γ G⟩* **using** *GCons wfG-elim append-g.simps* **by** *metis*
show *⟨is-satis-g i (Γ'[x::=v]_{Γv} @ G)⟩* **using** *** **by** *auto*
show *wfI Θ (Γ'[x::=v]_{Γv} @ G) i* **using** *GCons wfI-def subst-g-assoc-cons ⟨x ≠ x'⟩* **by** *auto*
show *Θ ; B ; G ⊢_{wf} v : b* **using** *GCons* **by** *auto*

qed

qed

moreover have *((x', b', c') #_Γ Γ' @ (x, b, c0) #_Γ G) = (((x', b', c') #_Γ Γ') @ (x, b, c0) #_Γ G)*
by *auto*

ultimately show *?case* **using** *GCons* **by** *metis*

qed

lemma *is-satis-eq*:

assumes *wfI Θ G i* **and** *wfCE Θ 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

14.5 Validity

lemma *subst-self-valid*:

fixes $v::v$

assumes $\Theta ; \mathcal{B} ; G \vdash v \Rightarrow \llbracket z : b \mid c \rrbracket$ **and** $\text{atom } z \nVdash v$

shows $\Theta ; \mathcal{B} ; G \models c[z::=v]_{cv}$

proof –

have $c = (CE\text{-val } (V\text{-var } z) == CE\text{-val } v)$ **using** *infer-v-form2 assms* **by** *presburger*

hence $c[z::=v]_{cv} = (CE\text{-val } (V\text{-var } z) == CE\text{-val } v)[z::=v]_{cv}$ **by** *auto*

also have $\dots = (((CE\text{-val } (V\text{-var } z))[z::=v]_{cev}) == ((CE\text{-val } v)[z::=v]_{cev}))$ **by** *fastforce*

also have $\dots = ((CE\text{-val } v) == ((CE\text{-val } v)[z::=v]_{cev}))$ **using** *subst-cev.simps subst-vv.simps* **by**

presburger

also have $\dots = (CE\text{-val } v == CE\text{-val } v)$ **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 $\Theta \vdash_{wf} ([\Phi]) \quad \wedge \quad \Theta ; \mathcal{B} ; G \vdash_{wf} []_{\Delta}$ **using** *wfD-emptyI wfPhi-emptyI infer-v-wf*

assms **by** *auto*

ultimately show *?thesis* **using** $* \text{wfCE-valI wfC-eqI}$ **by** *metis*

qed

show $\forall i. \text{wfI } \Theta \ G \ i \wedge \text{is-satis-g } i \ G \longrightarrow \text{is-satis } i \ c[z::=v]_{cv}$ **proof**(*rule,rule*)

fix i

assume $\langle \text{wfI } \Theta \ G \ i \wedge \text{is-satis-g } i \ G \rangle$

thus $\langle \text{is-satis } i \ c[z::=v]_{cv} \rangle$ **using** $** \text{is-satis-eq}$ **by** *auto*

qed

qed

qed

lemma *subst-valid-simple*:

fixes $v::v$

assumes $\Theta ; \mathcal{B} ; G \vdash v \Rightarrow \llbracket z0 : b \mid c0 \rrbracket$ **and**

$\text{atom } z0 \nVdash c$ **and** $\text{atom } z0 \nVdash v$

$\Theta ; \mathcal{B} ; (z0, b, c0) \#_{\Gamma} G \models c[z::=V\text{-var } z0]_{cv}$

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 $\text{atom } z0 \nVdash G$ **using** *assms valid-wf-all* **by** *meson*

moreover have $\text{wfV } \Theta \ \mathcal{B} \ G \ v \ b$ **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 $\text{wfI } \Theta \ (G'[x::=v]_{\Gamma v} @ G) \ i$ **and** $\text{wfG } \Theta \ \mathcal{B} \ (G' @ (x, b, c[z::=[x]^v]_{cv}) \#_{\Gamma} G)$ **and** $\text{eval-v } i$

$v \ sv$ **and** $\text{wfRCV } \Theta \ sv \ b$

shows $\text{wfI } \Theta \ (G' @ (x, b, c[z::=[x]^v]_{cv}) \#_{\Gamma} G) \ (i \ (x \mapsto sv))$

proof –

```

{
  fix  $xa::x$  and  $ba::b$  and  $ca::c$ 
  assume  $as: (xa, ba, ca) \in toSet ((G' @ ((x, b, c[z::=[x]^v]_{cv}) \#_{\Gamma} G)))$ 
  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  $(xa, ba, ca') \in toSet (G'[x::=v]_{\Gamma v} @ G)$  using  $wfG-member-subst2\ assms$ 
      as by metis
      then obtain  $s$  where  $Some\ s = i\ xa \wedge wfRCV\ \Theta\ s\ ba$  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  $\Gamma::\Gamma$ 
assumes  $\Theta ; \mathcal{B} ; \Gamma \models c[z::=v]_{cv}$  and  $\Theta ; \mathcal{B} ; \Gamma \vdash_{wf} v : b$  and
   $\Theta ; \mathcal{B} \vdash_{wf} \Gamma$  and  $atom\ x \nmid c$  and  $atom\ x \nmid \Gamma$  and
   $\Theta ; \mathcal{B} \vdash_{wf} (\Gamma' @ (x, b, c[z::=[x]^v]_{cv}) \#_{\Gamma} \Gamma)$  and
   $\Theta ; \mathcal{B} ; \Gamma' @ (x, b, c[z::=[x]^v]_{cv}) \#_{\Gamma} \Gamma \models c'$  (is  $\Theta ; \mathcal{B} ; ?G \models c'$ )
shows  $\Theta ; \mathcal{B} ; \Gamma'[x::=v]_{\Gamma v} @ \Gamma \models c'[x::=v]_{cv}$ 

```

proof –

```

have  $*: wfC\ \Theta\ \mathcal{B}\ (\Gamma' @ (x, b, c[z::=[x]^v]_{cv}) \#_{\Gamma} \Gamma)\ c'$  using  $valid.simps\ assms$  by metis
hence  $wfC\ \Theta\ \mathcal{B}\ (\Gamma'[x::=v]_{\Gamma v} @ \Gamma)\ (c'[x::=v]_{cv})$  using  $wf-subst(2)[OF\ *]\ b-of.simps\ assms$ 
subst-g-inside wfC-wf by metis
moreover have  $\forall i. wfI\ \Theta\ (\Gamma'[x::=v]_{\Gamma v} @ \Gamma)\ i \wedge is-satis-g\ i\ (\Gamma'[x::=v]_{\Gamma v} @ \Gamma) \longrightarrow is-satis\ i\ (c'[x::=v]_{cv})$ 

```

proof(*rule, rule*)

```

fix  $i$ 
assume  $as: wfI\ \Theta\ (\Gamma'[x::=v]_{\Gamma v} @ \Gamma)\ i \wedge is-satis-g\ i\ (\Gamma'[x::=v]_{\Gamma v} @ \Gamma)$ 
thm valid.simps
hence  $wfI: wfI\ \Theta\ \Gamma\ i$  using  $wfI-suffix\ infer-v-wf\ assms$  by metis
then obtain  $s$  where  $s: eval-v\ i\ v\ s$  and  $b: wfRCV\ \Theta\ s\ b$  using  $eval-v-exist\ infer-v-v-wf\ b-of.simps$ 
assms by metis
thm is-satis-g-i-upd2
have  $is1: is-satis-g\ (i(x \mapsto s))\ (\Gamma' @ (x, b, c[z::=[x]^v]_{cv}) \#_{\Gamma} \Gamma)$  proof(rule is-satis-g-i-upd2)
  show  $is-satis\ (i(x \mapsto s))\ (c[z::=[x]^v]_{cv})$  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  $\Theta ; \mathcal{B} ; (x, b, c[z::=[x]^v]_{cv}) \#_{\Gamma} \Gamma \vdash_{wf} c[z::=[x]^v]_{cv}$  using  $wfC-refl\ wfG-suffix$ 

```

assms by metis

moreover have $\Theta ; \mathcal{B} ; \Gamma \vdash_{wf} v : b$ **using** *assms infer-v-v-wf b-of.simps by metis*

ultimately show *?thesis* **using** *subst-c-satis[OF s , of $\Theta \mathcal{B} x b \ c[z::=[x]^v]_{cv} \ \Gamma \ c[z::=[x]^v]_{cv}$]*

wfG by auto

qed

show *atom x # Γ* **using** *assms by metis*

show *wfG $\Theta \mathcal{B} (\Gamma' @ (x, b, c[z::=[x]^v]_{cv}) \#_{\Gamma} \Gamma)$* **using** *valid-wf-all assms by metis*

show $\Theta ; \mathcal{B} ; \Gamma \vdash_{wf} v : b$ **using** *assms infer-v-v-wf by force*

show $i \llbracket v \rrbracket \sim s$ **using** *s by auto*

show $\Theta ; \Gamma'[x::=v]_{\Gamma v} @ \Gamma \vdash i$ **using** *as by auto*

show $i \models \Gamma'[x::=v]_{\Gamma v} @ \Gamma$ **using** *as by auto*

qed

hence *is-satis (i(x \mapsto s)) c' proof -*

have *wfI $\Theta (\Gamma' @ (x, b, c[z::=[x]^v]_{cv}) \#_{\Gamma} \Gamma) (i(x \mapsto s))$*

using *wfI-subst1[of $\Theta \Gamma' x v \ \Gamma i \mathcal{B} b c z \ s$] 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 $\Theta ; \mathcal{B} ; G \vdash v \Rightarrow \{ z0 : b \mid c0 \}$ **and** *atom x # c and atom x # G and wfG $\Theta \mathcal{B}$*
($G' @ (x, b, c[z::=[x]^v]_{cv}) \#_{\Gamma} G$) and atom z0 # v

$\Theta ; \mathcal{B} ; (z0, b, c0) \#_{\Gamma} G \models c[z::=V-var z0]_{cv}$ **and** *atom z0 # c and*

$\Theta ; \mathcal{B} ; G' @ (x, b, c[z::=[x]^v]_{cv}) \#_{\Gamma} G \models c'$ **(is $\Theta ; \mathcal{B} ; ?G \models c'$)**

shows $\Theta ; \mathcal{B} ; G'[x::=v]_{\Gamma v} @ G \models c'[x::=v]_{cv}$

proof -

have $\Theta ; \mathcal{B} ; G \models c[z::=v]_{cv}$

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 $\Theta \mathcal{B} G v b$ and wfG $\Theta \mathcal{B} G$*

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

14.6 Subtyping

lemma *subst-subtype:*

fixes $v::v$

assumes $\Theta ; \mathcal{B} ; \Gamma \vdash v \Rightarrow (\{ z0 : b \mid c0 \})$ **and**

$\Theta ; \mathcal{B} ; \Gamma \vdash (\{ z0 : b \mid c0 \}) \lesssim (\{ z : b \mid c \})$ **and**

$\Theta ; \mathcal{B} ; \Gamma' @ ((x, b, c[z::=[x]^v]_{cv}) \#_{\Gamma} \Gamma) \vdash (\{ z1 : b1 \mid c1 \}) \lesssim (\{ z2 : b1 \mid c2 \})$ **(is $\Theta ; \mathcal{B} ; ?G1 \vdash ?t1 \lesssim ?t2$) and**

atom z # (x, v) \wedge atom z0 # (c, x, v, z, Γ) \wedge atom z1 # (x, v) \wedge atom z2 # (x, v) and wsV $\Theta \mathcal{B} \Gamma v$

shows $\Theta ; \mathcal{B} ; \Gamma'[x::=v]_{\Gamma v} @ \Gamma \vdash \{ z1 : b1 \mid c1 \}[x::=v]_{\tau v} \lesssim \{ z2 : b1 \mid c2 \}[x::=v]_{\tau v}$

proof –

have $z2: \text{atom } z2 \# (x, v)$ **using** *assms* **by** *auto*
hence $x \neq z2$ **by** *auto*

obtain $xx::x$ **where** $xxf: \text{atom } xx \# (x, z1, c1, z2, c2, \Gamma' @ (x, b, c[z::=[x]^v]_{cv}) \#_{\Gamma} \Gamma, c1[x::=v]_{cv}, c2[x::=v]_{cv}, \Gamma'[x::=v]_{\Gamma_v} @ \Gamma,$
 $(\Theta, \mathcal{B}, \Gamma'[x::=v]_{\Gamma_v} @ \Gamma, z1, c1[x::=v]_{cv}, z2, c2[x::=v]_{cv})$ **(is** $\text{atom } xx \# ?tup$ **)**
using *obtain-fresh* **by** *blast*
hence $xxf2: \text{atom } xx \# (z1, c1, z2, c2, \Gamma' @ (x, b, c[z::=[x]^v]_{cv}) \#_{\Gamma} \Gamma)$ **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 $\Theta; \mathcal{B}; \Gamma \vdash v \Rightarrow \llbracket z0 : b \mid c0 \rrbracket$ **using** *assms* **by** *auto*

show $xf: \text{atom } x \# \Gamma$ **using** *subtype-g-wf wfG-inside-fresh-suffix* *assms* **by** *metis*

show $\text{atom } x \# c$ **proof** –

have $wfT \ \Theta \ \mathcal{B} \ \Gamma \ (\llbracket z : b \mid c \rrbracket)$ **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** $xf \ wfT\text{-fresh-}c$ *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 $\ast: \langle \Theta; \mathcal{B} \vdash_{wf} \Gamma' @ (x, b, c[z::=[x]^v]_{cv}) \#_{\Gamma} \Gamma \rangle$ **using** *subtype-g-wf* *assms* **by** *metis*

show $\langle \text{atom } xx \# \Gamma' @ (x, b, c[z::=[x]^v]_{cv}) \#_{\Gamma} \Gamma \rangle$ **using** $xxf \ \text{fresh-prod9}$ **by** *metis*

show $\langle \Theta; \mathcal{B} \vdash_{wf} b1 \rangle$ **using** *subtype-elimis[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 $\llbracket z1 : b1 \mid c1 \rrbracket = \llbracket xx : b1 \mid c1[z1::=V\text{-var } xx]_{cv} \rrbracket$ **using** $xxf \ \text{fresh-prod9}$ *type-eq-subst*
 $xxf2 \ \text{fresh-prodN}$ **by** *metis*

thus $\Theta; \mathcal{B}; \Gamma' @ (x, b, c[z::=[x]^v]_{cv}) \#_{\Gamma} \Gamma \vdash_{wf} \llbracket xx : b1 \mid c1[z1::=V\text{-var } xx]_{cv} \rrbracket$ **using** *subtype-wfT[OF assms(3)]* **by** *metis*

show $\text{atom } xx \# \Gamma' @ (x, b, c[z::=[x]^v]_{cv}) \#_{\Gamma} \Gamma$ **using** $xxf \ \text{fresh-prod9}$ **by** *metis*

qed

qed

show $\text{atom } z0 \# v$ **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 $(z0, c0, \Gamma, c, z)$, 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 $\text{atom } z0 \# c$ **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)* $xxf \ xxf2 \ \text{fresh-prodN}$ *append-g.simps* *subst-defs* **by** *metis*

qed

have $xfw1: \text{atom } z1 \# v \wedge \text{atom } x \# [xx]^v \wedge x \neq z1$

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 # v ∧ atom x # [xx]^v ∧ x ≠ 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 ⊖ B (Γ'[x::=v]_{Γv} ⊗ Γ) (⊥ z1 : b1 | c1[x::=v]_{cv} ⊥) proof –*
have *wfT ⊖ B (Γ'[x::=v]_{Γv} ⊗ Γ) (⊥ z1 : b1 | c1 ⊥)[x::=v]_{τv}*
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 # (x,v) using assms by auto*
ultimately show *?thesis using subst-tv.simps by auto*
qed
moreover have *wf2: wfT ⊖ B (Γ'[x::=v]_{Γv} ⊗ Γ) (⊥ z2 : b1 | c2[x::=v]_{cv} ⊥) proof –*
have *wfT ⊖ B (Γ'[x::=v]_{Γv} ⊗ Γ) (⊥ z2 : b1 | c2 ⊥)[x::=v]_{τv} 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 # (x,v) using assms by auto*
ultimately show *?thesis using subst-tv.simps by auto*
qed
moreover have *Θ ; B ; (xx, b1, c1[x::=v]_{cv}[z1::=V-var xx]_{cv}) #_Γ (Γ'[x::=v]_{Γv} ⊗ Γ) ⊨ (c2[x::=v]_{cv})[z2::=V-var*
xx]_{cv} proof –
have *xx ≠ x using xxf fresh-Pair fresh-at-base by fast*
hence *((xx, b1, subst-cv c1 z1 (V-var xx)) #_Γ Γ')[x::=v]_{Γv} = (xx, b1, (subst-cv c1 z1 (V-var xx)*
)[x::=v]_{cv}) #_Γ (Γ'[x::=v]_{Γv})
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-full*
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-full*
xfw2 by metis
ultimately show *?thesis using vd1 append-g.simps by metis*
qed
moreover have *atom xx # (Θ , B , Γ'[x::=v]_{Γv} ⊗ Γ, z1 , c1[x::=v]_{cv} , z2 , c2[x::=v]_{cv})*
using *xxf fresh-prodN by metis*
ultimately have *Θ ; B ; Γ'[x::=v]_{Γv} ⊗ Γ ⊢ ⊥ 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 ; B ; Γ ⊢ v ⇒ τ and*
Θ ; B ; Γ ⊢ τ ≲ (⊥ z : b | c ⊥)
Θ ; B ; Γ' ⊗ ((x,b,c[z::=[x]^v]_{cv}) #_Γ Γ) ⊢ τ1 ≲ τ2 and
atom z # (x,v)
shows *Θ ; B ; Γ'[x::=v]_{Γv} ⊗ Γ ⊢ τ1[x::=v]_{τv} ≲ τ2[x::=v]_{τv}*
proof –
obtain *z0 and b0 and c0 where zbc0: τ=(⊥ z0 : b0 | c0 ⊥) ∧ atom z0 # (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 # (x,v)*

using obtain-fresh-z by metis
 obtain z2 and b2 and c2 where zbc2: $\tau 2 = (\llbracket z2 : b2 \mid c2 \rrbracket) \wedge \text{atom } z2 \# (x, v)$
 using obtain-fresh-z by metis

 have b0=b using subtype-eq-base zbc0 assms by blast

 hence vinf: $\Theta ; \mathcal{B} ; \Gamma \vdash v \Rightarrow \llbracket z0 : b \mid c0 \rrbracket$ using assms zbc0 by blast
 have vsub: $\Theta ; \mathcal{B} ; \Gamma \vdash \llbracket z0 : b \mid c0 \rrbracket \lesssim \llbracket z : b \mid c \rrbracket$ using assms zbc0 $\langle b0=b \rangle$ by blast
 have beq:b1=b2 using subtype-eq-base
 using zbc1 zbc2 assms by blast
 have $\Theta ; \mathcal{B} ; \Gamma[x::=v]_{\Gamma v} @ \Gamma \vdash \llbracket z1 : b1 \mid c1 \rrbracket[x::=v]_{\tau v} \lesssim \llbracket z2 : b1 \mid c2 \rrbracket[x::=v]_{\tau v}$
 proof(rule subst-subtype[OF vinf vsub])
 show $\Theta ; \mathcal{B} ; \Gamma @ ((x, b, c[z::=[x]^v]_{cv}) \# \Gamma) \vdash \llbracket z1 : b1 \mid c1 \rrbracket \lesssim \llbracket z2 : b1 \mid c2 \rrbracket$
 using beq assms zbc1 zbc2 by auto
 show $\text{atom } z \# (x, v) \wedge \text{atom } z0 \# (c, x, v, z, \Gamma) \wedge \text{atom } z1 \# (x, v) \wedge \text{atom } z2 \# (x, v)$
 using zbc0 zbc1 zbc2 assms by blast
 show $\text{wfV } \Theta \mathcal{B} \Gamma v (b\text{-of } \tau)$ using infer-v-wf assms by simp
 qed

 thus ?thesis using zbc1 zbc2 $\langle b1=b2 \rangle$ assms by blast
 qed

 lemma subtype-if1:
 fixes v::v
 assumes $P ; \mathcal{B} ; \Gamma \vdash t1 \lesssim t2$ and $\text{wfV } P \mathcal{B} \Gamma v$ (base-for-lit l) and
 $\text{atom } z1 \# v$ and $\text{atom } z2 \# v$ and $\text{atom } z1 \# t1$ and $\text{atom } z2 \# t2$ and $\text{atom } z1 \# \Gamma$ and $\text{atom } z2 \# \Gamma$
 shows $P ; \mathcal{B} ; \Gamma \vdash \llbracket z1 : b\text{-of } t1 \mid \text{CE-val } v == \text{CE-val } (V\text{-lit } l) \text{ IMP } (c\text{-of } t1 \ z1) \rrbracket \lesssim \llbracket z2 : b\text{-of } t2 \mid \text{CE-val } v == \text{CE-val } (V\text{-lit } l) \text{ IMP } (c\text{-of } t2 \ z2) \rrbracket$
 proof -
 obtain z1' where $t1 : t1 = \llbracket z1' : b\text{-of } t1 \mid c\text{-of } t1 \ z1' \rrbracket \wedge \text{atom } z1' \# (z1, \Gamma, t1)$ using obtain-fresh-z-c-of by metis
 obtain z2' where $t2 : t2 = \llbracket z2' : b\text{-of } t2 \mid c\text{-of } t2 \ z2' \rrbracket \wedge \text{atom } z2' \# (z2, t2)$ using obtain-fresh-z-c-of by metis
 have $\text{beq}: b\text{-of } t1 = b\text{-of } t2$ using subtype-eq-base2 assms by auto

 have c1: $(c\text{-of } t1 \ z1')[z1'::=[z1]^v]_{cv} = c\text{-of } t1 \ z1$ using c-of-switch t1 assms by simp
 have c2: $(c\text{-of } t2 \ z2')[z2'::=[z2]^v]_{cv} = c\text{-of } t2 \ z2$ using c-of-switch t2 assms by simp

 have $P ; \mathcal{B} ; \Gamma \vdash \llbracket z1 : b\text{-of } t1 \mid [v]^{ce} == [[l]^v]^{ce} \text{ IMP } (c\text{-of } t1 \ z1')[z1'::=[z1]^v]_v \rrbracket \lesssim \llbracket z2 : b\text{-of } t1 \mid [v]^{ce} == [[l]^v]^{ce} \text{ IMP } (c\text{-of } t2 \ z2')[z2'::=[z2]^v]_v \rrbracket$
 proof(rule subtype-if)
 show $\langle P ; \mathcal{B} ; \Gamma \vdash \llbracket z1' : b\text{-of } t1 \mid c\text{-of } t1 \ z1' \rrbracket \lesssim \llbracket z2' : b\text{-of } t1 \mid c\text{-of } t2 \ z2' \rrbracket \rangle$ using t1 t2 assms beq by auto
 show $\langle P ; \mathcal{B} ; \Gamma \vdash_{wf} \llbracket z1 : b\text{-of } t1 \mid [v]^{ce} == [[l]^v]^{ce} \text{ IMP } (c\text{-of } t1 \ z1')[z1'::=[z1]^v]_v \rrbracket \rangle$
 using wfT-wfT-if-rev assms subtype-wfT c1 subst-defs by metis
 show $\langle P ; \mathcal{B} ; \Gamma \vdash_{wf} \llbracket z2 : b\text{-of } t1 \mid [v]^{ce} == [[l]^v]^{ce} \text{ IMP } (c\text{-of } t2 \ z2')[z2'::=[z2]^v]_v \rrbracket \rangle$
 using wfT-wfT-if-rev assms subtype-wfT c2 subst-defs beq by metis
 show $\langle \text{atom } z1 \# v \rangle$ using assms by auto
 show $\langle \text{atom } z1' \# \Gamma \rangle$ using t1 by auto
 show $\langle \text{atom } z1 \# c\text{-of } t1 \ z1' \rangle$ using t1 assms c-of-fresh by force
 show $\langle \text{atom } z2 \# c\text{-of } t2 \ z2' \rangle$ using t2 assms c-of-fresh by force

show $\langle \text{atom } z2 \# v \rangle$ using *assms* by *auto*
 qed
 then show *?thesis* using *t1 t2 assms c1 c2 beq subst-defs* by *metis*
 qed

14.7 Values

lemma *subst-infer-aux*:

fixes $\tau_1 :: \tau$ and $v' :: v$
 assumes $\Theta ; \mathcal{B} ; \Gamma \vdash v'[x ::= v]_{vv} \Rightarrow \tau_1$ and $\Theta ; \mathcal{B} ; \Gamma' \vdash v' \Rightarrow \tau_2$ and *b-of* $\tau_1 = \text{b-of } \tau_2$
 shows $\tau_1 = (\tau_2[x ::= v]_{\tau v})$
 proof –
 obtain *z1* and *b1* where *zb1*: $\tau_1 = (\llbracket z1 : b1 \mid C\text{-eq } (CE\text{-val } (V\text{-var } z1)) (CE\text{-val } (v'[x ::= v]_{vv})) \rrbracket)$
 $\wedge \text{atom } z1 \# ((CE\text{-val } (v'[x ::= v]_{vv}), CE\text{-val } v), v'[x ::= v]_{vv})$
 using *infer-v-form-fresh* [*OF assms*(1)] by *fastforce*
 obtain *z2* and *b2* where *zb2*: $\tau_2 = (\llbracket z2 : b2 \mid C\text{-eq } (CE\text{-val } (V\text{-var } z2)) (CE\text{-val } v') \rrbracket) \wedge \text{atom } z2 \# ((CE\text{-val } (v'[x ::= v]_{vv}), CE\text{-val } v, x, v), v')$
 using *infer-v-form-fresh* [*OF assms*(2)] by *fastforce*
 have *beq*: $b1 = b2$ using *assms zb1 zb2* by *simp*

 hence $(\llbracket z2 : b2 \mid C\text{-eq } (CE\text{-val } (V\text{-var } z2)) (CE\text{-val } v') \rrbracket)[x ::= v]_{\tau v} = (\llbracket z2 : b2 \mid C\text{-eq } (CE\text{-val } (V\text{-var } z2)) (CE\text{-val } (v'[x ::= v]_{vv})) \rrbracket)$
 using *subst-tv.simps subst-cv.simps subst-ev.simps forget-subst-vv* [*of x V-var z2*] *zb2* by *force*
 also have $\dots = (\llbracket z1 : b1 \mid C\text{-eq } (CE\text{-val } (V\text{-var } z1)) (CE\text{-val } (v'[x ::= v]_{vv})) \rrbracket)$
 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* $(\tau[x ::= v]_{\tau v}) = \text{b-of } \tau$
 proof –
 obtain *z* and *b* and *c* where $\tau = \llbracket z : b \mid c \rrbracket \wedge \text{atom } z \# (x, v)$
 using *has-fresh-z* by *blast*
 thus *?thesis* using *subst-tv.simps* by *simp*
 qed

lemma *fresh-g-fresh-v*:

fixes $x :: x$
 assumes $\text{atom } x \# \Gamma$ and *wfV* $\Theta \mathcal{B} \Gamma v b$
 shows $\text{atom } x \# v$
 using *assms wfV-suppl wfX-wfY wfG-atoms-suppl-eq fresh-def*
 by (*metis wfV-x-fresh*)

lemma *infer-v-fresh-g-fresh-v*:

fixes $x :: x$ and $\Gamma :: \Gamma$ and $v :: v$
 assumes $\text{atom } x \# \Gamma' @ \Gamma$ and $\Theta ; \mathcal{B} ; \Gamma \vdash v \Rightarrow \tau$
 shows $\text{atom } x \# v$
 proof –
 have $\text{atom } x \# \Gamma$ using *fresh-suffix assms* by *auto*
 moreover have *wfV* $\Theta \mathcal{B} \Gamma v (\text{b-of } \tau)$ 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** $\Gamma::\Gamma$

assumes $\text{atom } xa \# \Gamma' @ ((x, b, c[z::=[x]^v]_{cv}) \#_{\Gamma} \Gamma)$ **and** $\Theta ; \mathcal{B} ; \Gamma \vdash v \Rightarrow \tau$

shows $\text{atom } xa \# (x, v)$

proof –

have $\text{atom } xa \# x$ **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 $\text{atom } xa \# v$ **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 $\Theta ; \mathcal{B} \vdash_{wf} \Gamma' @ (x, b, c0[z0::=[x]^v]_{cv}) \#_{\Gamma} \Gamma$ **and** $\Theta ; \mathcal{B} ; \Gamma \vdash v \Rightarrow \tau$ **and** *b-of* $\tau = b$

shows $\Theta ; \mathcal{B} \vdash_{wf} \Gamma'[x::=v]_{\Gamma v} @ \Gamma$

using *wfG-subst-wfV infer-v-v-wf assms* **by** *auto*

lemma *fresh-subst-gv-inside*:

fixes $\Gamma::\Gamma$

assumes $\text{atom } z \# \Gamma' @ (x, b_1, c0[z0::=[x]^v]_{cv}) \#_{\Gamma} \Gamma$ **and** $\text{atom } z \# v$

shows $\text{atom } z \# \Gamma'[x::=v]_{\Gamma v} @ \Gamma$

unfolding *fresh-append-g* **using** *fresh-append-g assms fresh-subst-gv fresh-GCons* **by** *metis*

lemma *subst-t*:

fixes $x::x$ **and** $b::b$ **and** $xa::x$

assumes $\text{atom } z \# x$ **and** $\text{atom } z \# v$

shows $(\{ z : b \mid [[z]^v]^{ce} == [v'[x::=v]_{vv}]^{ce} \}) = (\{ z : b \mid [[z]^v]^{ce} == [v]^{ce} \} \{ [x::=v]_{\tau v} \})$

using *assms subst-vv.simps subst-tv.simps subst-cv.simps subst-cev.simps* **by** *auto*

lemma *infer-l-fresh*:

assumes $\vdash l \Rightarrow \tau$

shows $\text{atom } x \# \tau$

proof –

thm *infer-l-wf*

have $\square ; \{ \square \} \vdash_{wf} GNil$ **using** *wfG-nilI wfTh-emptyI* **by** *auto*

hence $\square ; \{ \square \} ; GNil \vdash_{wf} \tau$ **using** *assms infer-l-wf* **by** *auto*

thus *?thesis* **using** *fresh-def wfT-supp* **by** *force*

qed

lemma *subst-infer-v*:

fixes $v::v$ **and** $v'::v$

assumes $\Theta ; \mathcal{B} ; \Gamma' @ ((x, b_1, c0[z0::=[x]^v]_{cv}) \#_{\Gamma} \Gamma) \vdash v' \Rightarrow \tau_2$ **and**

$\Theta ; \mathcal{B} ; \Gamma \vdash v \Rightarrow \tau_1$ **and**

$\Theta ; \mathcal{B} ; \Gamma \vdash \tau_1 \lesssim (\{ z0 : b_1 \mid c0 \})$ **and** $\text{atom } z0 \# (x, v)$

shows $\Theta ; \mathcal{B} ; (\Gamma'[x::=v]_{\Gamma v}) @ \Gamma \vdash v'[x::=v]_{vv} \Rightarrow \tau_2[x::=v]_{\tau v}$

using *assms proof(nominal-induct \Gamma' @ ((x, b_1, c0[z0::=[x]^v]_{cv}) \#_{\Gamma} \Gamma) v' \tau_2 avoiding: x v rule: infer-v.strong-induct)*

case (*infer-v-varI* $\Theta \mathcal{B} b c xa z$)

```

have  $\Theta ; \mathcal{B} ; \Gamma[x::=v]_{\Gamma_v} @ \Gamma \vdash [xa]^v[x::=v]_{vv} \Rightarrow \{ z : b \mid [[z]^v]^{ce} == [[xa]^v[x::=v]_{vv}]^{ce} \}$ 
proof(cases  $x=xa$ )
  case True
    have  $\Theta ; \mathcal{B} ; \Gamma[x::=v]_{\Gamma_v} @ \Gamma \vdash v \Rightarrow \{ z : b \mid [[z]^v]^{ce} == [v]^{ce} \}$ 
    proof(rule infer-v-g-weakening)
      show  $\langle \Theta ; \mathcal{B} ; \Gamma \vdash v \Rightarrow \{ z : b \mid [[z]^v]^{ce} == [v]^{ce} \} \rangle$ 
      using infer-v-form infer-v-varI
      by (metis True lookup-inside-unique-b lookup-inside-wf ms-fresh-all(32) subtype-eq-base type-e-eq)
    show  $\langle toSet \Gamma \subseteq toSet (\Gamma[x::=v]_{\Gamma_v} @ \Gamma) \rangle$  by simp
    have  $\Theta ; \mathcal{B} ; \Gamma \vdash_{wf} v : b_1$  using infer-v-wf subtype-eq-base2 b-of.simps infer-v-varI by metis
    thus  $\langle \Theta ; \mathcal{B} \vdash_{wf} \Gamma[x::=v]_{\Gamma_v} @ \Gamma \rangle$ 
      using wfG-subst[OF infer-v-varI(3), of  $\Gamma' x b_1 c0[z0::=[x]^v]_{cv} \Gamma v$ ] subst-g-inside infer-v-varI
by metis
qed

  thus ?thesis using subst-vv.simps True by simp
next
  case False
    then obtain  $c'$  where  $c: Some(b, c') = lookup(\Gamma[x::=v]_{\Gamma_v} @ \Gamma) xa$  using lookup-subst2 infer-v-varI
by metis
    have  $\Theta ; \mathcal{B} ; \Gamma[x::=v]_{\Gamma_v} @ \Gamma \vdash [xa]^v \Rightarrow \{ z : b \mid [[z]^v]^{ce} == [[xa]^v]^{ce} \}$ 
    proof
      have  $\Theta ; \mathcal{B} ; \Gamma \vdash_{wf} v : b_1$  using infer-v-wf subtype-eq-base2 b-of.simps infer-v-varI by metis
      thus  $\Theta ; \mathcal{B} \vdash_{wf} \Gamma[x::=v]_{\Gamma_v} @ \Gamma$  using infer-v-varI
      using wfG-subst[OF infer-v-varI(3), of  $\Gamma' x b_1 c0[z0::=[x]^v]_{cv} \Gamma v$ ] subst-g-inside infer-v-varI
by metis
      show  $atom\ z \# xa$  using infer-v-varI by auto
      show  $Some(b, c') = lookup(\Gamma[x::=v]_{\Gamma_v} @ \Gamma) xa$  using c by auto
      show  $atom\ z \# (\Theta, \mathcal{B}, \Gamma[x::=v]_{\Gamma_v} @ \Gamma)$  by (fresh-mth add: infer-v-varI fresh-subst-gv-inside)
    qed
    then show ?thesis using subst-vv.simps False by simp
  qed
  thus ?case using subst-t fresh-prodN infer-v-varI by metis
next
  case (infer-v-litI  $\Theta \mathcal{B} l \tau$ )
  show ?case unfolding subst-vv.simps proof
    show  $\Theta ; \mathcal{B} \vdash_{wf} \Gamma[x::=v]_{\Gamma_v} @ \Gamma$  using wfG-subst-infer-v infer-v-litI subtype-eq-base2 b-of.simps
by metis
    have  $atom\ x \# \tau$  using infer-v-litI infer-l-fresh by metis
    thus  $\vdash l \Rightarrow \tau[x::=v]_{\tau_v}$  using infer-v-litI type-v-subst-fresh by simp
  qed
next
  case (infer-v-pairI  $z v1 v2 \Theta \mathcal{B} t1 t2$ )
  have  $\Theta ; \mathcal{B} ; \Gamma[x::=v]_{\Gamma_v} @$ 

$$\Gamma \vdash [v1[x::=v]_{vv}, v2[x::=v]_{vv}]^v \Rightarrow \{ z : [b-of\ t1[x::=v]_{\tau_v}, b-of\ t2[x::=v]_{\tau_v}]^b \mid [[z]^v]^{ce} == [[v1[x::=v]_{vv}, v2[x::=v]_{vv}]^v]^{ce} \}$$

  proof
    show  $\langle atom\ z \# (v1[x::=v]_{vv}, v2[x::=v]_{vv}) \rangle$  by (fresh-mth add: infer-v-pairI)
    show  $\langle atom\ z \# (\Theta, \mathcal{B}, \Gamma[x::=v]_{\Gamma_v} @ \Gamma) \rangle$  by (fresh-mth add: infer-v-pairI fresh-subst-gv-inside)
    show  $\langle \Theta ; \mathcal{B} ; \Gamma[x::=v]_{\Gamma_v} @ \Gamma \vdash v1[x::=v]_{vv} \Rightarrow t1[x::=v]_{\tau_v} \rangle$  using infer-v-pairI by metis
    show  $\langle \Theta ; \mathcal{B} ; \Gamma[x::=v]_{\Gamma_v} @ \Gamma \vdash v2[x::=v]_{vv} \Rightarrow t2[x::=v]_{\tau_v} \rangle$  using infer-v-pairI by metis
  qed

```

then show $?case$ using *subst-vv.simps subst-tv.simps infer-v-pairI b-of-subst* by *simp*
 next
 case (*infer-v-consI s dclist* Θ *dc tc* \mathcal{B} *va tv z*)

 have $\Theta ; \mathcal{B} ; \Gamma'[x::=v]_{\Gamma_v} @ \Gamma \vdash (V-cons\ s\ dc\ va[x::=v]_{vv}) \Rightarrow \{ z : B-id\ s \mid [[z]^v]^{ce} == [V-cons\ s\ dc\ va[x::=v]_{vv}]^{ce} \}$
 proof
 show $td : \langle AF-typedef\ s\ dclist \in set\ \Theta \rangle$ using *infer-v-consI* by *auto*
 show $dc : \langle (dc, tc) \in set\ dclist \rangle$ using *infer-v-consI* by *auto*
 show $\langle \Theta ; \mathcal{B} ; \Gamma'[x::=v]_{\Gamma_v} @ \Gamma \vdash va[x::=v]_{vv} \Rightarrow tv[x::=v]_{\tau_v} \rangle$ using *infer-v-consI* by *auto*
 have $\langle \Theta ; \mathcal{B} ; \Gamma'[x::=v]_{\Gamma_v} @ \Gamma \vdash tv[x::=v]_{\tau_v} \lesssim tc[x::=v]_{\tau_v} \rangle$
 using *subst-subtype-tau infer-v-consI* by *metis*
 moreover have $atom\ x \# tc$ using *wfTh-lookup-supp-empty[OF td dc] infer-v-wf infer-v-consI*
fresh-def by *fast*
 ultimately show $\langle \Theta ; \mathcal{B} ; \Gamma'[x::=v]_{\Gamma_v} @ \Gamma \vdash tv[x::=v]_{\tau_v} \lesssim tc \rangle$ by *simp*
 show $\langle atom\ z \# va[x::=v]_{vv} \rangle$ using *infer-v-consI* by *auto*
 show $\langle atom\ z \# (\Theta, \mathcal{B}, \Gamma'[x::=v]_{\Gamma_v} @ \Gamma) \rangle$ by (*fresh-mth add: infer-v-consI fresh-subst-gv-inside*)
 qed
 thus $?case$ using *subst-vv.simps subst-t[of z x v]* *infer-v-consI* by *metis*

 next
 case (*infer-v-conspI s bv dclist* Θ *dc tc* \mathcal{B} *va tv b z*)
 have $\Theta ; \mathcal{B} ; \Gamma'[x::=v]_{\Gamma_v} @ \Gamma \vdash (V-consp\ s\ dc\ b\ va[x::=v]_{vv}) \Rightarrow \{ z : B-app\ s\ b \mid [[z]^v]^{ce} == [V-consp\ s\ dc\ b\ va[x::=v]_{vv}]^{ce} \}$
 proof
 show $td : \langle AF-typedef-poly\ s\ bv\ dclist \in set\ \Theta \rangle$ using *infer-v-conspI* by *auto*
 show $dc : \langle (dc, tc) \in set\ dclist \rangle$ using *infer-v-conspI* by *auto*
 show $\langle \Theta ; \mathcal{B} ; \Gamma'[x::=v]_{\Gamma_v} @ \Gamma \vdash va[x::=v]_{vv} \Rightarrow tv[x::=v]_{\tau_v} \rangle$ using *infer-v-conspI* by *metis*
 have $\langle \Theta ; \mathcal{B} ; \Gamma'[x::=v]_{\Gamma_v} @ \Gamma \vdash tv[x::=v]_{\tau_v} \lesssim tc[bv::=b]_{\tau_b}[x::=v]_{\tau_v} \rangle$
 using *subst-subtype-tau infer-v-conspI* by *metis*
 moreover have $atom\ x \# tc[bv::=b]_{\tau_b}$ proof –
 have $supp\ tc \subseteq \{ atom\ bv \}$ 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 $\langle \Theta ; \mathcal{B} ; \Gamma'[x::=v]_{\Gamma_v} @ \Gamma \vdash tv[x::=v]_{\tau_v} \lesssim tc[bv::=b]_{\tau_b} \rangle$ by *simp*
 show $\langle atom\ z \# (\Theta, \mathcal{B}, \Gamma'[x::=v]_{\Gamma_v} @ \Gamma, va[x::=v]_{vv}, b) \rangle$ proof –
 have $atom\ z \# va[x::=v]_{vv}$ using *fresh-subst-v-if infer-v-conspI subst-v-v-def* by *metis*
 moreover have $atom\ z \# \Gamma'[x::=v]_{\Gamma_v} @ \Gamma$ using *fresh-subst-gv-inside infer-v-conspI* by *metis*
 ultimately show $?thesis$ using *fresh-prodN infer-v-conspI* by *metis*
 qed
 show $\langle atom\ bv \# (\Theta, \mathcal{B}, \Gamma'[x::=v]_{\Gamma_v} @ \Gamma, va[x::=v]_{vv}, b) \rangle$ proof –
 have $atom\ bv \# va[x::=v]_{vv}$ using *fresh-subst-v-if infer-v-conspI subst-v-v-def* by *metis*
 moreover have $atom\ bv \# \Gamma'[x::=v]_{\Gamma_v} @ \Gamma$ using *fresh-subst-gv-inside infer-v-conspI* by *metis*
 ultimately show $?thesis$ using *fresh-prodN infer-v-conspI* by *metis*
 qed
 show $\Theta ; \mathcal{B} \vdash_{wf} b$ using *infer-v-conspI* by *auto*
 qed
 thus $?case$ using *subst-vv.simps subst-t[of z x v]* *infer-v-conspI* by *metis*

qed

lemma *subst-infer-check-v*:

fixes $v::v$ **and** $v'::v$

assumes $\Theta ; \mathcal{B} ; \Gamma \vdash v \Rightarrow \tau_1$ **and**

$check\text{-}v \ \Theta \ \mathcal{B} \ (\Gamma' @ ((x, b_1, c0[z0::=[x]^v]_{cv}) \#_{\Gamma} \Gamma)) \ v' \ \tau_2$ **and**

$\Theta ; \mathcal{B} ; \Gamma \vdash \tau_1 \lesssim \llbracket z0 : b_1 \mid c0 \rrbracket$ **and** $atom \ z0 \ \# \ (x, v)$

shows $check\text{-}v \ \Theta \ \mathcal{B} \ ((\Gamma'[x::=v]_{\Gamma v}) @ \Gamma) \ (v'[x::=v]_{vv}) \ (\tau_2[x::=v]_{\tau v})$

proof –

obtain τ_2' **where** $t2: infer\text{-}v \ \Theta \ \mathcal{B} \ (\Gamma' @ (x, b_1, c0[z0::=[x]^v]_{cv}) \#_{\Gamma} \Gamma) \ v' \ \tau_2' \wedge \ \Theta ; \mathcal{B} ; (\Gamma' @ (x, b_1, c0[z0::=[x]^v]_{cv}) \#_{\Gamma} \Gamma) \vdash \tau_2' \lesssim \tau_2$

using *check-v-elimss assms* **by** *blast*

hence $infer\text{-}v \ \Theta \ \mathcal{B} \ ((\Gamma'[x::=v]_{\Gamma v}) @ \Gamma) \ (v'[x::=v]_{vv}) \ (\tau_2'[x::=v]_{\tau v})$

using *subst-infer-v[OF - assms(1) assms(3) assms(4)]* **by** *blast*

moreover **hence** $\Theta ; \mathcal{B} ; ((\Gamma'[x::=v]_{\Gamma v}) @ \Gamma) \vdash \tau_2'[x::=v]_{\tau v} \lesssim \tau_2[x::=v]_{\tau v}$

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 $\llbracket z : b \mid CE\text{-}val \ (V\text{-}var \ z) \rrbracket == CE\text{-}val \ v' \ \llbracket [x::=v]_{\tau v} \rrbracket = \llbracket z : b \mid CE\text{-}val \ (V\text{-}var \ z) \rrbracket == CE\text{-}val \ v'[x::=v]_{vv} \ \llbracket$

using *assms* **by** *auto*

lemma *subst-infer-v-form*:

fixes $v::v$ **and** $v'::v$ **and** $\Gamma::\Gamma$

assumes $\Theta ; \mathcal{B} ; \Gamma \vdash v \Rightarrow \tau_1$ **and**

$\Theta ; \mathcal{B} ; \Gamma' @ ((x, b_1, c0[z0::=[x]^v]_{cv}) \#_{\Gamma} \Gamma) \vdash v' \Rightarrow \tau_2$ **and** $b = b\text{-of} \ \tau_2$

$\Theta ; \mathcal{B} ; \Gamma \vdash \tau_1 \lesssim (\llbracket z0 : b_1 \mid c0 \rrbracket)$ **and** $atom \ z0 \ \# \ (x, v)$ **and** $atom \ z3' \ \# \ (x, v, v', \Gamma' @ ((x, b_1, c0[z0::=[x]^v]_{cv}) \#_{\Gamma} \Gamma))$

shows $\langle \Theta ; \mathcal{B} ; \Gamma'[x::=v]_{\Gamma v} @ \Gamma \vdash v'[x::=v]_{vv} \Rightarrow \llbracket z3' : b \mid CE\text{-}val \ (V\text{-}var \ z3') \rrbracket == CE\text{-}val \ v'[x::=v]_{vv} \ \llbracket$

proof –

have $\Theta ; \mathcal{B} ; \Gamma' @ ((x, b_1, c0[z0::=[x]^v]_{cv}) \#_{\Gamma} \Gamma) \vdash v' \Rightarrow \llbracket z3' : b\text{-of} \ \tau_2 \mid C\text{-eq} \ (CE\text{-}val \ (V\text{-}var \ z3')) \ (CE\text{-}val \ v') \ \llbracket$

proof(*rule infer-v-form4*)

show $\langle \Theta ; \mathcal{B} ; \Gamma' @ (x, b_1, c0[z0::=[x]^v]_{cv}) \#_{\Gamma} \Gamma \vdash v' \Rightarrow \tau_2 \rangle$ **using** *assms* **by** *metis*

show $\langle atom \ z3' \ \# \ (v', \Gamma' @ (x, b_1, c0[z0::=[x]^v]_{cv}) \#_{\Gamma} \Gamma) \rangle$ **using** *assms fresh-prodN* **by** *metis*

show $\langle b\text{-of} \ \tau_2 = b\text{-of} \ \tau_2 \rangle$ **by** *auto*

qed

hence $\langle \Theta ; \mathcal{B} ; \Gamma'[x::=v]_{\Gamma v} @ \Gamma \vdash v'[x::=v]_{vv} \Rightarrow \llbracket z3' : b\text{-of} \ \tau_2 \mid CE\text{-}val \ (V\text{-}var \ z3') \rrbracket == CE\text{-}val \ v' \ \llbracket [x::=v]_{\tau v} \rrbracket$

using *subst-infer-v assms* **by** *metis*

thus *?thesis* **using** *type-veq-subst fresh-prodN assms* **by** *metis*

qed

14.8 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

$\Theta ; \Phi ; \mathcal{B} ; G ; \Delta \vdash e \Rightarrow \tau_2$ **and** $G = (\Gamma' @ ((x, b_1, \text{subst-cv } c0 \ z0 \ (V\text{-var } x)) \#_{\Gamma} \Gamma))$

$\Theta ; \mathcal{B} ; \Gamma \vdash v \Rightarrow \tau_1$ **and**

$\Theta ; \mathcal{B} ; \Gamma \vdash \tau_1 \lesssim \llbracket z0 : b_1 \mid c0 \rrbracket$ **and** $\text{atom } z0 \nVdash (x, v)$

shows $\Theta ; \Phi ; \mathcal{B} ; ((\Gamma'[x::=v]_{\Gamma_v}) @ \Gamma) ; (\Delta[x::=v]_{\Delta_v}) \vdash (\text{subst-ev } e \ x \ v) \Rightarrow \tau_2[x::=v]_{\tau_v}$

using *assms proof(nominal-induct avoiding: x v rule: infer-e.strong-induct)*

case (*infer-e-valI* $\Theta \ \mathcal{B} \ \Gamma'' \ \Delta \ \Phi \ v' \ \tau$)

have $\Theta ; \Phi ; \mathcal{B} ; \Gamma'[x::=v]_{\Gamma_v} @ \Gamma ; \Delta[x::=v]_{\Delta_v} \vdash (AE\text{-val } (v'[x::=v]_{vv})) \Rightarrow \tau[x::=v]_{\tau_v}$

proof

show $\Theta ; \mathcal{B} ; \Gamma'[x::=v]_{\Gamma_v} @ \Gamma \vdash_{wf} \Delta[x::=v]_{\Delta_v}$ **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 $\Theta ; \mathcal{B} ; \Gamma'[x::=v]_{\Gamma_v} @ \Gamma \vdash v'[x::=v]_{vv} \Rightarrow \tau[x::=v]_{\tau_v}$ **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* $\Theta \ \mathcal{B} \ \Gamma'' \ \Delta \ \Phi \ v1 \ z1 \ c1 \ v2 \ z2 \ c2 \ z3$)

hence *z3f*: $\text{atom } z3 \nVdash CE\text{-op Plus } [v1]^{ce} [v2]^{ce}$ **using** *e.fresh ce.fresh opp.fresh* **by** *metis*

obtain $z3':x$ **where** $*: \text{atom } z3' \nVdash (x, v, AE\text{-op Plus } v1 \ v2, \ CE\text{-op Plus } [v1]^{ce} [v2]^{ce}, AE\text{-op Plus } v1[x::=v]_{vv} \ v2[x::=v]_{vv}, \ CE\text{-op Plus } [v1[x::=v]_{vv}]^{ce} [v2[x::=v]_{vv}]^{ce}, \Gamma'[x::=v]_{\Gamma_v} @ \Gamma)$

using *obtain-fresh* **by** *metis*

hence $*(\llbracket z3 : B\text{-int} \mid CE\text{-val } (V\text{-var } z3) == CE\text{-op Plus } [v1]^{ce} [v2]^{ce} \rrbracket) = \llbracket z3' : B\text{-int} \mid CE\text{-val } (V\text{-var } z3') == CE\text{-op Plus } [v1]^{ce} [v2]^{ce} \rrbracket$

using *type-e-eq infer-e-plusI fresh-Pair z3f* **by** *metis*

obtain $z1' \ b1' \ c1'$ **where** $z1: \text{atom } z1' \nVdash (x, v) \wedge \llbracket z1 : B\text{-int} \mid c1 \rrbracket = \llbracket z1' : b1' \mid c1' \rrbracket$ **using** *obtain-fresh-z* **by** *metis*

obtain $z2' \ b2' \ c2'$ **where** $z2: \text{atom } z2' \nVdash (x, v) \wedge \llbracket z2 : B\text{-int} \mid c2 \rrbracket = \llbracket z2' : b2' \mid c2' \rrbracket$ **using** *obtain-fresh-z* **by** *metis*

have $bb: b1' = B\text{-int} \wedge b2' = B\text{-int}$ **using** $z1 \ z2 \ \tau.\text{eq-iff}$ **by** *metis*

have $\Theta ; \Phi ; \mathcal{B} ; \Gamma'[x::=v]_{\Gamma_v} @ \Gamma ; \Delta[x::=v]_{\Delta_v} \vdash (AE\text{-op Plus } (v1[x::=v]_{vv}) (v2[x::=v]_{vv})) \Rightarrow \llbracket z3' : B\text{-int} \mid CE\text{-val } (V\text{-var } z3') == CE\text{-op Plus } ([v1[x::=v]_{vv}]^{ce}) ([v2[x::=v]_{vv}]^{ce}) \rrbracket$

proof

show $\langle \Theta ; \mathcal{B} ; \Gamma'[x::=v]_{\Gamma_v} @ \Gamma \vdash_{wf} \Delta[x::=v]_{\Delta_v} \rangle$

using *infer-e-plusI wfD-subst subtype-eq-base2 b-of.simps* **by** *metis*

show $\langle \Theta \vdash_{wf} \Phi \rangle$ **using** *infer-e-plusI* **by** *blast*
show $\langle \Theta ; \mathcal{B} ; \Gamma[x::=v]_{\Gamma_v} @ \Gamma \vdash v1[x::=v]_{vv} \Rightarrow \llbracket z1' : B-int \mid c1'[x::=v]_{cv} \rrbracket \rangle$ **using** *subst-tv.simps*
subst-infer-v infer-e-plusI z1 bb by metis
show $\langle \Theta ; \mathcal{B} ; \Gamma[x::=v]_{\Gamma_v} @ \Gamma \vdash v2[x::=v]_{vv} \Rightarrow \llbracket z2' : B-int \mid c2'[x::=v]_{cv} \rrbracket \rangle$ **using** *subst-tv.simps*
subst-infer-v infer-e-plusI z2 bb by metis
show $\langle atom\ z3' \# AE-op\ Plus\ v1[x::=v]_{vv}\ v2[x::=v]_{vv} \rangle$ **using** *fresh-prod6* ***** **by** *metis*
show $\langle atom\ z3' \# \Gamma[x::=v]_{\Gamma_v} @ \Gamma \rangle$ **using** ***** **by** *auto*
qed
moreover have $\llbracket z3' : B-int \mid CE-val\ (V-var\ z3') == CE-op\ Plus\ ([v1[x::=v]_{vv}]^{ce})\ ([v2[x::=v]_{vv}]^{ce}) \rrbracket$
 $= \llbracket z3' : B-int \mid CE-val\ (V-var\ z3') == CE-op\ Plus\ [v1]^{ce}\ [v2]^{ce} \rrbracket[x::=v]_{\tau_v}$
by (*subst subst-tv.simps, auto simp add: **)
ultimately show *?case* **using** *subst-ev.simps* ****** **by** *metis*
next
case (*infer-e-leqI* $\Theta\ \mathcal{B}\ \Gamma''\ \Delta\ \Phi\ v1\ z1\ c1\ v2\ z2\ c2\ z3$)

hence *z3f*: $atom\ z3 \# CE-op\ LEq\ [v1]^{ce}\ [v2]^{ce}$ **using** *e.fresh ce.fresh opp.fresh* **by** *metis*

obtain $z3'::x$ **where** $atom\ z3' \# (x, v, AE-op\ LEq\ v1\ v2,\ CE-op\ LEq\ [v1]^{ce}\ [v2]^{ce}, CE-op\ LEq\ [v1[x::=v]_{vv}]^{ce}\ [v2[x::=v]_{vv}]^{ce}, AE-op\ LEq\ v1[x::=v]_{vv}\ v2[x::=v]_{vv}, \Gamma[x::=v]_{\Gamma_v} @ \Gamma)$
using *obtain-fresh* **by** *metis*
hence $**(\llbracket z3 : B-bool \mid CE-val\ (V-var\ z3) == CE-op\ LEq\ [v1]^{ce}\ [v2]^{ce} \rrbracket) = \llbracket z3' : B-bool \mid CE-val\ (V-var\ z3') == CE-op\ LEq\ [v1]^{ce}\ [v2]^{ce} \rrbracket$
using *type-e-eq infer-e-leqI fresh-Pair z3f* **by** *metis*

obtain $z1'\ b1'\ c1'$ **where** $z1:atom\ z1' \# (x, v) \wedge \llbracket z1 : B-int \mid c1 \rrbracket = \llbracket z1' : b1' \mid c1' \rrbracket$ **using** *obtain-fresh-z* **by** *metis*
obtain $z2'\ b2'\ c2'$ **where** $z2:atom\ z2' \# (x, v) \wedge \llbracket z2 : B-int \mid c2 \rrbracket = \llbracket z2' : b2' \mid c2' \rrbracket$ **using** *obtain-fresh-z* **by** *metis*

have $bb:b1' = B-int \wedge b2' = B-int$ **using** $z1\ z2\ \tau.eq-iff$ **by** *metis*

have $\Theta ; \Phi ; \mathcal{B} ; \Gamma[x::=v]_{\Gamma_v} @ \Gamma ; \Delta[x::=v]_{\Delta_v} \vdash (AE-op\ LEq\ (v1[x::=v]_{vv})\ (v2[x::=v]_{vv})) \Rightarrow \llbracket z3' : B-bool \mid CE-val\ (V-var\ z3') == CE-op\ LEq\ ([v1[x::=v]_{vv}]^{ce})\ ([v2[x::=v]_{vv}]^{ce}) \rrbracket$
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-leqI subtype-eq-base2 b-of.simps* **by** *metis*
show $\langle \Theta \vdash_{wf} \Phi \rangle$ **using** *infer-e-leqI(2)* **by** *auto*
show $\langle \Theta ; \mathcal{B} ; \Gamma[x::=v]_{\Gamma_v} @ \Gamma \vdash v1[x::=v]_{vv} \Rightarrow \llbracket z1' : B-int \mid c1'[x::=v]_{cv} \rrbracket \rangle$ **using** *subst-tv.simps*
subst-infer-v infer-e-leqI z1 bb by metis
show $\langle \Theta ; \mathcal{B} ; \Gamma[x::=v]_{\Gamma_v} @ \Gamma \vdash v2[x::=v]_{vv} \Rightarrow \llbracket z2' : B-int \mid c2'[x::=v]_{cv} \rrbracket \rangle$ **using** *subst-tv.simps*
subst-infer-v infer-e-leqI z2 bb by metis
show $\langle atom\ z3' \# AE-op\ LEq\ v1[x::=v]_{vv}\ v2[x::=v]_{vv} \rangle$ **using** *fresh-Pair* ***** **by** *metis*
show $\langle atom\ z3' \# \Gamma[x::=v]_{\Gamma_v} @ \Gamma \rangle$ **using** ***** **by** *auto*
qed
moreover have $\llbracket z3' : B-bool \mid CE-val\ (V-var\ z3') == CE-op\ LEq\ ([v1[x::=v]_{vv}]^{ce})\ ([v2[x::=v]_{vv}]^{ce}) \rrbracket$
 $= \llbracket z3' : B-bool \mid CE-val\ (V-var\ z3') == CE-op\ LEq\ [v1]^{ce}\ [v2]^{ce} \rrbracket[x::=v]_{\tau_v}$
using *subst-tv.simps subst-ev.simps* ***** **by** *auto*
ultimately show *?case* **using** *subst-ev.simps* ****** **by** *metis*
next
case (*infer-e-appI* $\Theta\ \mathcal{B}\ \Gamma''\ \Delta\ \Phi\ f\ x'\ b\ c\ \tau'\ s'\ v'\ \tau$)

hence $x \neq x'$ **using** $\langle atom\ x' \# \Gamma'' \rangle$ **using** *wfG-inside-x-neq wfX-wfY* **by** *metis*


```

show ?case proof(subst subst-ev.simps,rule)
  show  $\langle \Theta ; \mathcal{B} ; \Gamma'[x::=v]_{\Gamma_v} @ \Gamma \vdash_{wf} \Delta[x::=v]_{\Delta_v} \rangle$  using infer-e-appI wfD-subst subtype-eq-base2
b-of.simps by metis
  show  $\langle \Theta \vdash_{wf} \Phi \rangle$  using infer-e-appI by metis
  show  $\langle \text{Some } (AF\text{-fundef } f \ (AF\text{-fun-typ-none } (AF\text{-fun-typ } x' \ b \ c \ \tau' \ s')) = \text{lookup-fun } \Phi \ f) \rangle$  using
infer-e-appI by metis

have  $\langle \Theta ; \mathcal{B} ; \Gamma'[x::=v]_{\Gamma_v} @ \Gamma \vdash v'[x::=v]_{vv} \Leftarrow \{ x' : b \mid c \} [x::=v]_{\tau_v} \rangle$  proof(rule subst-infer-check-v
)
  show  $\Theta ; \mathcal{B} ; \Gamma \vdash v \Rightarrow \tau_1$  using infer-e-appI by metis
  show  $\Theta ; \mathcal{B} ; \Gamma' @ (x, b_1, c0[z0::=[x]_{cv}] \#_{\Gamma} \Gamma \vdash v' \Leftarrow \{ x' : b \mid c \})$  using infer-e-appI by
metis
  show  $\Theta ; \mathcal{B} ; \Gamma \vdash \tau_1 \lesssim \{ z0 : b_1 \mid c0 \}$  using infer-e-appI by metis
  show  $\text{atom } z0 \# (x, v)$  using infer-e-appI by metis
qed
moreover have  $\text{atom } x \# c$  using wfPhi-f-simple-supp-c infer-e-appI fresh-def  $\langle x \neq x' \rangle$ 
   $\text{atom-eq-iff empty-iff infer-e-appI.hyps insert-iff subset-singletonD}$  by metis

moreover hence  $\text{atom } x \# \{ x' : b \mid c \}$  using  $\tau.\text{fresh supp-b-empty fresh-def}$  by blast
ultimately show  $\langle \Theta ; \mathcal{B} ; \Gamma'[x::=v]_{\Gamma_v} @ \Gamma \vdash v'[x::=v]_{vv} \Leftarrow \{ x' : b \mid c \} \rangle$  using forget-subst-tv
by metis

have  $*$ :  $\text{atom } x' \# (x, v)$  using infer-v-fresh-g-fresh-xv infer-e-appI check-v-wf by blast

show  $\langle \text{atom } x' \# (\Theta, \Phi, \mathcal{B}, \Gamma'[x::=v]_{\Gamma_v} @ \Gamma, \Delta[x::=v]_{\Delta_v}, v'[x::=v]_{vv}, \tau[x::=v]_{\tau_v}) \rangle$ 
  apply(unfold fresh-prodN, intro conjI)
  apply (fresh-subst-mth-aux add: infer-e-appI fresh-subst-gv wfD-wf subst-g-inside)
  using infer-e-appI fresh-subst-gv wfD-wf subst-g-inside apply metis
  using infer-e-appI fresh-subst-dv-if apply metis
done

have  $\text{supp } \tau' \subseteq \{ \text{atom } x' \} \cup \text{supp } \mathcal{B}$  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  $\text{atom } x \# \tau'$  using  $\langle x \neq x' \rangle$  fresh-def supp-at-base x-not-in-b-set by fastforce
thus  $\langle \tau[x::=v'[x::=v]_{vv}]_v = \tau[x::=v]_{\tau_v} \rangle$  using subst-tv-commute infer-e-appI subst-defs by metis
qed
next
case (infer-e-appPI  $\Theta \mathcal{B} \Gamma'' \Delta \Phi b' f bv x' b c \tau' s' v' \tau$ )

hence  $x \neq x'$  using  $\langle \text{atom } x' \# \Gamma'' \rangle$  using wfG-inside-x-neq wfX-wfY by metis

show ?case proof(subst subst-ev.simps,rule)
  show  $\langle \Theta ; \mathcal{B} ; \Gamma'[x::=v]_{\Gamma_v} @ \Gamma \vdash_{wf} \Delta[x::=v]_{\Delta_v} \rangle$  using infer-e-appPI wfD-subst subtype-eq-base2
b-of.simps by metis
  show  $\langle \Theta \vdash_{wf} \Phi \rangle$  using infer-e-appPI(4) by auto
  show  $\Theta ; \mathcal{B} \vdash_{wf} b'$  using infer-e-appPI(5) by auto
  show  $\text{Some } (AF\text{-fundef } f \ (AF\text{-fun-typ-some } bv \ (AF\text{-fun-typ } x' \ b \ c \ \tau' \ s')) = \text{lookup-fun } \Phi \ f)$  using
infer-e-appPI(6) by auto

show  $\Theta ; \mathcal{B} ; \Gamma'[x::=v]_{\Gamma_v} @ \Gamma \vdash v'[x::=v]_{vv} \Leftarrow \{ x' : b[bv::=b]_b \mid c[bv::=b]_b \}$  proof –

```

have $\langle \Theta ; \mathcal{B} ; \Gamma[x::=v]_{\Gamma v} @ \Gamma \vdash v'[x::=v]_{vv} \Leftarrow \{ \{ x' : b[bv::=b]_{bb} \mid c[bv::=b]_{cb} \} [x::=v]_{\tau v} \rangle$
proof(*rule subst-infer-check-v*)
show $\Theta ; \mathcal{B} ; \Gamma \vdash v \Rightarrow \tau_1$ **using** *infer-e-appPI* **by** *metis*
show $\Theta ; \mathcal{B} ; \Gamma' @ (x, b_1, c0[z0::=[x]_{cv}) \#_{\Gamma} \Gamma \vdash v' \Leftarrow \{ \{ x' : b[bv::=b]_{bb} \mid c[bv::=b]_{cb} \} \}$
using *infer-e-appPI* *subst-defs* **by** *metis*
show $\Theta ; \mathcal{B} ; \Gamma \vdash \tau_1 \lesssim \{ \{ z0 : b_1 \mid c0 \} \}$ **using** *infer-e-appPI* **by** *metis*
show *atom* $z0 \# (x, v)$ **using** *infer-e-appPI* **by** *metis*
qed
moreover **have** *atom* $x \# c$ **proof** –
have *supp* $c \subseteq \{ \text{atom } x', \text{atom } bv \}$ **using** *wfPhi-f-poly-supp-c*[*OF infer-e-appPI*(6)] *infer-e-appPI*
by *metis*
thus *?thesis* **unfolding** *fresh-def* **using** $\langle x \neq x' \rangle$ *atom-eq-iff* **by** *auto*
qed
moreover **hence** *atom* $x \# \{ \{ x' : b[bv::=b]_{bb} \mid c[bv::=b]_{cb} \} \}$ **using** $\tau.\text{fresh}$ *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* $\tau' \subseteq \{ \text{atom } x', \text{atom } bv \}$ **using** *wfPhi-f-poly-supp-t* *infer-e-appPI* **by** *metis*
hence *atom* $x \# \tau'$ **using** *fresh-def* $\langle x \neq x' \rangle$ **by** *force*
hence $*:\text{atom } x \# \tau'[bv::=b]_{\tau b}$ **using** *subst-b-fresh-x* *subst-b- τ -def* **by** *metis*
have *atom* $x' \# (x, v)$ **using** *infer-v-fresh-g-fresh-xv* *infer-e-appPI* *check-v-wf* **by** *blast*
thus *atom* $x' \# \Gamma[x::=v]_{\Gamma v} @ \Gamma$ **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* $bv \# (\Theta, \Phi, \mathcal{B}, \Gamma[x::=v]_{\Gamma v} @ \Gamma, \Delta[x::=v]_{\Delta v}, b', v'[x::=v]_{vv}, \tau[x::=v]_{\tau v})$
by (*fresh-mth* *add*: *infer-e-appPI* *fresh-subst-gv-inside*)
qed
next
case (*infer-e-fstI* $\Theta \mathcal{B} \Gamma'' \Delta \Phi v' z' b1 b2 c z$)

hence *zf*: *atom* $z \# CE\text{-fst } [v]^{ce}$ **using** *ce.fresh* *e.fresh* *opp.fresh* **by** *metis*

obtain $z3'::x$ **where** $*:\text{atom } z3' \# (x, v, AE\text{-fst } v', CE\text{-fst } [v]^{ce}, AE\text{-fst } v'[x::=v]_{vv}, \Gamma[x::=v]_{\Gamma v} @ \Gamma)$
using *obtain-fresh* **by** *auto*
hence $**:(\{ \{ z : b1 \mid CE\text{-val } (V\text{-var } z) == CE\text{-fst } [v]^{ce} \} \} = \{ \{ z3' : b1 \mid CE\text{-val } (V\text{-var } z3') == CE\text{-fst } [v]^{ce} \} \})$
using *type-e-eq* *infer-e-fstI*(4) *fresh-Pair* *zf* **by** *metis*

obtain $z1' b1' c1'$ **where** $z1:\text{atom } z1' \# (x, v) \wedge \{ \{ z' : B\text{-pair } b1 b2 \mid c \} \} = \{ \{ z1' : b1' \mid c1' \} \}$ **using**
obtain-fresh-z **by** *metis*

have $bb:b1' = B\text{-pair } b1 b2$ **using** $z1$ *τ .eq-iff* **by** *metis*
have $\Theta ; \Phi ; \mathcal{B} ; \Gamma[x::=v]_{\Gamma v} @ \Gamma ; \Delta[x::=v]_{\Delta v} \vdash (AE\text{-fst } v'[x::=v]_{vv}) \Rightarrow \{ \{ z3' : b1 \mid CE\text{-val } (V\text{-var } z3') == CE\text{-fst } [v]^{ce} \} \}$
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-fstI* *subtype-eq-base2*
b-of.simps **by** *metis*
show $\langle \Theta \vdash_{wf} \Phi \rangle$ **using** *infer-e-fstI* **by** *metis*
show $\langle \Theta ; \mathcal{B} ; \Gamma[x::=v]_{\Gamma v} @ \Gamma \vdash v'[x::=v]_{vv} \Rightarrow \{ \{ z1' : B\text{-pair } b1 b2 \mid c1'[x::=v]_{cv} \} \rangle$ **using**
subst-tv.simps *subst-infer-v* *infer-e-fstI* $z1$ bb **by** *metis*

show $\langle \text{atom } z\beta' \# AE\text{-fst } v'[x::=v]_{vv} \rangle$ **using** *fresh-Pair* * **by** *metis*
show $\langle \text{atom } z\beta' \# \Gamma'[x::=v]_{\Gamma_v} @ \Gamma \rangle$ **using** * **by** *auto*
qed
moreover **have** $\llbracket z\beta' : b1 \mid CE\text{-val } (V\text{-var } z\beta') \rrbracket == CE\text{-fst } [v'[x::=v]_{vv}]^{ce} \rrbracket = \llbracket z\beta' : b1 \mid CE\text{-val } (V\text{-var } z\beta') \rrbracket == CE\text{-fst } [v]^{ce} \rrbracket [x::=v]_{\tau_v}$
using *subst-tv.simps subst-ev.simps* * **by** *auto*
ultimately **show** *?case* **using** *subst-ev.simps* * ** **by** *metis*
next
case (*infer-e-sndI* $\Theta \mathcal{B} \Gamma'' \Delta \Phi v' z' b1 b2 c z$)
hence *zf*: $\text{atom } z \# CE\text{-snd } [v]^{ce}$ **using** *ce.fresh e.fresh opp.fresh* **by** *metis*

obtain $z\beta'::x$ **where** $\ast:\text{atom } z\beta' \# (x,v,AE\text{-snd } v', CE\text{-snd } [v]^{ce}, AE\text{-snd } v'[x::=v]_{vv}, \Gamma'[x::=v]_{\Gamma_v} @ \Gamma, v', \Gamma'')$ **using** *obtain-fresh* **by** *auto*
hence $\ast:(\llbracket z : b2 \mid CE\text{-val } (V\text{-var } z) \rrbracket == CE\text{-snd } [v]^{ce} \rrbracket) = \llbracket z\beta' : b2 \mid CE\text{-val } (V\text{-var } z\beta') \rrbracket == CE\text{-snd } [v]^{ce} \rrbracket$
using *type-e-eq infer-e-sndI(4) fresh-Pair zf* **by** *metis*

obtain $z1' b2' c1'$ **where** $z1:\text{atom } z1' \# (x,v) \wedge \llbracket z' : B\text{-pair } b1 b2 \mid c \rrbracket = \llbracket z1' : b2' \mid c1' \rrbracket$ **using** *obtain-fresh-z* **by** *metis*

have $bb:b2' = B\text{-pair } b1 b2$ **using** $z1 \tau.\text{eq-iff}$ **by** *metis*

have $\Theta ; \Phi ; \mathcal{B} ; \Gamma'[x::=v]_{\Gamma_v} @ \Gamma ; \Delta[x::=v]_{\Delta_v} \vdash (AE\text{-snd } (v'[x::=v]_{vv})) \Rightarrow \llbracket z\beta' : b2 \mid CE\text{-val } (V\text{-var } z\beta') \rrbracket == CE\text{-snd } ([v'[x::=v]_{vv}]^{ce}) \rrbracket$
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-sndI subtype-eq-base2 b-of.simps* **by** *metis*
show $\langle \Theta \vdash_{wf} \Phi \rangle$ **using** *infer-e-sndI* **by** *metis*
show $\langle \Theta ; \mathcal{B} ; \Gamma'[x::=v]_{\Gamma_v} @ \Gamma \vdash v'[x::=v]_{vv} \Rightarrow \llbracket z1' : B\text{-pair } b1 b2 \mid c1'[x::=v]_{cv} \rrbracket \rangle$ **using** *subst-tv.simps subst-infer-v infer-e-sndI z1 bb* **by** *metis*

show $\langle \text{atom } z\beta' \# AE\text{-snd } v'[x::=v]_{vv} \rangle$ **using** *fresh-Pair* * **by** *metis*
show $\langle \text{atom } z\beta' \# \Gamma'[x::=v]_{\Gamma_v} @ \Gamma \rangle$ **using** * **by** *auto*
qed
moreover **have** $\llbracket z\beta' : b2 \mid CE\text{-val } (V\text{-var } z\beta') \rrbracket == CE\text{-snd } ([v'[x::=v]_{vv}]^{ce}) \rrbracket = \llbracket z\beta' : b2 \mid CE\text{-val } (V\text{-var } z\beta') \rrbracket == CE\text{-snd } [v]^{ce} \rrbracket [x::=v]_{\tau_v}$
by(*subst subst-tv.simps, auto simp add: fresh-prodN* *)
ultimately **show** *?case* **using** *subst-ev.simps* * ** **by** *metis*
next
case (*infer-e-lenI* $\Theta \mathcal{B} \Gamma'' \Delta \Phi v' z' c z$)
hence *zf*: $\text{atom } z \# CE\text{-len } [v]^{ce}$ **using** *ce.fresh e.fresh opp.fresh* **by** *metis*
obtain $z\beta'::x$ **where** $\ast:\text{atom } z\beta' \# (x,v,AE\text{-len } v', CE\text{-len } [v]^{ce}, AE\text{-len } v'[x::=v]_{vv}, \Gamma'[x::=v]_{\Gamma_v} @ \Gamma, \Gamma'', v')$ **using** *obtain-fresh* **by** *auto*
hence $\ast:(\llbracket z : B\text{-int } \mid CE\text{-val } (V\text{-var } z) \rrbracket == CE\text{-len } [v]^{ce} \rrbracket) = \llbracket z\beta' : B\text{-int } \mid CE\text{-val } (V\text{-var } z\beta') \rrbracket == CE\text{-len } [v]^{ce} \rrbracket$
using *type-e-eq infer-e-lenI fresh-Pair zf* **by** *metis*

have $\ast\ast:\Theta ; \mathcal{B} ; \Gamma'' \vdash v' \Rightarrow \llbracket z\beta' : B\text{-bitvec } \mid CE\text{-val } (V\text{-var } z\beta') \rrbracket == CE\text{-val } v' \rrbracket$
using *infer-e-lenI infer-v-form3[OF infer-e-lenI(3), of z3] b-of.simps * fresh-Pair* **by** *metis*

have $\Theta ; \Phi ; \mathcal{B} ; \Gamma'[x::=v]_{\Gamma_v} @ \Gamma ; \Delta[x::=v]_{\Delta_v} \vdash (AE\text{-len } (v'[x::=v]_{vv})) \Rightarrow \llbracket z\beta' : B\text{-int } \mid CE\text{-val}$

$(V\text{-var } z3') == CE\text{-len } ([v'[x::=v]_{vv}]^{ce}) \}$
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-lenI subtype-eq-base2 b-of.simps by metis*
show $\langle \Theta \vdash_{wf} \Phi \rangle$ **using** *infer-e-lenI by metis*

have $\langle \Theta ; \mathcal{B} ; \Gamma'[x::=v]_{\Gamma_v} @ \Gamma \vdash v'[x::=v]_{vv} \Rightarrow \{ z3' : B\text{-bitvec} \mid CE\text{-val } (V\text{-var } z3') == CE\text{-val } v' \}_{[x::=v]_{\tau_v}} \rangle$
proof(*rule subst-infer-v*)
show $\langle \Theta ; \mathcal{B} ; \Gamma \vdash v \Rightarrow \tau_1 \rangle$ **using** *infer-e-lenI by metis*
show $\langle \Theta ; \mathcal{B} ; \Gamma' @ (x, b_1, c0[z0::=[x]^v]_{cv}) \# \Gamma \vdash v' \Rightarrow \{ z3' : B\text{-bitvec} \mid [[z3']^v]^{ce} == [v']^{ce} \} \rangle$ **using** **** infer-e-lenI by metis*
show $\Theta ; \mathcal{B} ; \Gamma \vdash \tau_1 \lesssim \{ z0 : b_1 \mid c0 \}$ **using** *infer-e-lenI by metis*
show *atom z0 # (x, v)* **using** *infer-e-lenI by metis*
qed

thus $\langle \Theta ; \mathcal{B} ; \Gamma'[x::=v]_{\Gamma_v} @ \Gamma \vdash v'[x::=v]_{vv} \Rightarrow \{ z3' : B\text{-bitvec} \mid CE\text{-val } (V\text{-var } z3') == CE\text{-val } v'[x::=v]_{vv} \} \rangle$
using *subst-tv.simps subst-ev.simps fresh-Pair * fresh-prodN subst-vv.simps by auto*

show $\langle \text{atom } z3' \# AE\text{-len } v'[x::=v]_{vv} \rangle$ **using** *fresh-Pair * by metis*
show $\langle \text{atom } z3' \# \Gamma'[x::=v]_{\Gamma_v} @ \Gamma \rangle$ **using** *fresh-Pair * by metis*
qed

moreover have $\{ z3' : B\text{-int} \mid CE\text{-val } (V\text{-var } z3') == CE\text{-len } ([v'[x::=v]_{vv}]^{ce}) \} = \{ z3' : B\text{-int} \mid CE\text{-val } (V\text{-var } z3') == CE\text{-len } [v']^{ce} \}_{[x::=v]_{\tau_v}}$
using *subst-tv.simps subst-ev.simps * by auto*

ultimately show *?case using subst-ev.simps * ** by metis*
next
case (*infer-e-mvarI* $\Theta \mathcal{B} \Gamma'' \Phi \Delta u \tau$)

have $\Theta ; \Phi ; \mathcal{B} ; \Gamma'[x::=v]_{\Gamma_v} @ \Gamma ; \Delta[x::=v]_{\Delta_v} \vdash (AE\text{-mvar } u) \Rightarrow \tau[x::=v]_{\tau_v}$
proof
show $\langle \Theta ; \mathcal{B} \vdash_{wf} \Gamma'[x::=v]_{\Gamma_v} @ \Gamma \rangle$ **proof** –
have *wfV* $\Theta \mathcal{B} \Gamma v$ (*b-of* τ_1) **using** *infer-v-wf infer-e-mvarI by auto*
moreover have *b-of* $\tau_1 = b_1$ **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 $\Gamma' x b_1 c0[z0::=[x]^v]_{cv} \Gamma v$ infer-e-mvarI subst-g-inside by metis*
qed
show $\langle \Theta \vdash_{wf} \Phi \rangle$ **using** *infer-e-mvarI by auto*
show $\langle \Theta ; \mathcal{B} ; \Gamma'[x::=v]_{\Gamma_v} @ \Gamma \vdash_{wf} \Delta[x::=v]_{\Delta_v} \rangle$ **using** *wfD-subst infer-e-mvarI subtype-eq-base2 b-of.simps by metis*
show $\langle (u, \tau[x::=v]_{\tau_v}) \in \text{setD } \Delta[x::=v]_{\Delta_v} \rangle$ **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 # CE-concat $[v1]^{ce} [v2]^{ce}$ using ce.fresh e.fresh opp.fresh by metis*

obtain $z3'::x$ **where** $atom\ z3' \# (x, v, v1, v2, AE-concat\ v1\ v2, CE-concat\ [v1]^{ce}\ [v2]^{ce}, AE-concat\ (v1[x::=v]_{vv})\ (v2[x::=v]_{vv}), \Gamma'[x::=v]_{\Gamma v} @ \Gamma, \Gamma'', v1, v2)$ **using** *obtain-fresh* **by** *auto*

hence $*(\{ z3 : B-bitvec \mid CE-val\ (V-var\ z3) == CE-concat\ [v1]^{ce}\ [v2]^{ce} \}) = \{ z3' : B-bitvec \mid CE-val\ (V-var\ z3') == CE-concat\ [v1]^{ce}\ [v2]^{ce} \}$

using *type-e-eq infer-e-concatI fresh-Pair zf* **by** *metis*

have $zfx: atom\ x \# z3'$ **using** *fresh-at-base fresh-prodN* * **by** *auto*

have $v1: \Theta; \mathcal{B}; \Gamma'' \vdash v1 \Rightarrow \{ z3' : B-bitvec \mid CE-val\ (V-var\ z3') == CE-val\ v1 \}$

using *infer-e-concatI infer-v-form3 b-of.simps* * *fresh-Pair* **by** *metis*

have $v2: \Theta; \mathcal{B}; \Gamma'' \vdash v2 \Rightarrow \{ z3' : B-bitvec \mid CE-val\ (V-var\ z3') == CE-val\ v2 \}$

using *infer-e-concatI infer-v-form3 b-of.simps* * *fresh-Pair* **by** *metis*

have $\Theta; \Phi; \mathcal{B}; \Gamma'[x::=v]_{\Gamma v} @ \Gamma; \Delta[x::=v]_{\Delta v} \vdash (AE-concat\ (v1[x::=v]_{vv})\ (v2[x::=v]_{vv})) \Rightarrow \{ z3' : B-bitvec \mid CE-val\ (V-var\ z3') == CE-concat\ ([v1[x::=v]_{vv}]^{ce})\ ([v2[x::=v]_{vv}]^{ce}) \}$

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-concatI subtype-eq-base2 b-of.simps* **by** *metis*

show $\langle \Theta \vdash_{wf} \Phi \rangle$ **by** *(simp add: infer-e-concatI)*

show $\langle \Theta; \mathcal{B}; \Gamma'[x::=v]_{\Gamma v} @ \Gamma \vdash v1[x::=v]_{vv} \Rightarrow \{ z3' : B-bitvec \mid CE-val\ (V-var\ z3') == CE-val\ (v1[x::=v]_{vv}) \} \rangle$

using *subst-infer-v-form infer-e-concatI fresh-prodN* * *b-of.simps* **by** *metis*

show $\langle \Theta; \mathcal{B}; \Gamma'[x::=v]_{\Gamma v} @ \Gamma \vdash v2[x::=v]_{vv} \Rightarrow \{ z3' : B-bitvec \mid CE-val\ (V-var\ z3') == CE-val\ (v2[x::=v]_{vv}) \} \rangle$

using *subst-infer-v-form infer-e-concatI fresh-prodN* * *b-of.simps* **by** *metis*

show $\langle atom\ z3' \# AE-concat\ v1[x::=v]_{vv}\ v2[x::=v]_{vv} \rangle$ **using** *fresh-Pair* * **by** *metis*

show $\langle atom\ z3' \# \Gamma'[x::=v]_{\Gamma v} @ \Gamma \rangle$ **using** *fresh-Pair* * **by** *metis*

qed

moreover **have** $\{ z3' : B-bitvec \mid CE-val\ (V-var\ z3') == CE-concat\ ([v1[x::=v]_{vv}]^{ce})\ ([v2[x::=v]_{vv}]^{ce}) \} = \{ z3' : B-bitvec \mid CE-val\ (V-var\ z3') == CE-concat\ [v1]^{ce}\ [v2]^{ce} \} [x::=v]_{\tau v}$

using *subst-tv.simps subst-ev.simps* * **by** *auto*

ultimately **show** *?case* **using** *subst-ev.simps* * * **by** *metis*

next

case *(infer-e-splitI* $\Theta\ \mathcal{B}\ \Gamma''\ \Delta\ \Phi\ v1\ z1\ c1\ v2\ z2\ z3)$

hence $atom\ z3 \# (x, v)$ **using** *fresh-Pair* **by** *auto*

have $\langle x \neq z3 \rangle$ **using** *infer-e-splitI* **by** *force*

have $\Theta; \Phi; \mathcal{B}; (\Gamma'[x::=v]_{\Gamma v} @ \Gamma); \Delta[x::=v]_{\Delta v} \vdash (AE-split\ v1[x::=v]_{vv}\ v2[x::=v]_{vv}) \Rightarrow$

$\{ z3 : [B-bitvec, B-bitvec]^b \mid [v1[x::=v]_{vv}]^{ce} == [\#1 [[z3]^v]^{ce}]^{ce} @@ [\#2 [[z3]^v]^{ce}]^{ce}]^{ce} \}$ *AND*

$[[\#1 [[z3]^v]^{ce}]^{ce}]^{ce} == [v2[x::=v]_{vv}]^{ce} \}$

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 $\langle \Theta; \mathcal{B}; \Gamma'[x::=v]_{\Gamma v} @ \Gamma \vdash v1[x::=v]_{vv} \Rightarrow \{ z1 : B-bitvec \mid c1 [x::=v]_{\tau v} \} \rangle$

using *subst-infer-v infer-e-splitI* **by** *metis*

thus $\langle \Theta; \mathcal{B}; \Gamma'[x::=v]_{\Gamma v} @ \Gamma \vdash v1[x::=v]_{vv} \Rightarrow \{ z1 : B-bitvec \mid c1[x::=v]_{cv} \} \rangle$

using *infer-e-splitI subst-tv.simps fresh-Pair* **by** *metis*

have $\langle x \neq z2 \rangle$ **using** *infer-e-splitI* **by** *force*

have ($\{ z2 : B\text{-int} \mid ([\text{leq } [[L\text{-num } 0]^v]^{ce} [[z2]^v]^{ce}]^{ce} == [[L\text{-true }]^v]^{ce})$
 $AND ([\text{leq } [[z2]^v]^{ce} [[v1[x::=v]_{vv}]^{ce}]^{ce}]^{ce} == [[L\text{-true }]^v]^{ce}) \}$) =
 $(\{ z2 : B\text{-int} \mid ([\text{leq } [[L\text{-num } 0]^v]^{ce} [[z2]^v]^{ce}]^{ce} == [[L\text{-true }]^v]^{ce})$
 $AND ([\text{leq } [[z2]^v]^{ce} [[v1]^{ce}]^{ce}]^{ce} == [[L\text{-true }]^v]^{ce}) \}$ $[x::=v]_{\tau v}$)
unfolding *subst-cv.simps subst-cev.simps subst-vv.simps* **using** $\langle x \neq z2 \rangle$ *infer-e-splitI fresh-Pair*
by *simp*

thus $\langle \Theta ; \mathcal{B} ; \Gamma[x::=v]_{\Gamma v} @$
 $\Gamma \vdash v2[x::=v]_{vv} \Leftarrow \{ z2 : B\text{-int} \mid [\text{leq } [[L\text{-num } 0]^v]^{ce} [[z2]^v]^{ce}]^{ce} == [[L\text{-true }]^v]^{ce}]^v]^{ce}$
 $AND [\text{leq } [[z2]^v]^{ce} [[v1[x::=v]_{vv}]^{ce}]^{ce}]^{ce} == [[L\text{-true }]^v]^{ce} \}$
using *infer-e-splitI subst-infer-check-v fresh-Pair* **by** *metis*

show $\langle \text{atom } z1 \# AE\text{-split } v1[x::=v]_{vv} v2[x::=v]_{vv} \rangle$ **using** *infer-e-splitI fresh-subst-vv-if* **by** *auto*
show $\langle \text{atom } z2 \# AE\text{-split } v1[x::=v]_{vv} v2[x::=v]_{vv} \rangle$ **using** *infer-e-splitI fresh-subst-vv-if* **by** *auto*
show $\langle \text{atom } z3 \# AE\text{-split } v1[x::=v]_{vv} v2[x::=v]_{vv} \rangle$ **using** *infer-e-splitI fresh-subst-vv-if* **by** *auto*

show $\langle \text{atom } z3 \# \Gamma[x::=v]_{\Gamma v} @ \Gamma \rangle$ **using** *fresh-subst-gv-inside infer-e-splitI* **by** *metis*
show $\langle \text{atom } z2 \# \Gamma[x::=v]_{\Gamma v} @ \Gamma \rangle$ **using** *fresh-subst-gv-inside infer-e-splitI* **by** *metis*
show $\langle \text{atom } z1 \# \Gamma[x::=v]_{\Gamma v} @ \Gamma \rangle$ **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 $(AE\text{-val } x)$
then show *?case using infer-e-elim(7)[OF AE-val(1)] infer-e-elim(7)[OF AE-val(2)] infer-v-uniqueness*
by *metis*
next
case $(AE\text{-app } f v)$

obtain $x1 b1 c1 s1' \tau1'$ **where** $t1: \text{Some } (AF\text{-fundef } f (AF\text{-fun-tyt-none } (AF\text{-fun-tyt } x1 b1 c1 \tau1' s1')) = \text{lookup-fun } \Phi f \wedge \tau_1 = \tau1'[x1::=v]_{\tau v}$ **using** *infer-e-app2E[OF AE-app(1)]* **by** *metis*
moreover obtain $x2 b2 c2 s2' \tau2'$ **where** $t2: \text{Some } (AF\text{-fundef } f (AF\text{-fun-tyt-none } (AF\text{-fun-tyt } x2 b2 c2 \tau2' s2')) = \text{lookup-fun } \Phi f \wedge \tau_2 = \tau2'[x2::=v]_{\tau v}$ **using** *infer-e-app2E[OF AE-app(2)]* **by** *metis*

have $\tau1'[x1::=v]_{\tau v} = \tau2'[x2::=v]_{\tau v}$ **using** $t1$ **and** $t2$ *fun-ret-unique* **by** *metis*
thus *?thesis using t1 t2 by auto*

next

case $(AE\text{-appP } f b v)$
obtain $bv1 x1 b1 c1 s1' \tau1'$ **where** $t1: \text{Some } (AF\text{-fundef } f (AF\text{-fun-tyt-some } bv1 (AF\text{-fun-tyt } x1 b1 c1 \tau1' s1')) = \text{lookup-fun } \Phi f \wedge \tau_1 = \tau1'[bv1::=b]_{\tau b}[x1::=v]_{\tau v}$ **using** *infer-e-appP2E[OF AE-appP(1)]* **by** *metis*
moreover obtain $bv2 x2 b2 c2 s2' \tau2'$ **where** $t2: \text{Some } (AF\text{-fundef } f (AF\text{-fun-tyt-some } bv2 (AF\text{-fun-tyt } x2 b2 c2 s2' \tau2')) = \text{lookup-fun } \Phi f \wedge \tau_2 = \tau2'[bv2::=b]_{\tau b}[x2::=v]_{\tau v}$

$x2\ b2\ c2\ \tau2'\ s2')) = \text{lookup-fun } \Phi\ f \wedge \tau_2 = \tau2'[bv2::=b]_{\tau b}[x2::=v]_{\tau v} \text{ using } \text{infer-e-appP2E}[OF\ AE\text{-appP}(2)] \text{ by } \text{metis}$

have $\tau1'[bv1::=b]_{\tau b}[x1::=v]_{\tau v} = \tau2'[bv2::=b]_{\tau b}[x2::=v]_{\tau v} \text{ using } t1 \text{ and } t2 \text{ fun-poly-ret-unique by } \text{metis}$

thus $?thesis \text{ using } t1\ t2 \text{ by } \text{auto}$

next

case $(AE\text{-op } opp\ v1\ v2)$

show $?case \text{ proof}(cases\ opp = Plus)$

case $True$

hence $\Theta ; \Phi ; \mathcal{B} ; GNil ; \Delta \vdash AE\text{-op } Plus\ v1\ v2 \Rightarrow \tau_1 \text{ and } \Theta ; \Phi ; \mathcal{B} ; GNil ; \Delta \vdash AE\text{-op } Plus\ v1\ v2 \Rightarrow \tau_2 \text{ using } AE\text{-op by } \text{auto}$

thm $\text{infer-e-elim}(3)$

thus $?thesis \text{ using } \text{infer-e-elim}(11)[OF\ \langle \Theta ; \Phi ; \mathcal{B} ; GNil ; \Delta \vdash AE\text{-op } Plus\ v1\ v2 \Rightarrow \tau_1 \rangle] \text{ infer-e-elim}(11)[OF\ \langle \Theta ; \Phi ; \mathcal{B} ; GNil ; \Delta \vdash AE\text{-op } Plus\ v1\ v2 \Rightarrow \tau_2 \rangle]$

by force

next

case $False$

hence $opp = LEq \text{ using } opp.\text{strong-exhaust by } \text{auto}$

hence $\Theta ; \Phi ; \mathcal{B} ; GNil ; \Delta \vdash AE\text{-op } LEq\ v1\ v2 \Rightarrow \tau_1 \text{ and } \Theta ; \Phi ; \mathcal{B} ; GNil ; \Delta \vdash AE\text{-op } LEq\ v1\ v2 \Rightarrow \tau_2 \text{ using } AE\text{-op by } \text{auto}$

thm $\text{infer-e-elim}(3)$

thus $?thesis \text{ using } \text{infer-e-elim}(12)[OF\ \langle \Theta ; \Phi ; \mathcal{B} ; GNil ; \Delta \vdash AE\text{-op } LEq\ v1\ v2 \Rightarrow \tau_1 \rangle] \text{ infer-e-elim}(12)[OF\ \langle \Theta ; \Phi ; \mathcal{B} ; GNil ; \Delta \vdash AE\text{-op } LEq\ v1\ v2 \Rightarrow \tau_2 \rangle]$

by force

qed

next

case $(AE\text{-concat } v1\ v2)$

obtain $z3::x \text{ where } t1:\tau_1 = \llbracket z3 : B\text{-bitvec} \mid \llbracket z3 \rrbracket^v \rrbracket^{ce} == CE\text{-concat } [v1]^{ce} [v2]^{ce} \rrbracket \wedge \text{atom } z3 \# v1 \wedge \text{atom } z3 \# v2 \text{ using } \text{infer-e-elim}(18)[OF\ AE\text{-concat}(1)] \text{ by } \text{metis}$

obtain $z3'::x \text{ where } t2:\tau_2 = \llbracket z3' : B\text{-bitvec} \mid \llbracket z3' \rrbracket^v \rrbracket^{ce} == CE\text{-concat } [v1]^{ce} [v2]^{ce} \rrbracket \wedge \text{atom } z3' \# v1 \wedge \text{atom } z3' \# v2 \text{ using } \text{infer-e-elim}(18)[OF\ AE\text{-concat}(2)] \text{ by } \text{metis}$

thus $?case \text{ using } t1\ t2\ \text{type-e-eq } ce.\text{fresh by } \text{metis}$

next

case $(AE\text{-fst } v)$

obtain $z1 \text{ and } b1 \text{ where } \tau_1 = \llbracket z1 : b1 \mid CE\text{-val } (V\text{-var } z1) == (CE\text{-fst } [v]^{ce}) \rrbracket \text{ using } \text{infer-v-form } AE\text{-fst by } \text{auto}$

obtain $xx :: x \text{ and } bb :: b \text{ and } xxa :: x \text{ and } bba :: b \text{ and } cc :: c \text{ where}$

$f1: \tau_2 = \llbracket xx : bb \mid CE\text{-val } (V\text{-var } xx) == CE\text{-fst } [v]^{ce} \rrbracket \wedge \Theta ; \mathcal{B} ; GNil \vdash_{wf} \Delta \wedge \Theta ; \mathcal{B} ; GNil \vdash v \Rightarrow \llbracket xxa : B\text{-pair } bb\ bba \mid cc \rrbracket \wedge \text{atom } xx \# v$

using $\text{infer-e-elim}(8)[OF\ AE\text{-fst}(2)] \text{ by } \text{metis}$

obtain $xxb :: x \text{ and } bbb :: b \text{ and } xxc :: x \text{ and } bbc :: b \text{ and } cca :: c \text{ where}$

$f2: \tau_1 = \llbracket xxb : bbb \mid CE\text{-val } (V\text{-var } xxb) == CE\text{-fst } [v]^{ce} \rrbracket \wedge \Theta ; \mathcal{B} ; GNil \vdash_{wf} \Delta \wedge \Theta ; \mathcal{B} ; GNil \vdash v \Rightarrow \llbracket xxc : B\text{-pair } bbb\ bbc \mid cca \rrbracket \wedge \text{atom } xxb \# v$

using $\text{infer-e-elim}(8)[OF\ AE\text{-fst}(1)] \text{ by } \text{metis}$

then have $B\text{-pair } bb\ bba = B\text{-pair } bbb\ bbc$

using $f1 \text{ by } (\text{metis } (\text{no-types})\ b\text{-of.simps infer-v-uniqueness})$

then have $\llbracket xx : bbb \mid CE\text{-val } (V\text{-var } xx) == CE\text{-fst } [v]^{ce} \rrbracket = \tau_2$
 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
 $f1: \tau_2 = \llbracket xx : bba \mid CE\text{-val } (V\text{-var } xx) == CE\text{-snd } [v]^{ce} \rrbracket \wedge \Theta ; \mathcal{B} ; GNil \vdash_{wf} \Delta \wedge \Theta ; \mathcal{B} ; GNil$
 $\vdash v \Rightarrow \llbracket xxa : B\text{-pair } bb \ bba \mid cc \rrbracket \wedge \text{atom } xx \# v$
 using *infer-e-elimis(9)[OF AE-snd(2)]* by *metis*
 obtain $xxb :: x$ and $bbb :: b$ and $xxc :: x$ and $bbc :: b$ and $cca :: c$ where
 $f2: \tau_1 = \llbracket xxb : bbc \mid CE\text{-val } (V\text{-var } xxb) == CE\text{-snd } [v]^{ce} \rrbracket \wedge \Theta ; \mathcal{B} ; GNil \vdash_{wf} \Delta \wedge \Theta ; \mathcal{B} ; GNil$
 $\vdash v \Rightarrow \llbracket xxc : B\text{-pair } bbb \ bbc \mid cca \rrbracket \wedge \text{atom } xxb \# v$
 using *infer-e-elimis(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 $\llbracket xx : bbc \mid CE\text{-val } (V\text{-var } xx) == CE\text{-snd } [v]^{ce} \rrbracket = \tau_2$
 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-elimis(10)[OF AE-mvar(1)] infer-e-elimis(10)[OF AE-mvar(2)] wfD-unique*
 by *metis*
 next
 case (*AE-len x*)
 then show *?case* using *infer-e-elimis(16)[OF AE-len(1)] infer-e-elimis(16)[OF AE-len(2)]* by *force*
 next
 case (*AE-split x1a x2*)
 then show *?case* using *infer-e-elimis(22)[OF AE-split(1)] infer-e-elimis(22)[OF AE-split(2)]* by *force*
 qed

14.9 Statements

lemma *subst-infer-check-v1*:

fixes $v::v$ and $v'::v$ and $\Gamma::\Gamma$
 assumes $\Gamma = \Gamma_1 @ ((x, b_1, c0[z0::=[x]^v]_{cv}) \# \Gamma_2)$ and
 $\Theta ; \mathcal{B} ; \Gamma_2 \vdash v \Rightarrow \tau_1$ and
 $\Theta ; \mathcal{B} ; \Gamma \vdash v' \Leftarrow \tau_2$ and
 $\Theta ; \mathcal{B} ; \Gamma_2 \vdash \tau_1 \lesssim \llbracket z0 : b_1 \mid c0 \rrbracket$ and $\text{atom } z0 \# (x, v)$
 shows $\Theta ; \mathcal{B} ; \Gamma[x::=v]_{\Gamma v} \vdash v'[x::=v]_{vv} \Leftarrow \tau_2[x::=v]_{\tau v}$
 using *subst-g-inside check-v-wf assms subst-infer-check-v* by *metis*

lemma *infer-v-c-valid*:

assumes $\Theta ; \mathcal{B} ; \Gamma \vdash v \Rightarrow \tau$ and $\Theta ; \mathcal{B} ; \Gamma \vdash \tau \lesssim \llbracket z : b \mid c \rrbracket$
 shows $\langle \Theta ; \mathcal{B} ; \Gamma \models c[z::=v]_{cv} \rangle$

proof –

obtain $z1$ and $b1$ and $c1$ where $*:\tau = \llbracket z1 : b1 \mid c1 \rrbracket \wedge \text{atom } z1 \# (c, v, \Gamma)$ using *obtain-fresh-z*
 by *metis*
 then have $b1 = b$ using *assms subtype-eq-base* by *metis*
 moreover then have $\Theta ; \mathcal{B} ; \Gamma \vdash v \Rightarrow \llbracket z1 : b \mid c1 \rrbracket$ using *assms ** by *auto*

moreover have $\Theta ; \mathcal{B} ; (z1, b, c1) \#_{\Gamma} \Gamma \models c[z::=[z1]^v]_{cv}$ **proof** –
have $\Theta ; \mathcal{B} ; (z1, b, c1[z1::=[z1]^v]_v) \#_{\Gamma} \Gamma \models c[z::=[z1]^v]_v$
using *subtype-valid[OF assms(2), of z1 z1 b c1 z c] * fresh-prodN (b1 = b) 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** $cs::branch\text{-}s$ **and** $x::x$ **and** $c::c$ **and** $b::b$ **and**
 $\Gamma_1::\Gamma$ **and** $\Gamma_2::\Gamma$ **and** $css::branch\text{-}list$
assumes $\Theta ; \mathcal{B} ; \Gamma_1 \vdash v \Rightarrow \tau$ **and** $\Theta ; \mathcal{B} ; \Gamma_1 \vdash \tau \lesssim \llbracket z : b \mid c \rrbracket$ **and**
 $atom\ z \not\# (x, v)$
shows $\Theta ; \Phi ; \mathcal{B} ; \Gamma ; \Delta \vdash s \Leftarrow \tau' \Longrightarrow$
 $\Gamma = (\Gamma_2 @ ((x, b, c[z::=[x]^v]_{cv}) \#_{\Gamma} \Gamma_1)) \Longrightarrow$
 $\Theta ; \Phi ; \mathcal{B} ; \Gamma[x::=v]_{\Gamma_v} ; \Delta[x::=v]_{\Delta_v} \vdash s[x::=v]_{sv} \Leftarrow \tau'[x::=v]_{\tau_v}$
and
 $\Theta ; \Phi ; \mathcal{B} ; \Gamma ; \Delta ; tid ; cons ; const ; v' \vdash cs \Leftarrow \tau' \Longrightarrow$
 $\Gamma = (\Gamma_2 @ ((x, b, c[z::=[x]^v]_{cv}) \#_{\Gamma} \Gamma_1)) \Longrightarrow$
 $\Theta ; \Phi ; \mathcal{B} ; \Gamma[x::=v]_{\Gamma_v} ; \Delta[x::=v]_{\Delta_v} ;$
 $tid ; cons ; const ; v'[x::=v]_{vv} \vdash cs[x::=v]_{sv} \Leftarrow \tau'[x::=v]_{\tau_v}$
and
 $\Theta ; \Phi ; \mathcal{B} ; \Gamma ; \Delta ; tid ; dclist ; v' \vdash css \Leftarrow \tau' \Longrightarrow$
 $\Gamma = (\Gamma_2 @ ((x, b, c[z::=[x]^v]_{cv}) \#_{\Gamma} \Gamma_1)) \Longrightarrow$
 $\Theta ; \Phi ; \mathcal{B} ; \Gamma[x::=v]_{\Gamma_v} ; \Delta[x::=v]_{\Delta_v} ; tid ; dclist ; v'[x::=v]_{vv} \vdash$
 $subst\text{-}branchlv\ css\ x\ v \Leftarrow \tau'[x::=v]_{\tau_v}$

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* $\Theta \ \mathcal{B} \ \Gamma \ \Delta \ \Phi \ v' \ \tau' \ \tau''$)

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 $\Theta ; \mathcal{B} ; \Gamma_1 \vdash \tau \lesssim \llbracket z : b \mid c \rrbracket$ **using** *check-valI* **by** *auto*

show $atom\ z \not\# (x, v)$ **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  $xa \ \Theta \ \Phi \ \mathcal{B} \ \Gamma \ \Delta \ ea \ \tau a \ za \ sa \ ba \ ca$ )
have *:(AS-let  $xa \ ea \ sa$ )[ $x::=v$ ] $_{sv}$ =(AS-let  $xa \ (ea[x::=v]_{ev}) \ sa[x::=v]_{sv}$ )
  using subst-sv.simps < atom  $xa \ \# \ x$  > < atom  $xa \ \# \ v$  > by auto
show ?case unfolding * proof

show atom  $xa \ \# \ (\Theta, \Phi, \mathcal{B}, \Gamma[x::=v]_{\Gamma v}, \Delta[x::=v]_{\Delta v}, ea[x::=v]_{ev}, \tau a[x::=v]_{\tau v})$ 
  by(subst-tuple-mth add: check-letI)

show atom  $za \ \# \ (xa, \Theta, \Phi, \mathcal{B}, \Gamma[x::=v]_{\Gamma v}, \Delta[x::=v]_{\Delta v}, ea[x::=v]_{ev},$ 
   $\tau a[x::=v]_{\tau v}, sa[x::=v]_{sv})$ 
  by(subst-tuple-mth add: check-letI)

show  $\Theta; \Phi; \mathcal{B}; \Gamma[x::=v]_{\Gamma v}; \Delta[x::=v]_{\Delta v} \vdash$ 
   $ea[x::=v]_{ev} \Rightarrow \{ \mid za : ba \mid ca[x::=v]_{cv} \}$ 
proof –
  have  $\Theta; \Phi; \mathcal{B}; \Gamma_2[x::=v]_{\Gamma v} @ \Gamma_1; \Delta[x::=v]_{\Delta v} \vdash$ 
   $ea[x::=v]_{ev} \Rightarrow \{ \mid za : ba \mid ca \} [x::=v]_{\tau v}$ 
  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  $\Theta; \Phi; \mathcal{B}; (xa, ba, ca[x::=v]_{cv}[za::=V-var \ x]_v) \#_{\Gamma} \Gamma[x::=v]_{\Gamma v};$ 
   $\Delta[x::=v]_{\Delta v} \vdash sa[x::=v]_{sv} \Leftarrow \tau a[x::=v]_{\tau v}$ 
proof –
  have  $\Theta; \Phi; \mathcal{B}; ((xa, ba, ca[za::=V-var \ x]_v) \#_{\Gamma} \Gamma)[x::=v]_{\Gamma v};$ 
   $\Delta[x::=v]_{\Delta v} \vdash sa[x::=v]_{sv} \Leftarrow \tau a[x::=v]_{\tau v}$ 
  apply(rule check-letI(23)[of  $(xa, ba, ca[za::=V-var \ x]_{cv}) \#_{\Gamma} \Gamma_2$ ])
  by(metis check-letI append-g.simps subst-defs)+

moreover have  $(xa, ba, ca[x::=v]_{cv}[za::=V-var \ x]_{cv}) \#_{\Gamma} \Gamma[x::=v]_{\Gamma v} =$ 
   $((xa, ba, ca[za::=V-var \ x]_{cv}) \#_{\Gamma} \Gamma)[x::=v]_{\Gamma v}$ 
  using subst-cv-commute subst-gv.simps check-letI
  by (metis ms-fresh-all(39) ms-fresh-all(49) subst-cv-commute-full)
ultimately show ?thesis
  using subst-defs by auto
qed
qed
next
case (check-assertI  $xa \ \Theta \ \Phi \ \mathcal{B} \ \Gamma \ \Delta \ ca \ \tau \ s$ )
show ?case unfolding subst-sv.simps proof
  show <atom  $xa \ \# \ (\Theta, \Phi, \mathcal{B}, \Gamma[x::=v]_{\Gamma v}, \Delta[x::=v]_{\Delta v}, ca[x::=v]_{cv}, \tau[x::=v]_{\tau v}, s[x::=v]_{sv})$ >
  by(subst-tuple-mth add: check-assertI)
  have  $xa \neq x$  using check-assertI by fastforce
  thus <  $\Theta; \Phi; \mathcal{B}; (xa, B\text{-bool}, ca[x::=v]_{cv}) \#_{\Gamma} \Gamma[x::=v]_{\Gamma v}; \Delta[x::=v]_{\Delta v} \vdash s[x::=v]_{sv} \Leftarrow \tau[x::=v]_{\tau v}$  >
  using check-assertI(12)[of  $(xa, B\text{-bool}, c) \#_{\Gamma} \Gamma_2 \ x \ v]$  check-assertI subst-gv.simps append-g.simps
by metis
  have  $\langle \Theta; \mathcal{B}; \Gamma_2[x::=v]_{\Gamma v} @ \Gamma_1 \models ca[x::=v]_{cv} \rangle$  proof(rule subst-valid )
  show  $\langle \Theta; \mathcal{B}; \Gamma_1 \models c[z::=v]_{cv} \rangle$  using infer-v-c-valid check-assertI by metis
  show <  $\Theta; \mathcal{B}; \Gamma_1 \vdash_{wf} v : b$  > using check-assertI infer-v-wf b-of.simps subtype-eq-base
  by (metis subtype-eq-base2)

```

```

show  $\langle \Theta ; \mathcal{B} \vdash_{wf} \Gamma_1 \rangle$  using check-assertI infer-v-wf by metis
have  $\Theta ; \mathcal{B} \vdash_{wf} \Gamma_2 @ (x, b, c[z::=[x]^v]_{cv}) \#_{\Gamma} \Gamma_1$  using check-assertI wfX-wfY by metis
thus  $\langle atom\ x \# \Gamma_1 \rangle$  using check-assertI wfG-suffix wfG-elim by metis

moreover have  $\Theta ; \mathcal{B} ; \Gamma_1 \vdash_{wf} \{ z : b \mid c \}$  using subtype-wfT check-assertI by metis
moreover have  $x \neq z$  using fresh-Pair check-assertI fresh-x-neq by metis
ultimately show  $\langle atom\ x \# c \rangle$  using check-assertI wfT-fresh-c by metis

show  $\langle \Theta ; \mathcal{B} \vdash_{wf} \Gamma_2 @ (x, b, c[z::=[x]^v]_{cv}) \#_{\Gamma} \Gamma_1 \rangle$  using check-assertI wfX-wfY by metis
show  $\langle \Theta ; \mathcal{B} ; \Gamma_2 @ (x, b, c[z::=[x]^v]_{cv}) \#_{\Gamma} \Gamma_1 \models ca \rangle$  using check-assertI by auto
qed
thus  $\langle \Theta ; \mathcal{B} ; \Gamma[x::=v]_{\Gamma v} \models ca[x::=v]_{cv} \rangle$  using check-assertI
proof –
  show ?thesis
  by (metis (no-types)  $\langle \Gamma = \Gamma_2 @ (x, b, c[z::=[x]^v]_{cv}) \#_{\Gamma} \Gamma_1 \rangle \langle \Theta ; \mathcal{B} ; \Gamma \models ca \rangle \langle \Theta ; \mathcal{B} ; \Gamma_2[x::=v]_{\Gamma v} @ \Gamma_1 \models ca[x::=v]_{cv} \rangle$  subst-g-inside valid-g-wf)
qed

have  $\Theta ; \mathcal{B} ; \Gamma_1 \vdash_{wf} v : b$  using infer-v-wf b-of.simps check-assertI
by (metis subtype-eq-base2)
thus  $\langle \Theta ; \mathcal{B} ; \Gamma[x::=v]_{\Gamma v} \vdash_{wf} \Delta[x::=v]_{\Delta v} \rangle$  using wf-subst2(6) check-assertI by metis
qed
next
case (check-branch-list-consI  $\Theta \Phi \mathcal{B} \Gamma \Delta tid\ dclist\ vv\ cs\ \tau\ css$ )
show ?case unfolding * using subst-sv.simps check-branch-list-consI by simp
next
case (check-branch-list-finalI  $\Theta \Phi \mathcal{B} \Gamma \Delta tid\ dclist\ v\ cs\ \tau$ )
show ?case unfolding * using subst-sv.simps check-branch-list-finalI by simp
next
case (check-branch-s-branchI  $\Theta \mathcal{B} \Gamma \Delta \tau\ const\ xa\ \Phi\ tid\ cons\ va\ sa$ )
hence  $*(AS\text{-}branch\ cons\ xa\ sa)[x::=v]_{sv} = (AS\text{-}branch\ cons\ xa\ sa[x::=v]_{sv})$  using subst-branchv.simps
fresh-Pair by metis
show ?case unfolding * proof

show  $\Theta ; \mathcal{B} ; \Gamma[x::=v]_{\Gamma v} \vdash_{wf} \Delta[x::=v]_{\Delta v}$ 
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  $\Theta ; \mathcal{B} ; \Gamma[x::=v]_{\Gamma v} \vdash_{wf} \tau[x::=v]_{\tau v}$ 
using wf-subst check-branch-s-branchI subtype-eq-base2 b-of.simps infer-v-wf by metis

show wft: $\Theta ; \{||\} ; GNil \vdash_{wf} const$  using check-branch-s-branchI by metis

show  $atom\ xa \# (\Theta, \Phi, \mathcal{B}, \Gamma[x::=v]_{\Gamma v}, \Delta[x::=v]_{\Delta v}, tid, cons, const, va[x::=v]_{vv}, \tau[x::=v]_{\tau v})$ 
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  $\Theta ; \Phi ; \mathcal{B} ; ((xa, b\text{-of}\ const, CE\text{-val}\ va == CE\text{-val}(V\text{-cons}\ tid\ cons\ (V\text{-var}\ xa)) \text{ AND } c\text{-of}\ const\ xa) \#_{\Gamma} \Gamma)[x::=v]_{\Gamma v} ; \Delta[x::=v]_{\Delta v} \vdash sa[x::=v]_{sv} \Leftarrow \tau[x::=v]_{\tau v}$ 
using check-branch-s-branchI by (metis append-g.simps(2))

```

moreover have $(xa, b\text{-of } \text{const}, CE\text{-val } va[x::=v]_{vv} == CE\text{-val } (V\text{-cons tid cons } (V\text{-var } xa))$
 $AND\ c\text{-of } (\text{const})\ xa) \#_{\Gamma} \Gamma[x::=v]_{\Gamma v} =$
 $((xa, b\text{-of } \text{const}, CE\text{-val } va == CE\text{-val } (V\text{-cons tid cons } (V\text{-var } xa))\ AND\ c\text{-of } \text{const}$
 $xa) \#_{\Gamma} \Gamma)[x::=v]_{\Gamma v}$
proof –
have $*:xa \neq x$ **using** *check-branch-s-branchI fresh-at-base by metis*
have $atom\ x \# \text{const}$ **using** *wfT-nil-suppl[OF wft] fresh-def by auto*
hence $atom\ x \# (\text{const}, xa)$ **using** *fresh-at-base wfT-nil-suppl[OF wft] fresh-Pair fresh-def * by auto*
moreover hence $(c\text{-of } (\text{const})\ xa)[x::=v]_{cv} = c\text{-of } (\text{const})\ xa$
using *c-of-fresh[of x const xa] forget-subst-cv wfT-nil-suppl wft by metis*
moreover hence $(V\text{-cons tid cons } (V\text{-var } xa))[x::=v]_{vv} = (V\text{-cons tid cons } (V\text{-var } xa))$ **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 $\Theta ; \Phi ; \mathcal{B} ; (xa, b\text{-of } \text{const}, CE\text{-val } va[x::=v]_{vv} == CE\text{-val } (V\text{-cons tid cons } (V\text{-var } xa))$
 $AND\ c\text{-of } \text{const } xa) \#_{\Gamma} \Gamma[x::=v]_{\Gamma v} ; \Delta[x::=v]_{\Delta v} \vdash sa[x::=v]_{sv} \Leftarrow \tau[x::=v]_{\tau v}$
by *metis*
qed

next

case $(check\text{-let2I } xa\ \Theta\ \Phi\ \mathcal{B}\ G\ \Delta\ t\ s1\ \tau a\ s2)$
hence $*(AS\text{-let2 } xa\ t\ s1\ s2)[x::=v]_{sv} = (AS\text{-let2 } xa\ t[x::=v]_{\tau v}\ (s1[x::=v]_{sv}\ s2[x::=v]_{sv}))$ **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 $atom\ xa \# (\Theta, \Phi, \mathcal{B}, G[x::=v]_{\Gamma v}, \Delta[x::=v]_{\Delta v}, t[x::=v]_{\tau v}, s1[x::=v]_{sv}, \tau a[x::=v]_{\tau v})$
by *(subst-tuple-mth add: check-let2I)*
show $\Theta ; \Phi ; \mathcal{B} ; G[x::=v]_{\Gamma v} ; \Delta[x::=v]_{\Delta v} \vdash s1[x::=v]_{sv} \Leftarrow t[x::=v]_{\tau v}$ **using** *check-let2I by metis*

have $\Theta ; \Phi ; \mathcal{B} ; ((xa, b\text{-of } t, c\text{-of } t\ xa) \#_{\Gamma} G)[x::=v]_{\Gamma v} ; \Delta[x::=v]_{\Delta v} \vdash s2[x::=v]_{sv} \Leftarrow \tau a[x::=v]_{\tau v}$
proof *(rule check-let2I(14))*
show $\langle (xa, b\text{-of } t, c\text{-of } t\ xa) \#_{\Gamma} G = (((xa, b\text{-of } t, c\text{-of } t\ xa) \#_{\Gamma} \Gamma_2)) @ (x, b, c[z::=[x]_{cv}]) \#_{\Gamma} \Gamma_1 \rangle$
using *check-let2I append-g.simps by metis*
show $\langle \Theta ; \mathcal{B} ; \Gamma_1 \vdash v \Rightarrow \tau \rangle$ **using** *check-let2I by metis*
show $\langle \Theta ; \mathcal{B} ; \Gamma_1 \vdash \tau \lesssim \llbracket z : b \mid c \rrbracket \rangle$ **using** *check-let2I by metis*
show $\langle atom\ z \# (x, v) \rangle$ **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 $\Theta ; \Phi ; \mathcal{B} ; (xa, b\text{-of } t[x::=v]_{\tau v}, c\text{-of } t[x::=v]_{\tau v}\ xa) \#_{\Gamma} G[x::=v]_{\Gamma v} ; \Delta[x::=v]_{\Delta v}$
 $\vdash s2[x::=v]_{sv} \Leftarrow \tau a[x::=v]_{\tau v}$
using *check-let2I(14) subst-gv.simps (xa ≠ x) b-of.simps by metis*
qed

next

case $(check\text{-varI } u\ \Theta\ \Phi\ \mathcal{B}\ \Gamma\ \Delta\ \tau'\ va\ \tau''\ s)$

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 $\Theta ; \Phi ; \mathcal{B} ; \text{subst-gv } \Gamma x v ; \Delta[x::=v]_{\Delta_v} \vdash AS\text{-var } u \tau'[x::=v]_{\tau_v} (va[x::=v]_{vv}) (\text{subst-sv } s x v)$
 $\Leftarrow \tau''[x::=v]_{\tau_v}$

proof(*rule Typing.check-varI*)
show $\text{atom } u \# (\Theta, \Phi, \mathcal{B}, \Gamma[x::=v]_{\Gamma_v}, \Delta[x::=v]_{\Delta_v}, \tau'[x::=v]_{\tau_v}, va[x::=v]_{vv}, \tau''[x::=v]_{\tau_v})$
by(*subst-tuple-mth add: check-varI*)
show $\Theta ; \mathcal{B} ; \Gamma[x::=v]_{\Gamma_v} \vdash va[x::=v]_{vv} \Leftarrow \tau'[x::=v]_{\tau_v}$ **using** *check-varI subst-infer-check-v *** **by** *metis*
show $\Theta ; \Phi ; \mathcal{B} ; \text{subst-gv } \Gamma x v ; (u, \tau'[x::=v]_{\tau_v}) \#_{\Delta} \Delta[x::=v]_{\Delta_v} \vdash s[x::=v]_{sv} \Leftarrow \tau''[x::=v]_{\tau_v}$

proof –
have $\text{wfD } \Theta \mathcal{B} (\Gamma_2 @ (x, b, c[z::=[x]^v]_{cv}) \#_{\Gamma} \Gamma_1) ((u, \tau') \#_{\Delta} \Delta)$ **using** *check-varI check-s-wf* **by** *meson*
moreover **have** $\text{wfV } \Theta \mathcal{B} \Gamma_1 v (b\text{-of } \tau)$ **using** *infer-v-wf check-varI(6) check-varI* **by** *metis*
have $\text{wfD } \Theta \mathcal{B} (\Gamma[x::=v]_{\Gamma_v}) ((u, \tau'[x::=v]_{\tau_v}) \#_{\Delta} \Delta[x::=v]_{\Delta_v})$ **proof**(*subst subst-dv.simps(2)[symmetric]*),
*subst **, rule wfD-subst*
show $\Theta ; \mathcal{B} ; \Gamma_1 \vdash v \Rightarrow \tau$ **using** *check-varI* **by** *auto*
show $\Theta ; \mathcal{B} ; \Gamma_2 @ (x, b, c[z::=[x]^v]_{cv}) \#_{\Gamma} \Gamma_1 \vdash_{wf} (u, \tau') \#_{\Delta} \Delta$ **using** *check-varI check-s-wf* **by** *simp*
show $b\text{-of } \tau = b$ **using** *check-varI subtype-eq-base2 b-of.simps* **by** *auto*
qed
thus *?thesis* **using** *check-varI* **by** *auto*
qed
qed
moreover **have** $\text{atom } u \# (x, v)$ **using** *u-fresh-xv* **by** *auto*
ultimately **show** *?case* **using** *subst-sv.simps(7)* **by** *auto*

next
case (*check-assignI P Φ B Γ Δ u τ1 v' z1 τ'*)

have $\text{wfG } P \mathcal{B} \Gamma$ **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 $P ; \Phi ; \mathcal{B} ; \Gamma[x::=v]_{\Gamma_v} ; \Delta[x::=v]_{\Delta_v} \vdash AS\text{-assign } u (v'[x::=v]_{vv}) \Leftarrow \tau'[x::=v]_{\tau_v}$
proof(*rule Typing.check-assignI*)
show $P \vdash_{wf} \Phi$ **using** *check-assignI* **by** *auto*
show $\text{wfD } P \mathcal{B} (\Gamma[x::=v]_{\Gamma_v}) \Delta[x::=v]_{\Delta_v}$ **using** *wf-subst(15)[OF check-assignI(2)] gs infer-v-v-wf*
check-assignI b-of.simps subtype-eq-base2 **by** *metis*
thus $(u, \tau1[x::=v]_{\tau_v}) \in \text{setD } \Delta[x::=v]_{\Delta_v}$ **using** *check-assignI subst-dv-member* **by** *metis*
thus $P ; \mathcal{B} ; \Gamma[x::=v]_{\Gamma_v} \vdash v'[x::=v]_{vv} \Leftarrow \tau1[x::=v]_{\tau_v}$ **using** *subst-infer-check-v check-assignI* **gs** **by** *metis*

have $P ; \mathcal{B} ; \Gamma_2[x::=v]_{\Gamma_v} @ \Gamma_1 \vdash \{ z : B\text{-unit} \mid \text{TRUE} \} [x::=v]_{\tau_v} \lesssim \tau'[x::=v]_{\tau_v}$ **proof**(*rule*
subst-subtype-tau)
show $P ; \mathcal{B} ; \Gamma_1 \vdash v \Rightarrow \tau$ **using** *check-assignI* **by** *auto*
show $P ; \mathcal{B} ; \Gamma_1 \vdash \tau \lesssim \{ z : b \mid c \}$ **using** *check-assignI* **by** *meson*
show $P ; \mathcal{B} ; \Gamma_2 @ (x, b, c[z::=[x]^v]_{cv}) \#_{\Gamma} \Gamma_1 \vdash \{ z : B\text{-unit} \mid \text{TRUE} \} \lesssim \tau'$ **using** *check-assignI*
by (*metis AbsI-eq-iff(3) τ.eq-iff c.fresh(1) c.perm-simps(1)*)
show $\text{atom } z \# (x, v)$ **using** *check-assignI* **by** *auto*
qed
moreover **have** $\{ z : B\text{-unit} \mid \text{TRUE} \} [x::=v]_{\tau_v} = \{ z : B\text{-unit} \mid \text{TRUE} \}$ **using** *subst-tv.simps*
subst-cv.simps check-assignI **by** *presburger*

ultimately show $P ; \mathcal{B} ; \Gamma[x::=v]_{\Gamma_v} \vdash \llbracket z : B\text{-unit} \mid \text{TRUE} \rrbracket \lesssim \tau'[x::=v]_{\tau_v}$ **using** *gs* **by** *auto*
qed
thus *?case* **using** *subst-sv.simps(5)* **by** *auto*

next
case (*check-whileI* $\Theta \Phi \mathcal{B} \Gamma \Delta s1 z' s2 \tau'$)
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*: $(\llbracket z : B\text{-unit} \mid \text{TRUE} \rrbracket)[x::=v]_{\tau_v} = (\llbracket z : B\text{-unit} \mid \text{TRUE} \rrbracket)$ **by** (*auto simp add: subst-sv.simps check-whileI*)
moreover **have** $(\llbracket z : B\text{-unit} \mid \text{TRUE} \rrbracket) = (\llbracket z' : B\text{-unit} \mid \text{TRUE} \rrbracket)$ **using** *type-eq-flip c.fresh flip-fresh-fresh* **by** *metis*
ultimately **have** *teq2*: $(\llbracket z' : B\text{-unit} \mid \text{TRUE} \rrbracket)[x::=v]_{\tau_v} = (\llbracket z' : B\text{-unit} \mid \text{TRUE} \rrbracket)$ **by** *metis*

hence $\Theta ; \Phi ; \mathcal{B} ; \Gamma[x::=v]_{\Gamma_v} ; \Delta[x::=v]_{\Delta_v} \vdash s1[x::=v]_{sv} \Leftarrow \llbracket z' : B\text{-bool} \mid \text{TRUE} \rrbracket$ **using** *check-whileI subst-sv.simps subst-top-eq* **by** *metis*
moreover **have** $\Theta ; \Phi ; \mathcal{B} ; \Gamma[x::=v]_{\Gamma_v} ; \Delta[x::=v]_{\Delta_v} \vdash s2[x::=v]_{sv} \Leftarrow \llbracket z' : B\text{-unit} \mid \text{TRUE} \rrbracket$ **using** *check-whileI subst-top-eq* **by** *metis*
moreover **have** $\Theta ; \mathcal{B} ; \Gamma[x::=v]_{\Gamma_v} \vdash \llbracket z' : B\text{-unit} \mid \text{TRUE} \rrbracket \lesssim \tau'[x::=v]_{\tau_v}$ **proof** –
have $\Theta ; \mathcal{B} ; \Gamma_2[x::=v]_{\Gamma_v} @ \Gamma_1 \vdash \llbracket z' : B\text{-unit} \mid \text{TRUE} \rrbracket[x::=v]_{\tau_v} \lesssim \tau'[x::=v]_{\tau_v}$ **proof** (*rule subst-subtype-tau*)
show $\Theta ; \mathcal{B} ; \Gamma_1 \vdash v \Rightarrow \tau$ **by** (*auto simp add: check-whileI*)
show $\Theta ; \mathcal{B} ; \Gamma_1 \vdash \tau \lesssim \llbracket z : b \mid c \rrbracket$ **by** (*auto simp add: check-whileI*)
show $\Theta ; \mathcal{B} ; \Gamma_2 @ (x, b, c[z::=[x]^v]_{cv}) \#_{\Gamma} \Gamma_1 \vdash \llbracket z' : B\text{-unit} \mid \text{TRUE} \rrbracket \lesssim \tau'$ **using** *check-whileI*
by *metis*
show *atom* $z \# (x, v)$ **by** (*auto simp add: check-whileI*)
qed
thus *?thesis* **using** *teq2 *** **by** *auto*
qed

ultimately **have** $\Theta ; \Phi ; \mathcal{B} ; \Gamma[x::=v]_{\Gamma_v} ; \Delta[x::=v]_{\Delta_v} \vdash \text{AS-while } s1[x::=v]_{sv} s2[x::=v]_{sv} \Leftarrow \tau'[x::=v]_{\tau_v}$
using *Typing.check-whileI* **by** *metis*
then **show** *?case* **using** *subst-sv.simps* **by** *metis*

next
case (*check-seqI* $P \Phi \mathcal{B} \Gamma \Delta s1 z s2 \tau$)
hence $P ; \Phi ; \mathcal{B} ; \Gamma[x::=v]_{\Gamma_v} ; \Delta[x::=v]_{\Delta_v} \vdash \text{AS-seq } (s1[x::=v]_{sv}) (s2[x::=v]_{sv}) \Leftarrow \tau[x::=v]_{\tau_v}$
using *Typing.check-seqI subst-top-eq check-seqI* **by** *metis*
then **show** *?case* **using** *subst-sv.simps* **by** *metis*

next
case (*check-caseI* $\Theta \Phi \mathcal{B} \Gamma \Delta \text{tid dclist } v' \text{cs } \tau \text{za}$)

have *wf*: *wfG* $\Theta \mathcal{B} \Gamma$ **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 $\Theta ; \Phi ; \mathcal{B} ; \Gamma[x::=v]_{\Gamma_v} ; \Delta[x::=v]_{\Delta_v} \vdash \text{AS-match } (v'[x::=v]_{vv}) (\text{subst-branchlv cs } x \text{ } v) \Leftarrow \tau[x::=v]_{\tau_v}$ **proof** (*rule Typing.check-caseI*)
show *check-branch-list* $\Theta \Phi \mathcal{B} (\Gamma[x::=v]_{\Gamma_v}) \Delta[x::=v]_{\Delta_v} \text{tid dclist } v'[x::=v]_{vv} (\text{subst-branchlv cs } x \text{ } v)$
using *check-caseI* **by** *auto*
show *AF-typedef* *tid dclist* $\in \text{set } \Theta$ **using** *check-caseI* **by** *auto*
show $\Theta ; \mathcal{B} ; \Gamma[x::=v]_{\Gamma_v} \vdash v'[x::=v]_{vv} \Leftarrow \llbracket \text{za} : B\text{-id tid} \mid \text{TRUE} \rrbracket$ **proof** –
have $\Theta ; \mathcal{B} ; \Gamma_2 @ (x, b, c[z::=[x]^v]_{cv}) \#_{\Gamma} \Gamma_1 \vdash v' \Leftarrow \llbracket \text{za} : B\text{-id tid} \mid \text{TRUE} \rrbracket$

using *check-caseI* **by** *argo*
hence $\Theta ; \mathcal{B} ; \Gamma_2[x::=v]_{\Gamma_v} @ \Gamma_1 \vdash v'[x::=v]_{vv} \Leftarrow (\llbracket za : B\text{-id tid} \mid TRUE \rrbracket)[x::=v]_{\tau_v}$
using *check-caseI subst-infer-check-v[OF check-caseI(7) - check-caseI(8) check-caseI(9)]* **by** *meson*
moreover have $(\llbracket za : B\text{-id tid} \mid TRUE \rrbracket) = ((\llbracket za : B\text{-id tid} \mid TRUE \rrbracket)[x::=v]_{\tau_v})$
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-iffI* $z' \Theta \Phi \mathcal{B} \Gamma \Delta va s1 s2 \tau'$)
show *?case* **unfolding** *subst-sv.simps* **proof**
show $\langle atom\ z' \# (\Theta, \Phi, \mathcal{B}, \Gamma[x::=v]_{\Gamma_v}, \Delta[x::=v]_{\Delta_v}, va[x::=v]_{vv}, s1[x::=v]_{sv}, s2[x::=v]_{sv}, \tau'[x::=v]_{\tau_v}) \rangle$

by(*subst-tuple-mth add: check-iffI*)
have $*: \llbracket z' : B\text{-bool} \mid TRUE \rrbracket[x::=v]_{\tau_v} = \llbracket z' : B\text{-bool} \mid TRUE \rrbracket$ **using** *subst-tv.simps check-iffI*
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-iffI check-v-wf* **by** *metis*
show $\langle \Theta ; \mathcal{B} ; \Gamma[x::=v]_{\Gamma_v} \vdash va[x::=v]_{vv} \Leftarrow \llbracket z' : B\text{-bool} \mid TRUE \rrbracket \rangle$
proof(*subst *[symmetric], rule subst-infer-check-vI[where $\Gamma_1=\Gamma_2$ and $\Gamma_2=\Gamma_1$]*)
show $\Gamma = \Gamma_2 @ ((x, b, c[z::=[x]_{cv}]) \#_{\Gamma} \Gamma_1)$ **using** *check-iffI* **by** *metis*
show $\langle \Theta ; \mathcal{B} ; \Gamma_1 \vdash v \Rightarrow \tau \rangle$ **using** *check-iffI* **by** *metis*
show $\langle \Theta ; \mathcal{B} ; \Gamma \vdash va \Leftarrow \llbracket z' : B\text{-bool} \mid TRUE \rrbracket \rangle$ **using** *check-iffI* **by** *metis*
show $\langle \Theta ; \mathcal{B} ; \Gamma_1 \vdash \tau \lesssim \llbracket z : b \mid c \rrbracket \rangle$ **using** *check-iffI* **by** *metis*
show $\langle atom\ z \# (x, v) \rangle$ **using** *check-iffI* **by** *metis*
qed

have $\llbracket z' : b\text{-of } \tau'[x::=v]_{\tau_v} \mid [va[x::=v]_{vv}]^{ce} \rrbracket == [[L\text{-true}]^v]^{ce} \text{ IMP } c\text{-of } \tau'[x::=v]_{\tau_v} z' \rrbracket$
 $= \llbracket z' : b\text{-of } \tau' \mid [va]^{ce} \rrbracket == [[L\text{-true}]^v]^{ce} \text{ IMP } c\text{-of } \tau' z' \rrbracket[x::=v]_{\tau_v}$
by(*simp add: subst-tv.simps fresh-Pair check-iffI b-of-subst subst-v-c-of*)

thus $\langle \Theta ; \Phi ; \mathcal{B} ; \Gamma[x::=v]_{\Gamma_v} ; \Delta[x::=v]_{\Delta_v} \vdash s1[x::=v]_{sv} \Leftarrow \llbracket z' : b\text{-of } \tau'[x::=v]_{\tau_v} \mid [va[x::=v]_{vv}]^{ce} \rrbracket$
 $]^{ce} == [[L\text{-true}]^v]^{ce} \text{ IMP } c\text{-of } \tau'[x::=v]_{\tau_v} z' \rrbracket$
using *check-iffI* **by** *metis*
have $\llbracket z' : b\text{-of } \tau'[x::=v]_{\tau_v} \mid [va[x::=v]_{vv}]^{ce} \rrbracket == [[L\text{-false}]^v]^{ce} \text{ IMP } c\text{-of } \tau'[x::=v]_{\tau_v} z' \rrbracket$
 $= \llbracket z' : b\text{-of } \tau' \mid [va]^{ce} \rrbracket == [[L\text{-false}]^v]^{ce} \text{ IMP } c\text{-of } \tau' z' \rrbracket[x::=v]_{\tau_v}$
by(*simp add: subst-tv.simps fresh-Pair check-iffI b-of-subst subst-v-c-of*)
thus $\langle \Theta ; \Phi ; \mathcal{B} ; \Gamma[x::=v]_{\Gamma_v} ; \Delta[x::=v]_{\Delta_v} \vdash s2[x::=v]_{sv} \Leftarrow \llbracket z' : b\text{-of } \tau'[x::=v]_{\tau_v} \mid [va[x::=v]_{vv}]^{ce} \rrbracket$
 $]^{ce} == [[L\text{-false}]^v]^{ce} \text{ IMP } c\text{-of } \tau'[x::=v]_{\tau_v} z' \rrbracket$
using *check-iffI* **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 $\Theta ; \mathcal{B} ; \Gamma_1 \vdash v \Leftarrow \llbracket z : b \mid c \rrbracket$ **and** $atom\ z \# (x, v)$
and $check\text{-}s\ \Theta\ \Phi\ \mathcal{B}\ \Gamma\ \Delta\ s\ \tau'$ **and** $\Gamma = (\Gamma_2 @ ((x, b, c[z::=[x]_{cv}]) \#_{\Gamma} \Gamma_1))$
shows $check\text{-}s\ \Theta\ \Phi\ \mathcal{B}\ (subst\text{-}gv\ \Gamma\ x\ v)\ \Delta[x::=v]_{\Delta_v}\ (s[x::=v]_{sv})\ (subst\text{-}tv\ \tau'\ x\ v)$
proof –
obtain τ **where** $\Theta ; \mathcal{B} ; \Gamma_1 \vdash v \Rightarrow \tau \wedge \Theta ; \mathcal{B} ; \Gamma_1 \vdash \tau \lesssim \llbracket z : b \mid c \rrbracket$ **using** *check-v-elimss 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-s* **and** *x::x* **and** *c::c* **and** *b::b* **and** $\Gamma::\Gamma$ **and** $\Gamma'::\Gamma$ **and** *css::branch-list*
shows *check-s* $\Theta \Phi \mathcal{B} G \Delta \ s \ t1 \implies \Theta ; \mathcal{B} ; G \vdash t1 \lesssim t2 \implies \text{check-s } \Theta \Phi \mathcal{B} G \Delta \ s \ t2$ **and**
check-branch-s $\Theta \Phi \mathcal{B} G \Delta \ tid \ cons \ const \ v \ cs \ t1 \implies \Theta ; \mathcal{B} ; G \vdash t1 \lesssim t2 \implies \text{check-branch-s}$
 $\Theta \Phi \mathcal{B} G \Delta \ tid \ cons \ const \ v \ cs \ t2$ **and**
check-branch-list $\Theta \Phi \mathcal{B} G \Delta \ tid \ dclist \ v \ css \ t1 \implies \Theta ; \mathcal{B} ; G \vdash t1 \lesssim t2 \implies \text{check-branch-list}$
 $\Theta \Phi \mathcal{B} G \Delta \ tid \ dclist \ v \ css \ t2$
proof(*induct arbitrary: t2 and t2 rule: check-s-check-branch-s-check-branch-list.inducts*)
case (*check-valI* $\Theta \Phi \mathcal{B} \Gamma \Delta \ \Phi \ v \ \tau' \ \tau$)
hence $\Theta ; \mathcal{B} ; \Gamma \vdash \tau' \lesssim t2$ **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* $x \# (\Theta, \Phi, \mathcal{B}, \Gamma, \Delta, e, t2)$ **using** *check-letI subtype-fresh-tau fresh-prodN* **by** *metis*
thm *subtype-fresh-tau*
show *atom* $z \# (x, \Theta, \Phi, \mathcal{B}, \Gamma, \Delta, e, t2, s)$ **using** *check-letI(2) subtype-fresh-tau[of z \tau \Gamma - - t2]*
fresh-prodN check-letI(6) **by** *auto*
show $\Theta ; \Phi ; \mathcal{B} ; \Gamma ; \Delta \vdash e \Rightarrow \{ z : b \mid c \}$ **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** *toSet* $\Gamma \subseteq \text{toSet } ((x, b, c[z::=[x]^v]_v) \#_{\Gamma} \Gamma)$ **by** *auto*
ultimately **have** $\Theta ; \mathcal{B} ; (x, b, c[z::=[x]^v]_v) \#_{\Gamma} \Gamma \vdash \tau \lesssim t2$ **using** *subtype-weakening[OF check-letI(6)]* **by** *auto*

thus $\Theta ; \Phi ; \mathcal{B} ; (x, b, c[z::=[x]^v]_v) \#_{\Gamma} \Gamma ; \Delta \vdash s \Leftarrow t2$ **using** *check-letI subst-defs* **by** *metis*
qed

next

next

case (*check-branch-list-consI* $\Theta \Phi \mathcal{B} \Gamma \Delta \ tid \ dclist \ v \ cs \ \tau \ css$)
then show *?case* **using** *Typing.check-branch-list-consI* **by** *auto*

next

case (*check-branch-list-finalI* $\Theta \Phi \mathcal{B} \Gamma \Delta \ tid \ dclist \ v \ cs \ \tau$)
then show *?case* **using** *Typing.check-branch-list-finalI* **by** *auto*

next

case (*check-branch-s-branchI* $\Theta \mathcal{B} \Gamma \Delta \ \tau \ const \ x \ \Phi \ tid \ cons \ v \ s$)
show *?case* **proof**
have *atom* $x \# t2$ **using** *subtype-fresh-tau[of x \tau] check-branch-s-branchI(5,8) fresh-prodN* **by** *metis*
thus *atom* $x \# (\Theta, \Phi, \mathcal{B}, \Gamma, \Delta, tid, cons, const, v, t2)$ **using** *check-branch-s-branchI fresh-prodN* **by** *metis*
show *wfT* $\Theta \mathcal{B} \Gamma \ t2$ **using** *subtype-wf check-branch-s-branchI* **by** *meson*
show $\Theta ; \Phi ; \mathcal{B} ; (x, b\text{-of } const, CE\text{-val } v == CE\text{-val}(V\text{-cons } tid \ cons \ (V\text{-var } x)) \text{ AND } c\text{-of } const \ x) \#_{\Gamma} \Gamma ; \Delta \vdash s \Leftarrow t2$ **proof** –
have *wfG* $\Theta \mathcal{B} ((x, b\text{-of } const, CE\text{-val } v == CE\text{-val}(V\text{-cons } tid \ cons \ (V\text{-var } x)) \text{ AND } c\text{-of } const \ x) \#_{\Gamma} \Gamma)$ **using** *check-s-wf check-branch-s-branchI* **by** *metis*

moreover have $toSet \Gamma \subseteq toSet ((x, b-of \text{const}, CE-val \ v == CE-val \ (V-cons \ tid \ cons \ (V-var \ x))) \text{ AND } c-of \text{const} \ x) \#_{\Gamma} \Gamma)$ **by auto**
hence $\Theta ; \mathcal{B} ; ((x, b-of \text{const}, CE-val \ v == CE-val \ (V-cons \ tid \ cons \ (V-var \ x))) \text{ AND } c-of \text{const} \ x) \#_{\Gamma} \Gamma) \vdash \tau \lesssim t2$
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-iffI* $z \ \Theta \ \Phi \ \mathcal{B} \ \Gamma \ \Delta \ v \ s1 \ s2 \ \tau$)
show *?case proof(rule Typing.check-iffI)*
have $*:atom \ z \ \# \ t2$ **using** *subtype-fresh-tau[of z τ Γ] check-iffI fresh-prodN by auto*
thus $\langle atom \ z \ \# \ (\Theta, \Phi, \mathcal{B}, \Gamma, \Delta, v, s1, s2, t2) \rangle$ **using** *check-iffI fresh-prodN by auto*
show $\langle \Theta ; \mathcal{B} ; \Gamma \vdash v \Leftarrow \{ z : B\text{-bool} \mid TRUE \} \rangle$ **using** *check-iffI by auto*
show $\langle \Theta ; \Phi ; \mathcal{B} ; \Gamma ; \Delta \vdash s1 \Leftarrow \{ z : b-of \ t2 \mid [v]^{ce} == [[L\text{-true}]^v]^{ce} \text{ IMP } c-of \ t2 \ z \} \rangle$
using *check-iffI subtype-iff1 fresh-prodN base-for-lit.simps b-of.simps * check-v-wf by metis*

show $\langle \Theta ; \Phi ; \mathcal{B} ; \Gamma ; \Delta \vdash s2 \Leftarrow \{ z : b-of \ t2 \mid [v]^{ce} == [[L\text{-false}]^v]^{ce} \text{ IMP } c-of \ t2 \ z \} \rangle$
using *check-iffI subtype-iff1 fresh-prodN base-for-lit.simps b-of.simps * check-v-wf by metis*
qed

next

case (*check-assertI* $x \ \Theta \ \Phi \ \mathcal{B} \ \Gamma \ \Delta \ c \ \tau \ s$)
thm *subtype-fresh-tau[where ?t1.0=τ and ?x=x]*
show *?case proof*
have $atom \ x \ \# \ t2$ **using** *subtype-fresh-tau[OF - - ⟨Θ ; B ; Γ ⊢ τ ≲ t2⟩] check-assertI fresh-prodN*
by *simp*
thus $atom \ x \ \# \ (\Theta, \Phi, \mathcal{B}, \Gamma, \Delta, c, t2, s)$ **using** *subtype-fresh-tau check-assertI fresh-prodN by simp*
have $\Theta ; \mathcal{B} ; (x, B\text{-bool}, c) \#_{\Gamma} \Gamma \vdash \tau \lesssim t2$ **apply**(*rule subtype-weakening*)
using *check-assertI apply simp*
using *toSet.simps apply blast*
using *check-assertI check-s-wf by simp*
thus $\Theta ; \Phi ; \mathcal{B} ; (x, B\text{-bool}, c) \#_{\Gamma} \Gamma ; \Delta \vdash s \Leftarrow t2$ **using** *check-assertI by simp*
show $\Theta ; \mathcal{B} ; \Gamma \models c$ **using** *check-assertI by auto*
show $\Theta ; \mathcal{B} ; \Gamma \vdash_{wf} \Delta$ **using** *check-assertI by auto*
qed

next

case (*check-let2I* $x \ P \ \Phi \ \mathcal{B} \ G \ \Delta \ t \ s1 \ \tau \ s2$)
have $wfG \ P \ \mathcal{B} \ ((x, b-of \ t, c-of \ t \ x) \#_{\Gamma} \ G)$
using *check-let2I check-s-wf by metis*
moreover have $toSet \ G \subseteq toSet ((x, b-of \ t, c-of \ t \ x) \#_{\Gamma} \ G)$ **by auto**
ultimately have $*:P ; \mathcal{B} ; (x, b-of \ t, c-of \ t \ x) \#_{\Gamma} \ G \vdash \tau \lesssim t2$ **using** *check-let2I subtype-weakening*
by *metis*
show *?case proof(rule Typing.check-let2I)*
have $atom \ x \ \# \ t2$ **using** *subtype-fresh-tau[of x τ] check-let2I fresh-prodN by metis*
thus $atom \ x \ \# \ (P, \Phi, \mathcal{B}, G, \Delta, t, s1, t2)$ **using** *check-let2I fresh-prodN by metis*
show $P ; \Phi ; \mathcal{B} ; G ; \Delta \vdash s1 \Leftarrow t$ **using** *check-let2I by blast*
show $P ; \Phi ; \mathcal{B} ; (x, b-of \ t, c-of \ t \ x) \#_{\Gamma} \ G ; \Delta \vdash s2 \Leftarrow t2$ **using** *check-let2I * by blast*
qed

```

next
  case (check-varI u  $\Theta$   $\Phi$   $\mathcal{B}$   $\Gamma$   $\Delta$   $\tau'$  v  $\tau$  s)
  show ?case proof(rule Typing.check-varI)
    have atom u  $\#$  t2 using u-fresh-t by auto
    thus  $\langle$ atom u  $\#$  ( $\Theta$ ,  $\Phi$ ,  $\mathcal{B}$ ,  $\Gamma$ ,  $\Delta$ ,  $\tau'$ , v, t2) $\rangle$  using check-varI fresh-prodN by auto
    show  $\langle$  $\Theta$  ;  $\mathcal{B}$  ;  $\Gamma$   $\vdash$  v  $\Leftarrow$   $\tau'$  $\rangle$  using check-varI by auto
    show  $\langle$   $\Theta$  ;  $\Phi$  ;  $\mathcal{B}$  ;  $\Gamma$  ; (u,  $\tau'$ )  $\#_{\Delta}$   $\Delta$   $\vdash$  s  $\Leftarrow$  t2 $\rangle$  using check-varI by auto
  qed
next
  case (check-assignI  $\Delta$  u  $\tau$  P G v z  $\tau'$ )
  then show ?case using Typing.check-assignI by (meson subtype-trans)
next
  case (check-whileI  $\Delta$  G P s1 z s2  $\tau'$ )
  then show ?case using Typing.check-whileI by (meson subtype-trans)
next
  case (check-seqI  $\Delta$  G P s1 z s2  $\tau$ )
  then show ?case using Typing.check-seqI by blast
next
  case (check-caseI  $\Delta$   $\Gamma$   $\Theta$  tid cs  $\tau$  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  $\Theta$  ;  $\Phi$  ;  $\mathcal{B}$  ; GNil ;  $\Delta$   $\vdash$  AS-let x e1 s  $\Leftarrow$   $\tau \implies \Theta$  ;  $\Phi$  ;  $\mathcal{B}$  ; GNil ;  $\Delta$   $\vdash$  e1  $\Rightarrow$   $\tau_1 \implies$ 
     $\Theta$  ;  $\Phi$  ;  $\mathcal{B}$  ; GNil ;  $\Delta$   $\vdash$  e2  $\Rightarrow$   $\tau_2 \implies \Theta$  ;  $\Phi$  ;  $\mathcal{B}$  ; GNil  $\vdash$   $\tau_2 \lesssim \tau_1 \implies \Theta$  ;  $\Phi$  ;  $\mathcal{B}$  ; GNil ;  $\Delta$   $\vdash$  AS-let
  x e2 s  $\Leftarrow$   $\tau$  and
    check-branch-s  $\Theta$   $\Phi$   $\{\|\}$  GNil  $\Delta$  tid dc const v cs  $\tau \implies$  True and
    check-branch-list  $\Theta$   $\Phi$   $\{\|\}$   $\Gamma$   $\Delta$  tid dclist v css  $\tau \implies$  True
proof(nominal-induct GNil  $\Delta$  AS-let x e1 s  $\tau$  and  $\tau$  and  $\tau$  avoiding: e2  $\tau_1$   $\tau_2$ 
  rule: check-s-check-branch-s-check-branch-list.strong-induct)
  case (check-letI x1  $\Theta$   $\Phi$   $\mathcal{B}$   $\Delta$   $\tau_1$  z1 s1 b1 c1)
  obtain z2 and b2 and c2 where t2: $\tau_2 = \llbracket z2 : b2 \mid c2 \rrbracket \wedge$  atom z2  $\#$  (x1,  $\Theta$ ,  $\Phi$ ,  $\mathcal{B}$ , GNil,  $\Delta$ , e2,
 $\tau_1$ , s1)
    using obtain-fresh-z by metis

    obtain z1a and b1a and c1a where t1: $\tau_1 = \llbracket z1a : b1a \mid c1a \rrbracket \wedge$  atom z1a  $\#$  x1 using
infer-e-uniqueness check-letI by metis
    hence t3:  $\llbracket z1a : b1a \mid c1a \rrbracket = \llbracket z1 : b1 \mid c1 \rrbracket$  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  $\Theta$  ;  $\Phi$  ;  $\mathcal{B}$  ; GNil ;  $\Delta$   $\vdash$  AS-let x1 e2 s1  $\Leftarrow$   $\tau_1$  proof
      show  $\langle$ atom x1  $\#$  ( $\Theta$ ,  $\Phi$ ,  $\mathcal{B}$ , GNil,  $\Delta$ , e2,  $\tau_1$ ) $\rangle$  using check-letI t2 fresh-prodN by metis
      show  $\langle$ atom z2  $\#$  (x1,  $\Theta$ ,  $\Phi$ ,  $\mathcal{B}$ , GNil,  $\Delta$ , e2,  $\tau_1$ , s1) $\rangle$  using check-letI t2 using check-letI t2
fresh-prodN by metis
      show  $\langle$  $\Theta$  ;  $\Phi$  ;  $\mathcal{B}$  ; GNil ;  $\Delta$   $\vdash$  e2  $\Rightarrow$   $\llbracket z2 : b2 \mid c2 \rrbracket$  $\rangle$  using check-letI t2 by metis

      have  $\langle$   $\Theta$  ;  $\Phi$  ;  $\mathcal{B}$  ; GNil@( $x1$ , b2, c2[z2::= $x1$ ] $^v$ ) $\rangle_{cv}$   $\#_{\Gamma}$  GNil ;  $\Delta$   $\vdash$  s1  $\Leftarrow$   $\tau_1$  $\rangle$ 
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 $\langle \Theta ; \Phi ; \mathcal{B} ; GNil @ (x1, b2, c1a[z1a::=[x1]^v]_{cv}) \#_{\Gamma} GNil ; \Delta \vdash s1 \Leftarrow \tau1 \rangle$ **using** *check-letI*
beq append-g.simps subst-defs **by** *metis*
show $\langle \Theta ; \mathcal{B} ; GNil \vdash \llbracket z2 : b2 \mid c2 \rrbracket \lesssim \llbracket z1a : b2 \mid c1a \rrbracket \rangle$ **using** *check-letI beq t1 t2* **by** *metis*
have $atom\ x1 \# c2$ **using** *t2 check-letI τ -fresh-c fresh-prodN* **by** *blast*
moreover $have\ atom\ x1 \# c1a$ **using** *t1 check-letI τ -fresh-c fresh-prodN* **by** *blast*
ultimately $show\ \langle atom\ x1 \# (z1a, z2, c1a, c2) \rangle$ **using** *t1 t2 fresh-prodN fresh-x-neq* **by** *metis*
qed

thus $\langle \Theta ; \Phi ; \mathcal{B} ; (x1, b2, c2[z2::=[x1]^v]_v) \#_{\Gamma} GNil ; \Delta \vdash s1 \Leftarrow \tau1 \rangle$ **using** *append-g.simps*
subst-defs **by** *metis*
qed

moreover $have\ AS\text{-}let\ x1\ e2\ s1 = AS\text{-}let\ x\ e2\ s$ **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 Substitution 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 $atom\ bv \ \# \ v$
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 $subst-b-c-of$:
 $(c-of\ \tau\ z)[bv::=b]_{cb} = c-of\ (\tau[bv::=b]_{\tau b})\ z$
proof($nominal-induct\ \tau\ avoiding: z\ rule:\tau.strong-induct$)
case ($T-refined-type\ z'\ b'\ c'$)
moreover **have** $atom\ bv \ \# \ [z]^\nu$ **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 $subst-b-b-of$:
 $(b-of\ \tau)[bv::=b]_{bb} = b-of\ (\tau[bv::=b]_{\tau b})$
by($nominal-induct\ \tau\ rule:\tau.strong-induct, simp\ add: b-of.simps\ subst-tb.simps$)

lemma $subst-b-if$:
 $\{z : b-of\ \tau[bv::=b]_{\tau b} \mid CE-val\ (v[bv::=b]_{vb})\} == CE-val\ (V-lit\ ll)\ IMP\ c-of\ \tau[bv::=b]_{\tau b}\ z\} =$
 $\{z : b-of\ \tau \mid CE-val\ (v)\} == CE-val\ (V-lit\ ll)\ IMP\ c-of\ \tau\ z\} [bv::=b]_{\tau b}$
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 $subst-b-top-eq$:
 $\{z : B-unit \mid TRUE\} [bv::=b]_{\tau b} = \{z : B-unit \mid TRUE\}$ **and** $\{z : B-bool \mid TRUE\} [bv::=b]_{\tau b} =$
 $\{z : B-bool \mid TRUE\}$ **and** $\{z : B-id\ tid \mid TRUE\} = \{z : B-id\ tid \mid TRUE\} [bv::=b]_{\tau b}$
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 $(v'[x::=V-var\ y]_{cv})[bv::=b]_{cb} = (v'[bv::=b]_{cb})[x::=V-var\ y]_{cv}$
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 $\Theta ; \{|\}\vdash \Gamma[bv::=b]_{\Gamma b} \vdash \tau 1[bv::=b]_{\tau b} \lesssim \tau 2[bv::=b]_{\tau b}$

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 $** : \Theta ; \{|bv|\} ; (x, b, c[z::=V-var\ x]_{cv}) \#_{\Gamma} \Gamma \models c'[z'::=V-var\ x]_{cv}$ **using** *validI subst-defs*
by *metis*

thm *Typing.subtype-baseI*

have $\Theta ; \{|\}\vdash \Gamma[bv::=b]_{\Gamma b} \vdash \{z : b[bv::=b]_{bb} \mid c[bv::=b]_{cb}\} \lesssim \{z' : b[bv::=b]_{bb} \mid c'[bv::=b]_{cb}\}$

proof(*rule Typing.subtype-baseI*)

show $\Theta ; \{|\}\vdash \Gamma[bv::=b]_{\Gamma b} \vdash_{wf} \{z : b[bv::=b]_{bb} \mid c[bv::=b]_{cb}\}$

using *subtype-baseI assms wf-b-subst(4) subst-tb.simps subst-defs* **by** *metis*

show $\Theta ; \{|\}\vdash \Gamma[bv::=b]_{\Gamma b} \vdash_{wf} \{z' : b[bv::=b]_{bb} \mid c'[bv::=b]_{cb}\}$

using *subtype-baseI assms wf-b-subst(4) subst-tb.simps* **by** *metis*

show *atom* $x \# (\Theta, \{|\}\vdash bv\ fset, \Gamma[bv::=b]_{\Gamma b}, z, c[bv::=b]_{cb}, z', c'[bv::=b]_{cb})$

apply(*unfold fresh-prodN, auto simp add: * fresh-prodN fresh-empty-fset*)

using *subst-b-fresh-x * fresh-prodN (atom x # c) (atom x # c') subst-defs subtype-baseI* **by** *metis+*

have $\Theta ; \{|\}\vdash (x, b[bv::=b]_{bb}, c[z::=V-var\ x]_v[bv::=b]_{cb}) \#_{\Gamma} \Gamma[bv::=b]_{\Gamma b} \models c'[z'::=V-var\ x]_v[bv::=b]_{cb}$

using $**$ *subst-b-valid subst-gb.simps assms subtype-baseI* **by** *metis*

thus $\Theta ; \{|\}\vdash (x, b[bv::=b]_{bb}, (c[bv::=b]_{cb})[z::=V-var\ x]_v) \#_{\Gamma} \Gamma[bv::=b]_{\Gamma b} \models (c'[bv::=b]_{cb})[z'::=V-var\ x]_v$

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 *b-of-subst-bv*:

(*b-of* τ) $[x::=v]_{bb} = b-of\ (\tau[x::=v]_{\tau b})$

proof –

obtain $z\ b\ c$ **where** $*:\tau = \{z : b \mid c\} \wedge atom\ z \# (x, v)$ **using** *obtain-fresh-z* **by** *metis*

thus *?thesis using subst-tv.simps * by auto*

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 $\Theta ; \{|\}\vdash G[bv::=b]_{\Gamma b} \vdash v[bv::=b]_{vb} \Rightarrow (\tau[bv::=b]_{\tau b})$

using *assms proof(nominal-induct avoiding: b bv 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* $z \# x$ **using** *infer-v-varI* **by** *auto*

show *atom* $z \# (\Theta, \{|\}\vdash \Gamma[bv::=b]_{\Gamma b})$ **by**(*fresh-mth add: infer-v-varI subst-b-fresh-x subst-b-Γ-def fresh-prodN fresh-empty-fset*)

```

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  $\Theta \mathcal{B} \Gamma t1 t2$ )
show ?case unfolding subst-b-simps b-of-subst-bv proof
show atom z  $\# (v1[bv::=b]_{vb}, v2[bv::=b]_{vb})$  by(fresh-mth add: infer-v-pairI subst-b-fresh-x)
show atom z  $\# (\Theta, \{\|\}, \Gamma[bv::=b]_{\Gamma b})$  by(fresh-mth add: infer-v-pairI subst-b-fresh-x subst-b- $\Gamma$ -def
fresh-empty-fset)
show  $\Theta ; \{\|\} ; \Gamma[bv::=b]_{\Gamma b} \vdash v1[bv::=b]_{vb} \Rightarrow t1[bv::=b]_{\tau b}$  using infer-v-pairI by auto
show  $\Theta ; \{\|\} ; \Gamma[bv::=b]_{\Gamma b} \vdash v2[bv::=b]_{vb} \Rightarrow t2[bv::=b]_{\tau b}$  using infer-v-pairI by auto
qed
next
case (infer-v-consI s dclist  $\Theta dc tc \mathcal{B} \Gamma v tv z$ )
show ?case unfolding subst-b-simps b-of-subst-bv proof
show AF-typedef s dclist  $\in$  set  $\Theta$  using infer-v-consI by auto
show (dc, tc)  $\in$  set dclist using infer-v-consI by auto
show  $\Theta ; \{\|\} ; \Gamma[bv::=b]_{\Gamma b} \vdash v[bv::=b]_{vb} \Rightarrow tv[bv::=b]_{\tau b}$  using infer-v-consI by auto
show  $\Theta ; \{\|\} ; \Gamma[bv::=b]_{\Gamma b} \vdash tv[bv::=b]_{\tau b} \lesssim tc$  proof -
have atom bv  $\# tc$  using wfTh-lookup-supp-empty fresh-def infer-v-consI infer-v-wf by fast
moreover have  $\Theta ; \{\|\} ; \Gamma[bv::=b]_{\Gamma b} \vdash tv[bv::=b]_{\tau b} \lesssim tc[bv::=b]_{\tau b}$ 
using subst-b-subtype infer-v-consI by simp
ultimately show ?thesis using forget-subst subst-b- $\tau$ -def by metis
qed
show atom z  $\# v[bv::=b]_{vb}$  using infer-v-consI using subst-b-fresh-x subst-b-v-def by metis
show atom z  $\# (\Theta, \{\|\}, \Gamma[bv::=b]_{\Gamma b})$  by(fresh-mth add: infer-v-consI subst-b-fresh-x subst-b- $\Gamma$ -def
fresh-empty-fset)
qed
next
case (infer-v-conspI s bv2 dclist2  $\Theta dc tc \mathcal{B} \Gamma v tv ba z$ )
thm Typing.infer-v-conspI
have  $\Theta ; \{\|\} ; \Gamma[bv::=b]_{\Gamma b} \vdash V\text{-consp } s \text{ } dc (ba[bv::=b]_{bb}) (v[bv::=b]_{vb}) \Rightarrow \llbracket z : B\text{-app } s (ba[bv::=b]_{bb})$ 
 $\mid \llbracket [z]^v \rrbracket^{ce} == \llbracket V\text{-consp } s \text{ } dc (ba[bv::=b]_{bb}) (v[bv::=b]_{vb}) \rrbracket^{ce} \rrbracket$ 
proof(rule Typing.infer-v-conspI)
show AF-typedef-poly s bv2 dclist2  $\in$  set  $\Theta$  using infer-v-conspI by auto
show (dc, tc)  $\in$  set dclist2 using infer-v-conspI by auto
show  $\Theta ; \{\|\} ; \Gamma[bv::=b]_{\Gamma b} \vdash v[bv::=b]_{vb} \Rightarrow tv[bv::=b]_{\tau b}$ 
using infer-v-conspI subst-tb.simps by metis

show  $\Theta ; \{\|\} ; \Gamma[bv::=b]_{\Gamma b} \vdash tv[bv::=b]_{\tau b} \lesssim tc[bv2::=ba[bv::=b]_{bb}]_{\tau b}$  proof -
have supp tc  $\subseteq \{ \text{atom } bv2 \}$  using infer-v-conspI wfTh-poly-lookup-supp wfX-wfY by metis
moreover have bv2  $\neq bv$  using  $\langle \text{atom } bv2 \# \mathcal{B} \rangle \langle \mathcal{B} = \{|bv|\} \rangle$  fresh-at-base fresh-def
using fresh-finset by fastforce
ultimately have atom bv  $\# tc$  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  $\Theta ; \{\|\} ; \Gamma[bv::=b]_{\Gamma b} \vdash tv[bv::=b]_{\tau b} \lesssim tc[bv2::=ba]_{\tau b}[bv::=b]_{\tau b}$ 
using infer-v-conspI(7) subst-b-subtype infer-v-conspI by metis
ultimately show ?thesis by auto
qed

```

```

show  $atom\ z \# (\Theta, \{\|\}, \Gamma[bv::=b]_{\Gamma b}, v[bv::=b]_{vb}, ba[bv::=b]_{bb})$ 
  apply(unfold fresh-prodN, intro conjI, auto simp add: infer-v-conspI fresh-empty-fset)
  using  $\langle atom\ z \# \Gamma \rangle$  fresh-subst-if subst-b- $\Gamma$ -def x-fresh-b apply metis
  using  $\langle atom\ z \# v \rangle$  fresh-subst-if subst-b-v-def x-fresh-b by metis
show  $atom\ bv2 \# (\Theta, \{\|\}, \Gamma[bv::=b]_{\Gamma b}, v[bv::=b]_{vb}, ba[bv::=b]_{bb})$ 
  apply(unfold fresh-prodN, intro conjI, auto simp add: infer-v-conspI fresh-empty-fset)
  using  $\langle atom\ bv2 \# b \rangle$   $\langle atom\ bv2 \# \Gamma \rangle$  fresh-subst-if subst-b- $\Gamma$ -def apply metis
  using  $\langle atom\ bv2 \# b \rangle$   $\langle atom\ bv2 \# v \rangle$  fresh-subst-if subst-b-v-def apply metis
  using  $\langle atom\ bv2 \# b \rangle$   $\langle atom\ bv2 \# ba \rangle$  fresh-subst-if subst-b-b-def by metis
show  $\Theta ; \{\|\} \vdash_{wf} ba[bv::=b]_{bb}$ 
  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  $\Theta ; B ; G \vdash v \Leftarrow \tau$  and  $\Theta ; \{\|\} \vdash_{wf} b$  and  $B = \{|bv|\}$ 
  shows  $\Theta ; \{\|\} ; G[bv::=b]_{\Gamma b} \vdash v[bv::=b]_{vb} \Leftarrow (\tau[bv::=b]_{\tau b})$ 
proof –
  obtain  $\tau'$  where  $\Theta ; B ; G \vdash v \Rightarrow \tau' \wedge \Theta ; B ; G \vdash \tau' \lesssim \tau$  using check-v-elim[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*: $\tau.strong-induct$)
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 $atom \ z \ \# \ v[bv ::= b]_{vb}$ **using** *x-fresh-b fresh-subst-if subst-b-v-def T-refined-type* **by** *metis*

hence $\{ z : b \mid c[bv ::= b]_{\tau b}[x ::= v[bv ::= b]_{vb}]_{\tau v} = \{ z : b[bv ::= b]_{bb} \mid (c[bv ::= b]_{cb})[x ::= v[bv ::= b]_{vb}]_{cv} \}$
using *subst-tv.simps subst-tb.simps T-refined-type fresh-Pair* **by** *metis*

moreover have $\{ z : b[bv ::= b]_{bb} \mid (c[bv ::= b]_{cb})[x ::= v[bv ::= b]_{vb}]_{cv} \} = \{ z : b \mid c[x ::= v]_{cv}[bv ::= b]_{\tau b} \}$
using *subst-tv.simps subst-tb.simps ceq $\tau.fresh$ 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 $atom \ bv \ \# \ \tau'$
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 # τ')* **by** *auto*
also have $\dots = \tau'[bv' ::= b]_{\tau b}[x' ::= v]_{\tau v}[bv ::= b]_{\tau b}$
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 $\Theta ; \Phi ; B ; G ; D \vdash e \Rightarrow \tau$ **and** $\Theta ; \{|\}\vdash_{wf} b$ **and** $B = \{ |bv| \}$
shows $\Theta ; \Phi ; \{|\}\vdash G[bv ::= b]_{\Gamma b} ; D[bv ::= b]_{\Delta b} \vdash (e[bv ::= b]_{eb}) \Rightarrow (\tau[bv ::= b]_{\tau b})$
using *assms* **proof** (*nominal-induct* *avoiding*: b *rule*: *infer-e.strong-induct*)
case (*infer-e-valI* $\Theta \ \mathcal{B} \ \Gamma \ \Delta \ \Phi \ v \ \tau$)
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* $\Theta \ \mathcal{B} \ \Gamma \ \Delta \ \Phi \ v1 \ z1 \ c1 \ v2 \ z2 \ c2 \ z3$)
thm *wf-b-subst(15)*
show *?case* **unfolding** *subst-b.simps subst-eb.simps* **proof** (*rule Typing.infer-e-plusI*)
show $\Theta ; \{|\}\vdash \Gamma[bv ::= b]_{\Gamma b} \vdash_{wf} \Delta[bv ::= b]_{\Delta b}$ **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  $\Theta ; \{|\}\} ; \Gamma[bv::=b]_{\Gamma_b} \vdash v1[bv::=b]_{vb} \Rightarrow \{ \mid z1 : B\text{-int} \mid c1[bv::=b]_{cb} \}$  using subst-b-infer-v
infer-e-plusI subst-b-simps by force
  show  $\Theta ; \{|\}\} ; \Gamma[bv::=b]_{\Gamma_b} \vdash v2[bv::=b]_{vb} \Rightarrow \{ \mid z2 : B\text{-int} \mid c2[bv::=b]_{cb} \}$  using subst-b-infer-v
infer-e-plusI subst-b-simps by force
  show atom z3  $\#$  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  $\#$   $\Gamma[bv::=b]_{\Gamma_b}$  using subst-g-b-x-fresh infer-e-plusI by auto
qed
next
case (infer-e-leqI  $\Theta \mathcal{B} \Gamma \Delta \Phi v1 z1 c1 v2 z2 c2 z3$ )
show ?case unfolding subst-b-simps proof(rule Typing.infer-e-leqI)
  show  $\Theta ; \{|\}\} ; \Gamma[bv::=b]_{\Gamma_b} \vdash_{wf} \Delta[bv::=b]_{\Delta_b}$  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  $\Theta ; \{|\}\} ; \Gamma[bv::=b]_{\Gamma_b} \vdash v1[bv::=b]_{vb} \Rightarrow \{ \mid z1 : B\text{-int} \mid c1[bv::=b]_{cb} \}$  using subst-b-infer-v
infer-e-leqI subst-b-simps by force
  show  $\Theta ; \{|\}\} ; \Gamma[bv::=b]_{\Gamma_b} \vdash v2[bv::=b]_{vb} \Rightarrow \{ \mid z2 : B\text{-int} \mid c2[bv::=b]_{cb} \}$  using subst-b-infer-v
infer-e-leqI subst-b-simps by force
  show atom z3  $\#$  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  $\#$   $\Gamma[bv::=b]_{\Gamma_b}$  using subst-g-b-x-fresh infer-e-leqI by auto
qed
next
case (infer-e-appI  $\Theta \mathcal{B} \Gamma \Delta \Phi f x b' c \tau' s' v \tau$ )
show ?case proof(subst subst-eb.simps, rule Typing.infer-e-appI)
  show  $\Theta ; \{|\}\} ; \Gamma[bv::=b]_{\Gamma_b} \vdash_{wf} \Delta[bv::=b]_{\Delta_b}$  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  $\tau'$  s'))) = lookup-fun  $\Phi f$  using
infer-e-appI by auto

have atom bv  $\#$  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 supp c  $\subseteq \{atom\ x\}$  using infer-e-appI wfPhi-f-simple-supp-c[OF -  $\langle \Theta \vdash_{wf} \Phi \rangle$ ] by simp
  hence atom bv  $\#$  c using
    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 fresh-prodN by metis
  thus ?thesis
    using forget-subst subst-b-c-def fresh-def[of atom bv c] by metis
qed
show  $\Theta ; \{|\}\} ; \Gamma[bv::=b]_{\Gamma_b} \vdash v[bv::=b]_{vb} \Leftarrow \{ \mid x : b' \mid c \}$ 
  using subst-b-check-v subst-tb.simps subst-vb.simps infer-e-appI
proof -
  have  $\Theta ; \{|\ bv |\} ; \Gamma \vdash v \Leftarrow \{ \mid x : b' \mid c \}$ 
    by (metis  $\langle \mathcal{B} = \{|\ bv |\} \rangle \langle \Theta ; \mathcal{B} ; \Gamma \vdash v \Leftarrow \{ \mid x : b' \mid c \} \rangle$ )

```

```

    then show ?thesis
    by (metis (no-types)  $\langle \Theta ; \{||\} \vdash_{wf} b \rangle \langle b' = b'[bv::=b]_{bb} \rangle$  subst-b-check-v subst-tb.simps ceq)
qed
show atom  $x \# (\Theta, \Phi, \{||\}::bv \text{ fset}, \Gamma[bv::=b]_{\Gamma b}, \Delta[bv::=b]_{\Delta b}, v[bv::=b]_{vb}, \tau[bv::=b]_{\tau b})$ 
  apply (fresh-mth add: fresh-prodN subst-g-b-x-fresh infer-e-appI )
  using subst-b-fresh-x infer-e-appI apply metis+
done
have supp  $\tau' \subseteq \{ \text{atom } x \}$  using wfPhi-f-simple-supp-t infer-e-appI by auto
hence atom  $bv \# \tau'$  using fresh-def fresh-at-base by force
then show  $\tau'[x::=v[bv::=b]_{vb}]_v = \tau[bv::=b]_{\tau b}$  using infer-e-appI
  forget-subst subst-b- $\tau$ -def subst-tv-subst-vb-switch subst-defs by metis
qed
next
case (infer-e-appPI  $\Theta' \mathcal{B} \Gamma' \Delta \Phi' b' f' bv' x' b1 \ c \ \tau' s' v' \tau 1$ )

have beq:  $b1[bv'::=b]_{bb}[bv::=b]_{bb} = b1[bv'::=b'[bv::=b]_{bb}]_{bb}$ 
proof -
  have supp  $b1 \subseteq \{ \text{atom } bv' \}$  using wfPhi-f-poly-supp-b infer-e-appPI
  using supp-at-base by blast
  moreover have  $bv \neq bv'$  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  $c \subseteq \{ \text{atom } bv', \text{atom } x' \}$  using wfPhi-f-poly-supp-c infer-e-appPI
  using supp-at-base by blast
  moreover have  $bv \neq bv'$  using infer-e-appPI fresh-def supp-at-base
  by (simp add: fresh-def supp-at-base)
  moreover have atom  $x' \neq \text{atom } bv$  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  $\Theta'; \{||\}; \Gamma'[bv::=b]_{\Gamma b} \vdash_{wf} \Delta[bv::=b]_{\Delta b}$  using wf-b-subst subst-db.simps infer-e-appPI wfX-wfY
by metis
  show  $\Theta' \vdash_{wf} \Phi'$  using infer-e-appPI by auto
  show Some (AF-fundef  $f' (AF\text{-fun-}typ\text{-}some\ bv' (AF\text{-fun-}typ\ x' b1 \ c \ \tau' s')) = lookup\text{-}fun\ \Phi' f'$ )
using infer-e-appPI by auto
  thus  $\Theta'; \{||\}; \Gamma'[bv::=b]_{\Gamma b} \vdash v'[bv::=b]_{vb} \Leftarrow \{ x': b1[bv'::=b'[bv::=b]_{bb}]_b \mid c[bv'::=b'[bv::=b]_{bb}]_b \}$ 
  }
  using subst-b-check-v subst-tb.simps subst-b.simps infer-e-appPI
proof -
  have  $\Theta'; \{||\}; \Gamma'[bv::=b]_{\Gamma b} \vdash v'[bv::=b]_{vb} \Leftarrow \{ x': b1[bv'::=b]_b[bv::=b]_{bb} \mid (c[bv'::=b]_b)[bv::=b]_{cb} \}$ 
  }
  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  $x' \# \Gamma'[bv::=b]_{\Gamma b}$  using subst-g-b-x-fresh infer-e-appPI by auto
show  $\tau'[bv'::=b'[bv::=b]_{bb}]_b[x'::=v'[bv::=b]_{vb}]_v = \tau 1[bv::=b]_{\tau b}$  proof -

```

```

    have supp  $\tau' \subseteq \{ \text{atom } x', \text{atom } bv' \}$  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  $\text{atom } bv \# \tau'$  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  $\text{atom } bv' \# (\Theta', \Phi', \{|\}, \Gamma[bv::=b]_{\Gamma b}, \Delta[bv::=b]_{\Delta b}, b'[bv::=b]_{bb}, v'[bv::=b]_{vb}, \tau 1[bv::=b]_{\tau b})$ 
    unfolding fresh-prodN apply( auto simp add: infer-e-appPI fresh-empty-fset)
    using fresh-subst-if subst-b- $\Gamma$ -def subst-b- $\Delta$ -def subst-b-b-def subst-b-v-def subst-b- $\tau$ -def infer-e-appPI
by metis+
  show  $\Theta' ; \{|\} \vdash_{wf} b'[bv::=b]_{bb}$  using infer-e-appPI wf-b-subst by simp
  qed
next
case (infer-e-fstI  $\Theta \mathcal{B} \Gamma \Delta \Phi v z' b1 b2 c z$ )
  show ?case unfolding subst-b-simps proof(rule Typing.infer-e-fstI)
    show  $\Theta ; \{|\} ; \Gamma[bv::=b]_{\Gamma b} \vdash_{wf} \Delta[bv::=b]_{\Delta b}$  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  $\Theta ; \{|\} ; \Gamma[bv::=b]_{\Gamma b} \vdash v[bv::=b]_{vb} \Rightarrow \{ z' : B\text{-pair } b1[bv::=b]_{bb} b2[bv::=b]_{bb} \mid c[bv::=b]_{cb} \}$ 
      using subst-b-infer-v subst-tb.simps subst-b-simps infer-e-fstI by force
    show  $\text{atom } z \# AE\text{-fst } (v[bv::=b]_{vb})$  using infer-e-fstI subst-b-fresh-x subst-b-v-def e.fresh by metis
    show  $\text{atom } z \# \Gamma[bv::=b]_{\Gamma b}$  using subst-g-b-x-fresh infer-e-fstI by auto
  qed
next
case (infer-e-sndI  $\Theta \mathcal{B} \Gamma \Delta \Phi v z' b1 b2 c z$ )
  show ?case unfolding subst-b-simps proof(rule Typing.infer-e-sndI)
    show  $\Theta ; \{|\} ; \Gamma[bv::=b]_{\Gamma b} \vdash_{wf} \Delta[bv::=b]_{\Delta b}$  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  $\Theta ; \{|\} ; \Gamma[bv::=b]_{\Gamma b} \vdash v[bv::=b]_{vb} \Rightarrow \{ z' : B\text{-pair } b1[bv::=b]_{bb} b2[bv::=b]_{bb} \mid c[bv::=b]_{cb} \}$ 
      using subst-b-infer-v subst-tb.simps subst-b-simps infer-e-sndI by force
    show  $\text{atom } z \# AE\text{-snd } (v[bv::=b]_{vb})$  using infer-e-sndI subst-b-fresh-x subst-b-v-def e.fresh by metis
    show  $\text{atom } z \# \Gamma[bv::=b]_{\Gamma b}$  using subst-g-b-x-fresh infer-e-sndI by auto
  qed
next
case (infer-e-lenI  $\Theta \mathcal{B} \Gamma \Delta \Phi v z' c z$ )
  show ?case unfolding subst-b-simps proof(rule Typing.infer-e-lenI)
    show  $\Theta ; \{|\} ; \Gamma[bv::=b]_{\Gamma b} \vdash_{wf} \Delta[bv::=b]_{\Delta b}$  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  $\Theta ; \{|\} ; \Gamma[bv::=b]_{\Gamma b} \vdash v[bv::=b]_{vb} \Rightarrow \{ z' : B\text{-bitvec} \mid c[bv::=b]_{cb} \}$ 
      using subst-b-infer-v subst-tb.simps subst-b-simps infer-e-lenI by force
    show  $\text{atom } z \# AE\text{-len } (v[bv::=b]_{vb})$  using infer-e-lenI subst-b-fresh-x subst-b-v-def e.fresh by metis
    show  $\text{atom } z \# \Gamma[bv::=b]_{\Gamma b}$  using subst-g-b-x-fresh infer-e-lenI by auto
  qed
next

```

```

case (infer-e-mvarI  $\Theta \mathcal{B} \Gamma \Delta \Phi u \tau$ )
show ?case proof(subst subst subst-eb.simps, rule Typing.infer-e-mvarI)
  show  $\Theta ; \{||\} \vdash_{wf} \Gamma[bv::=b]_{\Gamma_b}$  using infer-e-mvarI wf-b-subst by auto
  show  $\Theta \vdash_{wf} \Phi$  using infer-e-mvarI by auto
  show  $\Theta ; \{||\} ; \Gamma[bv::=b]_{\Gamma_b} \vdash_{wf} \Delta[bv::=b]_{\Delta_b}$  using infer-e-mvarI using wf-b-subst(10)
subst-db.simps infer-e-sndI wfX-wfY
  by (metis wf-b-subst(15))
  show  $(u, \tau[bv::=b]_{\tau_b}) \in \text{setD } \Delta[bv::=b]_{\Delta_b}$  using infer-e-mvarI subst-db.simps set-insert
    subst-d-b-member by simp
qed
next
case (infer-e-concatI  $\Theta \mathcal{B} \Gamma \Delta \Phi v1 z1 c1 v2 z2 c2 z3$ )
show ?case unfolding subst-b-simps proof(rule Typing.infer-e-concatI)
  show  $\Theta ; \{||\} ; \Gamma[bv::=b]_{\Gamma_b} \vdash_{wf} \Delta[bv::=b]_{\Delta_b}$  using wf-b-subst(10) subst-db.simps infer-e-concatI
wfX-wfY
  by (metis wf-b-subst(15))
  show  $\Theta \vdash_{wf} \Phi$  using infer-e-concatI by auto
  show  $\Theta ; \{||\} ; \Gamma[bv::=b]_{\Gamma_b} \vdash v1[bv::=b]_{v_b} \Rightarrow \{ z1 : B\text{-bitvec} \mid c1[bv::=b]_{c_b} \}$ 
    using subst-b-infer-v subst-tb.simps subst-b-simps infer-e-concatI by force
  show  $\Theta ; \{||\} ; \Gamma[bv::=b]_{\Gamma_b} \vdash v2[bv::=b]_{v_b} \Rightarrow \{ z2 : B\text{-bitvec} \mid c2[bv::=b]_{c_b} \}$ 
    using subst-b-infer-v subst-tb.simps subst-b-simps infer-e-concatI by force
  show  $\text{atom } z3 \# AE\text{-concat } (v1[bv::=b]_{v_b}) (v2[bv::=b]_{v_b})$  using infer-e-concatI subst-b-fresh-x
subst-b-v-def e.fresh by metis
  show  $\text{atom } z3 \# \Gamma[bv::=b]_{\Gamma_b}$  using subst-g-b-x-fresh infer-e-concatI by auto
qed
next
case (infer-e-splitI  $\Theta \mathcal{B} \Gamma \Delta \Phi v1 z1 c1 v2 z2 z3$ )
show ?case unfolding subst-b-simps proof(rule Typing.infer-e-splitI)
  show  $\langle \Theta ; \{||\} ; \Gamma[bv::=b]_{\Gamma_b} \vdash_{wf} \Delta[bv::=b]_{\Delta_b} \rangle$  using wf-b-subst(10) subst-db.simps infer-e-splitI
wfX-wfY
  by (metis wf-b-subst(15))
  show  $\langle \Theta \vdash_{wf} \Phi \rangle$  using infer-e-splitI by auto
  show  $\langle \Theta ; \{||\} ; \Gamma[bv::=b]_{\Gamma_b} \vdash v1[bv::=b]_{v_b} \Rightarrow \{ z1 : B\text{-bitvec} \mid c1[bv::=b]_{c_b} \} \rangle$ 
    using subst-b-infer-v subst-tb.simps subst-b-simps infer-e-splitI by force
  show  $\langle \Theta ; \{||\} ; \Gamma[bv::=b]_{\Gamma_b} \vdash v2[bv::=b]_{v_b} \Leftarrow \{ z2 : B\text{-int} \mid \text{leq } [ [ L\text{-num } 0 ]^v ]^{ce} [ [ z2 ]^v ]^{ce} ]^{ce} == [ [ L\text{-true } ]^v ]^{ce} \text{ AND } [ \text{leq } [ [ z2 ]^v ]^{ce} [ [ v1[bv::=b]_{v_b} ]^{ce} ]^{ce} ]^{ce} == [ [ L\text{-true } ]^v ]^{ce} \} \rangle$ 
    using subst-b-check-v subst-tb.simps subst-b-simps infer-e-splitI
  proof –
    have  $\Theta ; \{||\} ; \Gamma[bv::=b]_{\Gamma_b} \vdash v2[bv::=b]_{v_b} \Leftarrow \{ z2 : B\text{-int} \mid [ \text{leq } [ [ L\text{-num } 0 ]^v ]^{ce} [ [ z2 ]^v ]^{ce} ]^{ce} == [ [ L\text{-true } ]^v ]^{ce} \text{ AND } [ \text{leq } [ [ z2 ]^v ]^{ce} [ [ v1 ]^{ce} ]^{ce} ]^{ce} == [ [ L\text{-true } ]^v ]^{ce} \} [bv::=b]_{\tau_b}$ 
      using infer-e-splitI.hyps(7) infer-e-splitI.prem(1) infer-e-splitI.prem(2) subst-b-check-v by
presburger
    then show ?thesis
      by simp
  qed
  show  $\langle \text{atom } z1 \# AE\text{-split } (v1[bv::=b]_{v_b}) (v2[bv::=b]_{v_b}) \rangle$  using infer-e-splitI subst-b-fresh-x subst-b-v-def
e.fresh by metis
  show  $\langle \text{atom } z1 \# \Gamma[bv::=b]_{\Gamma_b} \rangle$  using subst-g-b-x-fresh infer-e-splitI by auto

  show  $\langle \text{atom } z2 \# AE\text{-split } (v1[bv::=b]_{v_b}) (v2[bv::=b]_{v_b}) \rangle$  using infer-e-splitI subst-b-fresh-x subst-b-v-def
e.fresh by metis

```

show $\langle \text{atom } z2 \# \Gamma[bv ::= b]_{\Gamma b} \rangle$ **using** *subst-g-b-x-fresh infer-e-splitI* **by** *auto*
show $\langle \text{atom } z3 \# AE\text{-split } (v1[bv ::= b]_{vb}) (v2[bv ::= b]_{vb}) \rangle$ **using** *infer-e-splitI subst-b-fresh-x subst-b-v-def*
e.fresh **by** *metis*
show $\langle \text{atom } z3 \# \Gamma[bv ::= b]_{\Gamma b} \rangle$ **using** *subst-g-b-x-fresh infer-e-splitI* **by** *auto*
qed
qed

lemma *subst-b-c-of-forget*:
assumes *atom bv # const*
shows $(c\text{-of } \text{const } x)[bv ::= b]_{cb} = c\text{-of } \text{const } x$
using *assms proof(nominal-induct const avoiding: x rule:τ.strong-induct)*
case $(T\text{-refined-type } x' \text{ } b' \text{ } c')$
hence $c\text{-of } \{ x' : b' \mid c' \} x = c'[x' ::= V\text{-var } x]_{cv}$ **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*
 $\tau\text{-fresh-c fresh-subst-cv-if has-subst-b-class.subst-b-fresh-x ms-fresh-all}(37)$ *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** $\tau::\tau$
assumes $\Theta ; \{||\} \vdash_{wf} b$ **and** $B = \{bv\}$
shows $\Theta ; \Phi ; B ; G ; D \vdash s \Leftarrow \tau \Longrightarrow \Theta ; \Phi ; \{||\} ; G[bv ::= b]_{\Gamma b} ; D[bv ::= b]_{\Delta b} \vdash (s[bv ::= b]_{sb}) \Leftarrow (\tau[bv ::= b]_{\tau b})$ **and**
 $\Theta ; \Phi ; B ; G ; D ; tid ; cons ; const ; v \vdash cs \Leftarrow \tau \Longrightarrow \Theta ; \Phi ; \{||\} ; G[bv ::= b]_{\Gamma b} ; D[bv ::= b]_{\Delta b} ; tid ; cons ; const ; v[bv ::= b]_{vb} \vdash (subst\text{-}branchb \text{ } cs \text{ } bv \text{ } b) \Leftarrow (\tau[bv ::= b]_{\tau b})$ **and**
 $\Theta ; \Phi ; B ; G ; D ; tid ; dclist ; v \vdash css \Leftarrow \tau \Longrightarrow \Theta ; \Phi ; \{||\} ; G[bv ::= b]_{\Gamma b} ; D[bv ::= b]_{\Delta b} ; tid ; dclist ; v[bv ::= b]_{vb} \vdash (subst\text{-}branchlb \text{ } css \text{ } bv \text{ } b) \Leftarrow (\tau[bv ::= b]_{\tau b})$
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 $(check\text{-}valI \text{ } \Theta \text{ } B \text{ } \Gamma \text{ } \Delta \text{ } \Phi \text{ } v \text{ } \tau' \text{ } \tau)$
show *?case*
apply $(subst \text{ } subst\text{-}sb.simps, rule \text{ } Typing.check\text{-}valI)$
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 $(check\text{-}letI \text{ } x \text{ } \Theta \text{ } \Phi \text{ } B \text{ } \Gamma \text{ } \Delta \text{ } e \text{ } \tau \text{ } z \text{ } s \text{ } b' \text{ } c)$
show *?case* **proof** $(subst \text{ } subst\text{-}sb.simps, rule \text{ } Typing.check\text{-}letI)$

show $\text{atom } x \# (\Theta, \Phi, \{||\}, \Gamma[bv ::= b]_{\Gamma b}, \Delta[bv ::= b]_{\Delta b}, e[bv ::= b]_{eb}, \tau[bv ::= b]_{\tau b})$
apply $(unfold \text{ } fresh\text{-}prodN, auto)$

```

    apply(simp add: check-letI fresh-empty-fset)+
    apply(metis * subst-b-fresh-x check-letI fresh-prodN)+ done
show atom z # (x,  $\Theta$ ,  $\Phi$ ,  $\{\|\}$ ,  $\Gamma[bv::=b]_{\Gamma b}$ ,  $\Delta[bv::=b]_{\Delta b}$ ,  $e[bv::=b]_{eb}$ ,  $\tau[bv::=b]_{\tau b}$ ,  $s[bv::=b]_{sb}$ )
    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  $\Theta$  ;  $\Phi$  ;  $\{\|\}$  ;  $\Gamma[bv::=b]_{\Gamma b}$  ;  $\Delta[bv::=b]_{\Delta b} \vdash e[bv::=b]_{eb} \Rightarrow \{\!| z : b'[bv::=b]_{bb} \mid c[bv::=b]_{cb} \!\}$ 
    using check-letI subst-b-infer-e subst-tb.simps by metis
have  $c[z::=[x]^v]_{cv}[bv::=b]_{cb} = (c[bv::=b]_{cb})[z::=V\text{-var } x]_{cv}$ 
    using subst-cv-subst-bb-commute[of bv V-var x c z b] fresh-at-base by simp
thus  $\Theta$  ;  $\Phi$  ;  $\{\|\}$  ;  $((x, b'[bv::=b]_{bb}, (c[bv::=b]_{cb})[z::=V\text{-var } x]_v) \#_{\Gamma} \Gamma[bv::=b]_{\Gamma b}$  ;  $\Delta[bv::=b]_{\Delta b} \vdash$ 
 $s[bv::=b]_{sb} \Leftarrow \tau[bv::=b]_{\tau b}$ 
    using check-letI subst-gb.simps subst-defs by metis
qed
next
case (check-assertI x  $\Theta$   $\Phi$   $\mathcal{B}$   $\Gamma$   $\Delta$  c  $\tau$  s)
show ?case proof(subst subst-sb.simps, rule Typing.check-assertI)
    show atom x # ( $\Theta$ ,  $\Phi$ ,  $\{\|\}$ ,  $\Gamma[bv::=b]_{\Gamma b}$ ,  $\Delta[bv::=b]_{\Delta b}$ ,  $c[bv::=b]_{cb}$ ,  $\tau[bv::=b]_{\tau b}$ ,  $s[bv::=b]_{sb}$ )
        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  $\Theta$  ;  $\Phi$  ;  $\{\|\}$  ;  $((x, B\text{-bool}, c) \#_{\Gamma} \Gamma)[bv::=b]_{\Gamma b}$  ;  $\Delta[bv::=b]_{\Delta b} \vdash s[bv::=b]_{sb} \Leftarrow \tau[bv::=b]_{\tau b}$  using
check-assertI
    by metis
    thus  $\Theta$  ;  $\Phi$  ;  $\{\|\}$  ;  $(x, B\text{-bool}, c[bv::=b]_{cb}) \#_{\Gamma} \Gamma[bv::=b]_{\Gamma b}$  ;  $\Delta[bv::=b]_{\Delta b} \vdash s[bv::=b]_{sb} \Leftarrow \tau[bv::=b]_{\tau b}$ 
using subst-gb.simps by auto
    show  $\Theta$  ;  $\{\|\}$  ;  $\Gamma[bv::=b]_{\Gamma b} \models c[bv::=b]_{cb}$  using subst-b-valid check-assertI by simp
    show  $\Theta$  ;  $\{\|\}$  ;  $\Gamma[bv::=b]_{\Gamma b} \vdash_{wf} \Delta[bv::=b]_{\Delta b}$  using wf-b-subst2(6) check-assertI by simp
qed
next
case (check-branch-list-consI  $\Theta$   $\Phi$   $\mathcal{B}$   $\Gamma$   $\Delta$  tid dclist v cs  $\tau$  css)
then show ?case unfolding subst-branchlb.simps using Typing.check-branch-list-consI by simp
next
case (check-branch-list-finalI  $\Theta$   $\Phi$   $\mathcal{B}$   $\Gamma$   $\Delta$  tid dclist v cs  $\tau$ )
then show ?case unfolding subst-branchlb.simps using Typing.check-branch-list-finalI by simp
next
case (check-branch-s-branchI  $\Theta$   $\mathcal{B}$   $\Gamma$   $\Delta$   $\tau$  const x  $\Phi$  tid cons v s)

show ?case unfolding subst-b-simps proof(rule Typing.check-branch-s-branchI)
    show  $\Theta$  ;  $\{\|\}$  ;  $\Gamma[bv::=b]_{\Gamma b} \vdash_{wf} \Delta[bv::=b]_{\Delta b}$  using check-branch-s-branchI wf-b-subst subst-db.simps
by metis
    show  $\vdash_{wf} \Theta$  using check-branch-s-branchI by auto
    show  $\Theta$  ;  $\{\|\}$  ;  $\Gamma[bv::=b]_{\Gamma b} \vdash_{wf} \tau[bv::=b]_{\tau b}$  using check-branch-s-branchI wf-b-subst by metis

show atom x # ( $\Theta$ ,  $\Phi$ ,  $\{\|\}$ ,  $\Gamma[bv::=b]_{\Gamma b}$ ,  $\Delta[bv::=b]_{\Delta b}$ , tid, cons, const,  $v[bv::=b]_{vb}$ ,  $\tau[bv::=b]_{\tau b}$ )
    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: $\Theta$  ;  $\{\|\}$  ;  $GNil \vdash_{wf} \text{const}$  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 *wf 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 $\Theta ; \Phi ; \{||\} ; (x, b\text{-of const, CE-val } (v[bv::=b]_{vb}) == CE\text{-val}(V\text{-cons tid cons}$
 $(V\text{-var } x)) \text{ AND } c\text{-of const } x) \#_{\Gamma} \Gamma[bv::=b]_{\Gamma b} ; \Delta[bv::=b]_{\Delta b} \vdash s[bv::=b]_{sb} \Leftarrow \tau[bv::=b]_{\tau b}$
using *check-branch-s-branchI subst-gb.simps* **by** *auto*

qed

next

case (*check-ifI z $\Theta \Phi \mathcal{B} \Gamma \Delta v s1 s2 \tau$*)
show ?*case unfolding subst-b-simps proof(rule Typing.check-ifI)*
show $\langle atom\ z\ \# (\Theta, \Phi, \{||\}, \Gamma[bv::=b]_{\Gamma b}, \Delta[bv::=b]_{\Delta b}, v[bv::=b]_{vb}, s1[bv::=b]_{sb}, s2[bv::=b]_{sb}, \tau[bv::=b]_{\tau b}) \rangle$
by(*unfold fresh-prodN, auto, auto simp add: check-ifI fresh-empty-fset subst-b-fresh-x*)
have $\llbracket z : B\text{-bool} \mid TRUE \rrbracket[bv::=b]_{\tau b} = \llbracket z : B\text{-bool} \mid TRUE \rrbracket$ **by** *auto*
thus $\langle \Theta ; \{||\} ; \Gamma[bv::=b]_{\Gamma b} \vdash v[bv::=b]_{vb} \Leftarrow \llbracket z : B\text{-bool} \mid TRUE \rrbracket \rangle$ **using** *check-ifI subst-b-check-v*
by *metis*
show $\langle \Theta ; \Phi ; \{||\} ; \Gamma[bv::=b]_{\Gamma b} ; \Delta[bv::=b]_{\Delta b} \vdash s1[bv::=b]_{sb} \Leftarrow \llbracket z : b\text{-of } \tau[bv::=b]_{\tau b} \mid CE\text{-val}$
 $(v[bv::=b]_{vb}) == CE\text{-val}(V\text{-lit } L\text{-true}) \text{ IMP } c\text{-of } \tau[bv::=b]_{\tau b}\ z \rrbracket \rangle$
using *subst-b-if check-ifI by metis*
show $\langle \Theta ; \Phi ; \{||\} ; \Gamma[bv::=b]_{\Gamma b} ; \Delta[bv::=b]_{\Delta b} \vdash s2[bv::=b]_{sb} \Leftarrow \llbracket z : b\text{-of } \tau[bv::=b]_{\tau b} \mid CE\text{-val}$
 $(v[bv::=b]_{vb}) == CE\text{-val}(V\text{-lit } L\text{-false}) \text{ IMP } c\text{-of } \tau[bv::=b]_{\tau b}\ z \rrbracket \rangle$
using *subst-b-if check-ifI by metis*
qed

next

case (*check-let2I x $\Theta \Phi \mathcal{B} G \Delta t s1 \tau s2$*)
show ?*case unfolding subst-b-simps proof(rule Typing.check-let2I)*
have *atom x # b using x-fresh-b by auto*
show $\langle atom\ x\ \# (\Theta, \Phi, \{||\}, G[bv::=b]_{\Gamma b}, \Delta[bv::=b]_{\Delta b}, t[bv::=b]_{\tau b}, s1[bv::=b]_{sb}, \tau[bv::=b]_{\tau b}) \rangle$
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 $\langle \Theta ; \Phi ; \{||\} ; G[bv::=b]_{\Gamma b} ; \Delta[bv::=b]_{\Delta b} \vdash s1[bv::=b]_{sb} \Leftarrow t[bv::=b]_{\tau b} \rangle$ **using** *check-let2I*
subst-tb.simps by auto
show $\langle \Theta ; \Phi ; \{||\} ; (x, b\text{-of } t[bv::=b]_{\tau b}, c\text{-of } t[bv::=b]_{\tau b}\ x) \#_{\Gamma} G[bv::=b]_{\Gamma b} ; \Delta[bv::=b]_{\Delta b} \vdash$
 $s2[bv::=b]_{sb} \Leftarrow \tau[bv::=b]_{\tau b} \rangle$
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 $\Theta \Phi \mathcal{B} \Gamma \Delta \tau' v \tau s$*)
show ?*case unfolding subst-b-simps proof(rule Typing.check-varI)*
show *atom u # ($\Theta, \Phi, \{||\}, \Gamma[bv::=b]_{\Gamma b}, \Delta[bv::=b]_{\Delta b}, \tau'[bv::=b]_{\tau b}, v[bv::=b]_{vb}, \tau[bv::=b]_{\tau b}$)*
by(*unfold fresh-prodN, auto simp add: check-varI fresh-empty-fset subst-b-fresh-u*)
show $\Theta ; \{||\} ; \Gamma[bv::=b]_{\Gamma b} \vdash v[bv::=b]_{vb} \Leftarrow \tau'[bv::=b]_{\tau b}$ **using** *check-varI subst-b-check-v* **by** *auto*
show $\Theta ; \Phi ; \{||\} ; (subst\text{-gb } \Gamma\ bv\ b) ; (u, (\tau'[bv::=b]_{\tau b})) \#_{\Delta} (subst\text{-db } \Delta\ bv\ b) \vdash (s[bv::=b]_{sb})$


```

<= (τ[bv::=b]τb) using check-varI by auto
qed
next
case (check-assignI Θ Φ B Γ Δ u τ v z τ')
show ?case unfolding subst-b-simps proof( rule Typing.check-assignI)
  show Θ ⊢wf Φ using check-assignI by auto
  show Θ ; {||} ; Γ[bv::=b]Γb ⊢wf Δ[bv::=b]Δb using wf-b-subst check-assignI by auto
  show (u, τ[bv::=b]τb) ∈ setD Δ[bv::=b]Δb using check-assignI subst-d-b-member by simp
  show Θ ; {||} ; Γ[bv::=b]Γb ⊢ v[bv::=b]vb <= τ[bv::=b]τb using check-assignI subst-b-check-v by
auto
    show Θ ; {||} ; Γ[bv::=b]Γb ⊢ { z : B-unit | TRUE } ≲ τ'[bv::=b]τb using check-assignI
subst-b-subtype subst-b-simps subst-tb.simps by fastforce
  qed
next
case (check-whileI Θ Φ B Γ Δ s1 z s2 τ')
show ?case unfolding subst-b-simps proof(rule Typing.check-whileI)
  show Θ ; Φ ; {||} ; Γ[bv::=b]Γb ; Δ[bv::=b]Δb ⊢ s1[bv::=b]sb <= { z : B-bool | TRUE } using
check-whileI by auto
  show Θ ; Φ ; {||} ; Γ[bv::=b]Γb ; Δ[bv::=b]Δb ⊢ s2[bv::=b]sb <= { z : B-unit | TRUE } using
check-whileI by auto
  show Θ ; {||} ; Γ[bv::=b]Γb ⊢ { z : B-unit | TRUE } ≲ τ'[bv::=b]τb using subst-b-subtype
check-whileI by fastforce
  qed
next
case (check-seqI Θ Φ 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 B Γ Δ tid dclist v cs τ z)
show ?case unfolding subst-b-simps proof(rule Typing.check-caseI)
  show (Θ ; Φ ; {||} ; Γ[bv::=b]Γb ; Δ[bv::=b]Δb ; tid ; dclist ; v[bv::=b]vb ⊢ subst-branchlb cs bv b
<= τ[bv::=b]τb) using check-caseI by auto
  show (AF-typedef tid dclist ∈ set Θ) using check-caseI by auto
  show (Θ ; {||} ; Γ[bv::=b]Γb ⊢ v[bv::=b]vb <= { 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]τb using subst-b-eq by auto
    then show ?thesis
  by (metis (no-types) check-caseI.hyps(4) check-caseI.prem(1) check-caseI.prem(2) subst-b-check-v)
  qed
  show (⊢wf Θ) using check-caseI by auto
qed
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

abbreviation *delta-ext* (- \sqsubseteq -) **where**
 delta-ext $\Delta \Delta' \equiv (\text{setD } \Delta \subseteq \text{setD } \Delta')$

nominal-function *dc-of* :: *branch-s* \Rightarrow *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 $\Theta \vdash \delta \sim \Delta$ **and** *atom* *u* $\# \delta$
 shows *atom* *u* $\# \Delta$
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* $\Theta \delta \Delta v \tau u'$)
 hence $\Theta ; \{||\} ; GNil \vdash_{wf} \tau$ **using** *check-v-wf* **by** *meson*
 hence *supp* $\tau = \{\}$ **using** *wfT-supp* **by** *fastforce*
 moreover have *atom* *u* $\# u'$ **using** *delta-sim-consI* *fresh-Cons* *fresh-Pair* **by** *blast*
 moreover have *atom* *u* $\# \Delta$ **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 $\Delta :: \Delta$
 assumes $\Theta \vdash \delta \sim \Delta$ **and** $(u, v) \in \text{set } \delta$ **and** $(u, \tau) \in \text{setD } \Delta$ **and** $\Theta ; \{||\} ; GNil \vdash_{wf} \Delta$
 shows $\Theta ; \{||\} ; GNil \vdash v \Leftarrow \tau$
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 $*:\Theta \vdash ((u',v')\#\delta) \sim \Delta$ **using** *uv Cons* **by** *blast*
 then obtain τ' and Δ' **where** *tt*: $\Theta ; \{\|\} ; GNil \vdash v' \Leftarrow \tau' \wedge u \notin \text{fst} \text{ ' set } \delta \wedge \Delta = (u,\tau')\#\Delta'$
using *delta-sim-elim*(\mathcal{J})[*OF* *] **by** *metis*
 moreover **hence** $v'=v$ **using** *Cons True*
 by (*metis Pair-inject fst-conv image-eqI set-ConsD uv*)
 moreover **have** $\tau=\tau'$ **using** *wfD-unique tt Cons*
 setD.simps list.set-intros **by** *blast*
 ultimately show *?thesis* **by** *metis*
next
 case *False*
 hence $*:\Theta \vdash ((u',v')\#\delta) \sim \Delta$ **using** *uv Cons* **by** *blast*
 then obtain τ' and Δ' **where** *tt*: $\Theta \vdash \delta \sim \Delta' \wedge \Theta ; \{\|\} ; GNil \vdash v' \Leftarrow \tau' \wedge u' \notin \text{fst} \text{ ' set } \delta \wedge \Delta = (u',\tau')\#\Delta'$ **using** *delta-sim-elim*(\mathcal{J})[*OF* *] **by** *metis*

 moreover **hence** $\Theta ; \{\|\} ; GNil \vdash_{wf} \Delta'$ **using** *wfD-elim Cons delta-sim-elim* **by** *metis*
 ultimately show *?thesis* **using** *Cons*
 using *False* **by** *auto*
qed
qed

lemma *delta-sim-delta-lookup*:
 assumes $\Theta \vdash \delta \sim \Delta$ **and** $(u, \{\!| z : b \mid c |\!\}) \in \text{setD } \Delta$
 shows $\exists v. (u,v) \in \text{set } \delta$
using *assms* **by**(*induct rule: delta-sim.inducts,auto+*)

lemma *update-d-stable*:
 $\text{fst} \text{ ' set } \delta = \text{fst} \text{ ' set } (\text{update-d } \delta \ u \ v)$
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 $\Delta::\Delta$
 assumes $\Theta \vdash \delta \sim \Delta$ **and** $\Theta ; \{\|\} ; GNil \vdash v \Leftarrow \tau$ **and** $(u,\tau) \in \text{setD } \Delta$ **and** $\Theta ; \{\|\} ; GNil \vdash_{wf} \Delta$
 shows $\Theta \vdash (\text{update-d } \delta \ u \ v) \sim \Delta$
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 $*:\Theta \vdash ((u',v')\#\delta) \sim \Delta$ **using** *uv Cons* **by** *blast*
 then obtain τ' and Δ' **where** *tt*: $\Theta \vdash \delta \sim \Delta' \wedge \Theta ; \{\|\} ; GNil \vdash v' \Leftarrow \tau' \wedge u' \notin \text{fst} \text{ ' set } \delta \wedge \Delta =$

$(u', \tau') \#_{\Delta'} \text{ using } \text{delta-sim-elim} * \text{ by } \text{metis}$

```

show ?case proof(cases u=u')
  case True
  then have  $(u, \tau') \in \text{setD } \Delta$  using tt by auto
  then have  $\tau = \tau'$  using Cons wfD-unique by metis
  moreover have  $\text{update-d } ((u', v') \# \delta) u v = ((u', v) \# \delta)$  using  $\text{update-d.simps True}$  by presburger
  ultimately show ?thesis using  $\text{delta-sim-consI tt Cons True}$ 
    by (simp add: tt uv)
next
case False
have  $\Theta \vdash (u', v') \# (\text{update-d } \delta u v) \sim (u', \tau') \#_{\Delta'}$ 
proof(rule  $\text{delta-sim-consI}$ )
  show  $\Theta \vdash \text{update-d } \delta u v \sim \Delta'$  using Cons using  $\text{delta-sim-consI}$ 
     $\text{delta-sim.simps update-d.simps Cons delta-sim-elim uv tt}$ 
     $\text{False fst-conv set-ConsD wfG-elim wfD-elim}$  by (metis  $\text{setD-ConsD}$ )
  show  $\Theta ; \{||\} ; \text{GNil} \vdash v' \Leftarrow \tau'$  using tt by auto
  show  $u' \notin \text{fst 'set (update-d } \delta u v)$  using  $\text{update-d.simps Cons update-d-stable tt}$  by auto
qed
thus ?thesis using  $\text{False update-d.simps uv}$ 
  by (simp add: tt)
qed
qed

```

16.2 Preservation

Types are preserved under reduction step

16.2.1 Function Application

lemma *check-s-x-fresh*:

fixes $x::x$ and $s::s$
 assumes $\Theta ; \Phi ; B ; \text{GNil} ; D \vdash s \Leftarrow \tau$
 shows $\text{atom } x \# s \wedge \text{atom } x \# \tau \wedge \text{atom } x \# D$

proof —

have $\Theta ; \Phi ; B ; \text{GNil} ; D \vdash_{wf} s : b\text{-of } \tau$ using $\text{check-s-wf[OF assms]}$ by auto
 moreover have $\Theta ; B ; \text{GNil} \vdash_{wf} \tau$ using check-s-wf assms by auto
 moreover have $\Theta ; B ; \text{GNil} \vdash_{wf} D$ using check-s-wf assms by auto
 ultimately show ?thesis using wf-supp x-fresh-u
 by (meson $\text{fresh-GNil wfS-x-fresh wfT-x-fresh wfD-x-fresh}$)

qed

lemma *check-funtyp-subst-b*:

fixes $b'::b$
 assumes $\text{check-funtyp } \Theta \Phi \{||bv||\} (AF\text{-fun-typ } x b c \tau s)$ and $\langle \Theta ; \{||\} \vdash_{wf} b' \rangle$
 shows $\text{check-funtyp } \Theta \Phi \{||\} (AF\text{-fun-typ } x b[bv::=b]_{bb} (c[bv::=b]_{cb}) \tau[bv::=b]_{\tau b} s[bv::=b]_{sb})$
 using assms **proof** ($\text{nominal-induct } \{||bv||\} AF\text{-fun-typ } x b c \tau s$ rule: $\text{check-funtyp.strong-induct}$)
 case ($\text{check-funtypI } x' \Theta \Phi c' s' \tau'$)
 have $\text{check-funtyp } \Theta \Phi \{||\} (AF\text{-fun-typ } x' b[bv::=b]_{bb} (c'[bv::=b]_{cb}) \tau'[bv::=b]_{\tau b} s'[bv::=b]_{sb})$ **proof**
 show $\langle \text{atom } x' \# (\Theta, \Phi, \{||\}::bv \text{ fset}, b[bv::=b]_{bb}) \rangle$ using $\text{check-funtypI fresh-prodN x-fresh-b fresh-empty-fset}$
 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**
 $\Theta ; \{||\} ; GNil \vdash v \Leftarrow \llbracket x : b \mid c \rrbracket$
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 \tau s avoiding: v x rule: check-funtyp.strong-induct)*
case $(check\text{-}funtypI\ x'\ \Theta\ \Phi\ c'\ s'\ \tau')$

hence *eq1*: $\llbracket x' : b \mid c' \rrbracket = \llbracket x : b \mid c \rrbracket$ **using** *funtyp-eq-iff-equalities* **by** *metis*

obtain x'' **and** c'' **where** $xf:\llbracket x : b \mid c \rrbracket = \llbracket x'' : b \mid c'' \rrbracket \wedge atom\ x'' \# (x', v) \wedge atom\ x'' \# (x, c)$
using *obtain-fresh-z3* **by** *metis*
moreover have $atom\ x' \# c''$ **proof** –
have $supp\ \llbracket x'' : b \mid c'' \rrbracket = \{\}$ **using** *eq1 check-funtypI xf check-v-wf wfT-nil-sup* **by** *metis*
hence $supp\ c'' \subseteq \{atom\ x''\}$ **using** $\tau.supp\ eq1\ xf$ **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 $atom\ x' \# c$ **proof** –

have $supp\ \llbracket x : b \mid c \rrbracket = \{\}$ **using** *eq1 check-funtypI xf check-v-wf wfT-nil-sup* **by** *metis*
hence $supp\ c \subseteq \{atom\ x\}$ **using** $\tau.supp$ **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  $\langle \Theta ; \{||\} ; GNil \vdash v \Leftarrow \llbracket x'' : b \mid c'' \rrbracket \rangle$  using check-funtypI eq1 xf by metis
  show  $\langle atom\ x'' \# (x', v) \rangle$  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
   $\Theta ; \{||\} ; GNil \vdash v \Leftarrow \llbracket x : b \mid c \rrbracket$ 
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-induct)
case (check-fundefq-simpleI  $\Theta \Phi x' c' t' s'$ )
hence eq:  $\llbracket x : b \mid c \rrbracket = \llbracket x' : b \mid c' \rrbracket \wedge s'[x'::=v]_{sv} = s[x::=v]_{sv} \wedge t[x::=v]_{\tau v} = t'[x'::=v]_{\tau v}$ 
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-fun-typ  $x' b'' c' t' s' = [[atom bv]]lst.$  AF-fun-typ x b c t s
shows  $\llbracket x' : b''[bv'::=b]_{bb} \mid c'[bv'::=b]_{cb} \rrbracket = \llbracket x : b[bv::=b]_{bb} \mid c[bv::=b]_{cb} \rrbracket \wedge$ 
   $s'[bv'::=b]_{sb}[x'::=v]_{sv} = s[bv::=b]_{sb}[x::=v]_{sv} \wedge t'[bv'::=b]_{\tau b}[x'::=v]_{\tau v} = t[bv::=b]_{\tau b}[x::=v]_{\tau v}$ 

```

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
   $\Theta ; \{||\} ; GNil \vdash v \Leftarrow \llbracket x : b[bv::=b]_{bb} \mid c[bv::=b]_{cb} \rrbracket$  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  $bv' \Theta \Phi x' b'' c' t' s'$ )

```

```

hence **:  $\llbracket x' : b''[bv'::=b]_{bb} \mid c'[bv'::=b]_{cb} \rrbracket = \llbracket x : b[bv::=b]_{bb} \mid c[bv::=b]_{cb} \rrbracket \wedge$ 

```

$$s'[bv'::=b]_{sb}[x'::=v]_{sv} = s[bv::=b]_{sb}[x::=v]_{sv} \wedge t'[bv'::=b]_{\tau b}[x'::=v]_{\tau v} = t[bv::=b]_{\tau b}[x::=v]_{\tau v}$$

using *funtyp-poly-eq-iff-equalities* **by** *metis*

have $*:check-funtyp \ \Theta \ \Phi \ \{\|\} \ (AF-fun-tyt \ x' \ b''[bv'::=b]_{bb} \ (c'[bv'::=b]_{cb}) \ (t'[bv'::=b]_{\tau b}) \ s'[bv'::=b]_{sb})$
using *check-funtyp-subst-b* $[OF \ check-funtypq-polyI(5) \ check-funtypq-polyI(8)]$ **by** *metis*
moreover have $\Theta ; \{\|\} ; GNil \vdash v \Leftarrow \{ \{ x' : b''[bv'::=b]_{bb} \mid c'[bv'::=b]_{cb} \} \}$ **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* $\Theta \ \Phi \ (AF-fundef \ f \ (AF-fun-tyt-none \ (AF-fun-tyt \ x \ b \ c \ t \ s)))$ **and**

$\Theta ; \{\|\} ; GNil \vdash v \Leftarrow \{ \{ x : b \mid c \} \}$ **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* $(AF-fundef \ f \ (AF-fun-tyt-none \ (AF-fun-tyt \ x \ b \ c \ t \ s)))$ *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* $\Theta \ \Phi \ (AF-fundef \ f \ (AF-fun-tyt-some \ bv \ (AF-fun-tyt \ x \ b \ c \ t \ s)))$ **and**

$\Theta ; \{\|\} ; GNil \vdash v \Leftarrow \{ \{ x : b[bv::=b]_{bb} \mid c[bv::=b]_{cb} \} \}$ **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* $(AF-fundef \ f \ (AF-fun-tyt-some \ bv \ (AF-fun-tyt \ x \ b \ c \ t \ s)))$ *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 $(AF-fundef \ f \ (AF-fun-tyt-none \ (AF-fun-tyt \ x1 \ b1 \ c1 \ \tau1' \ s1')))$ $= \text{lookup-fun} \ \Phi \ f$ **and**
 $(\forall fd \in \text{set} \ \Phi. \ check-fundef \ \Theta \ \Phi \ fd)$

shows $\Theta ; \Phi ; B ; G ; \Delta \vdash ss \Leftarrow \tau \Longrightarrow B = \{\|\} \Longrightarrow G = GNil \Longrightarrow ss = LET \ x = (AE-app \ f \ v) \ IN \ s \Longrightarrow$

$\Theta ; \Phi ; \{\|\} ; GNil ; \Delta \vdash LET \ x : (\tau1'[x1::=v]_{\tau v}) = (s1'[x1::=v]_{sv}) \ IN \ s \Leftarrow \tau$ **and**

check-branch-s $\Theta \ \Phi \ \mathcal{B} \ GNil \ \Delta \ tid \ dc \ const \ v \ cs \ \tau \Longrightarrow \text{True}$ **and**

check-branch-list $\Theta \ \Phi \ \mathcal{B} \ \Gamma \ \Delta \ tid \ dclist \ v \ css \ \tau \Longrightarrow \text{True}$

using *assms* **proof**(*nominal-induct* τ **and** τ **and** τ *avoiding*: *v rule: check-s-check-branch-s-check-branch-list.strong-induct*)

case (*check-letI* $x2 \ \Theta \ \Phi \ \mathcal{B} \ \Gamma \ \Delta \ e \ \tau \ z \ s2 \ b \ c$)

hence *eq*: $e = (AE-app \ f \ v)$ **by** *simp*

hence $\ast:\Theta ; \Phi ; \{\|\} ; GNil ; \Delta \vdash (AE\text{-}app\ f\ v) \Rightarrow \{\| z : b \mid c \}$ **using** *check-letI* **by** *auto*

then obtain $x3\ b3\ c3\ \tau3\ s3$ **where**

$\ast:\Theta ; \{\|\} ; GNil \vdash_{wf} \Delta \wedge \Theta \vdash_{wf} \Phi \wedge \text{Some } (AF\text{-}fundef\ f\ (AF\text{-}fun\text{-}typ\text{-}none\ (AF\text{-}fun\text{-}typ\ x3\ b3\ c3\ \tau3\ s3))) = \text{lookup-fun } \Phi\ f \wedge$

$\Theta ; \{\|\} ; GNil \vdash v \Leftarrow \{\| x3 : b3 \mid c3 \} \wedge \text{atom } x3 \# (\Theta, \Phi, (\{\|\}::bv\ fset), GNil, \Delta, v, \{\| z : b \mid c \}) \wedge \tau3[x3::=v]_{\tau v} = \{\| z : b \mid c \}$

using *infer-e-elimis(6)[OF *] subst-defs* **by** *metis*

obtain $z3$ **where** $z3:\{\| x3 : b3 \mid c3 \} = \{\| z3 : b3 \mid c3[x3::=V\text{-}var\ z3]_{cv} \} \wedge \text{atom } z3 \# (x3, v, c3, x1, c1)$ **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 $?ft = AF\text{-}fun\text{-}typ\ x3\ b3\ c3\ \tau3\ s3$

have $supp\ \tau3 \subseteq \{atom\ x3\} \wedge supp\ s3 \subseteq \{atom\ x3\}$ **using** *wfPhi-f-supp *** **by** *metis*

have $\Theta ; \Phi ; \{\|\} ; GNil ; \Delta \vdash AS\text{-}let2\ x2\ \tau3[x3::=v]_{\tau v}\ (s3[x3::=v]_{sv})\ s2 \Leftarrow \tau$ **proof**

show $\langle atom\ x2 \# (\Theta, \Phi, \{\|\}::bv\ fset, GNil, \Delta, \tau3[x3::=v]_{\tau v}, s3[x3::=v]_{sv}, \tau) \rangle$

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 $\langle \Theta ; \Phi ; \{\|\} ; GNil ; \Delta \vdash s3[x3::=v]_{sv} \Leftarrow \tau3[x3::=v]_{\tau v} \rangle$ **proof**(*rule fundef-simple-check*)

show $\langle check\text{-}fundef\ \Theta\ \Phi\ (AF\text{-}fundef\ f\ (AF\text{-}fun\text{-}typ\text{-}none\ (AF\text{-}fun\text{-}typ\ x3\ b3\ c3\ \tau3\ s3))) \rangle$ **using**

$\ast\ check\text{-}letI\ lookup\text{-}fun\text{-}member$ **by** *metis*

show $\langle \Theta ; \{\|\} ; GNil \vdash v \Leftarrow \{\| x3 : b3 \mid c3 \} \rangle$ **using** $\ast\ast$ **by** *auto*

show $\langle \Theta ; \{\|\} ; GNil \vdash_{wf} \Delta \rangle$ **using** $\ast\ast$ **by** *auto*

qed

show $\langle \Theta ; \Phi ; \{\|\} ; (x2, b\text{-}of\ \tau3[x3::=v]_{\tau v}, c\text{-}of\ \tau3[x3::=v]_{\tau v}\ x2) \#_{\Gamma} GNil ; \Delta \vdash s2 \Leftarrow \tau \rangle$

using *check-letI ** b-of.simps c-of.simps subst-defs* **by** *metis*

qed

moreover have $AS\text{-}let2\ x2\ \tau3[x3::=v]_{\tau v}\ (s3[x3::=v]_{sv})\ s2 = AS\text{-}let2\ x\ (\tau1'[x1::=v]_{\tau v})\ (s1'[x1::=v]_{sv})$

s proof –

have $\ast: [[atom\ x2]]lst. s2 = [[atom\ x]]lst. s$ **using** *check-letI s-branch-s-branch-list.eq-iff* **by** *auto*

moreover have $\tau3[x3::=v]_{\tau v} = \tau1'[x1::=v]_{\tau v}$ **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** $tm::'a::\{has\text{-}subst\text{-}v, has\text{-}subst\text{-}b\}$ **and** $x::x$

assumes $supp\ tm \subseteq \{atom\ bv, atom\ x\}$ **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 $\tau 1'$ s1')))) = lookup-fun Φ f
and *($\forall fd \in set \Phi. check-fundef \Theta \Phi fd$)*
shows $\Theta ; \Phi ; B ; G ; \Delta \vdash ss \Leftarrow \tau \implies B = \{\|\} \implies G = GNil \implies ss = LET\ x = (AE-appP\ f\ b'\ v)\ IN\ s \implies \Theta ; \{\|\} \vdash_{wf} b' \implies$
 $\Theta ; \Phi ; \{\|\} ; GNil ; \Delta \vdash LET\ x : (\tau 1' [bv1::=b]_{\tau b} [x1::=v]_{\tau v}) = (s1' [bv1::=b]_{sb} [x1::=v]_{sv})$
IN s $\Leftarrow \tau$ and
check-branch-s $\Theta \Phi \mathcal{B} GNil \Delta tid\ dc\ const\ v\ cs\ \tau \implies True$ and
check-branch-list $\Theta \Phi \mathcal{B} \Gamma \Delta tid\ dclist\ v\ css\ \tau \implies True$
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 $\Theta \Phi \mathcal{B} \Gamma \Delta e\ \tau\ z\ s2\ b\ c$)*

hence *eq: e = (AE-appP f b' v)* **by** *simp*

hence $*:\Theta ; \Phi ; \{\|\} ; GNil ; \Delta \vdash (AE-appP\ f\ b'\ v) \Rightarrow \{\|\ z : b \mid c \}$ **using** *check-letI* **by** *auto*

then obtain $x3\ b3\ c3\ \tau3\ s3\ bv3$ **where**

$*:\Theta ; \{\|\} ; GNil \vdash_{wf} \Delta \wedge \Theta \vdash_{wf} \Phi \wedge Some\ (AF-fundef\ f\ (AF-fun-typ-some\ bv3\ (AF-fun-typ\ x3\ b3\ c3\ \tau3\ s3))) = lookup-fun\ \Phi\ f \wedge$
 $\Theta ; \{\|\} ; GNil \vdash v \Leftarrow \{\|\ x3 : b3[bv3::=b]_{bb} \mid c3[bv3::=b]_{cb} \} \wedge atom\ x3 \# GNil \wedge$
 $\tau3[bv3::=b]_{\tau b} [x3::=v]_{\tau v} = \{\|\ z : b \mid c \}$
 $\wedge \Theta ; \{\|\} \vdash_{wf} b'$
using *infer-e-elim(21)[OF *] subst-defs* **by** *metis*

obtain $z3$ **where** $z3:\{\|\ x3 : b3 \mid c3 \} = \{\|\ z3 : b3 \mid c3[x3::=V-var\ z3]_{cv} \} \wedge atom\ z3 \# (x3,$
 $v, c3, x1, c1)$ **using** *obtain-fresh-z3* **by** *metis*

let $?ft = (AF-fun-typ\ x3\ (b3[bv3::=b]_{bb})\ (c3[bv3::=b]_{cb})\ (\tau3[bv3::=b]_{\tau b})\ (s3[bv3::=b]_{sb}))$

have $*:check-fundef\ \Theta\ \Phi\ (AF-fundef\ f\ (AF-fun-typ-some\ bv3\ (AF-fun-typ\ x3\ b3\ c3\ \tau3\ s3)))$ **using**
 $**\ check-letI\ lookup-fun-member$ **by** *metis*

hence $ftq:check-funtypq\ \Theta\ \Phi\ (AF-fun-typ-some\ bv3\ (AF-fun-typ\ x3\ b3\ c3\ \tau3\ s3))$ **using** *check-fundef-elim*
by *auto*

let $?ft = AF-fun-typ-some\ bv3\ (AF-fun-typ\ x3\ b3\ c3\ \tau3\ s3)$

have $supp\ \tau3 \subseteq \{atom\ x3, atom\ bv3\} \wedge supp\ s3 \subseteq \{atom\ x3, atom\ bv3\}$

using *wfPhi-f-poly-supp-t wfPhi-f-poly-supp-s* **** by** *metis*

have $\Theta ; \Phi ; \{\|\} ; GNil ; \Delta \vdash AS-let2\ x2\ \tau3[bv3::=b]_{\tau b} [x3::=v]_{\tau v}\ (s3[bv3::=b]_{sb} [x3::=v]_{sv})\ s2 \Leftarrow \tau$
proof

show $\langle atom\ x2 \# (\Theta, \Phi, \{\|\}::bv\ fset, GNil, \Delta, \tau3[bv3::=b]_{\tau b} [x3::=v]_{\tau v}, s3[bv3::=b]_{sb} [x3::=v]_{sv},$
 $\tau) \rangle$

proof –

```

thm fresh-subst-v-subst-b
have atom x2 # τ3[bv3::=b]τb[x3::=v]τv
  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  $\forall b. \text{atom } x2 = \text{atom } x3 \vee \text{atom } x2 \# s3[bv3::=b]_b$ 
  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  $\langle \Theta; \Phi; \{\|\}; GNil; \Delta \vdash s3[bv3::=b]sb[x3::=v]sv \Leftarrow \tau3[bv3::=b]τb[x3::=v]τv \rangle$  proof (rule
fundef-poly-check)
  show  $\langle \text{check-fundef } \Theta \Phi (AF\text{-fundef } f (AF\text{-fun-typ-some } bv3 (AF\text{-fun-typ } x3 \ b3 \ c3 \ \tau3 \ s3))) \rangle$ 
  using ** lookup-fun-member check-letI by metis
  show  $\langle \Theta; \{\|\}; GNil \vdash v \Leftarrow \{ x3 : b3[bv3::=b]bb \mid c3[bv3::=b]cb \} \rangle$  using ** by metis
  show  $\langle \Theta; \{\|\}; GNil \vdash_{wf} \Delta \rangle$  using ** by metis
  show  $\langle \Theta; \{\|\} \vdash_{wf} b' \rangle$  using ** by metis
  qed
  show  $\langle \Theta; \Phi; \{\|\}; (x2, \text{b-of } \tau3[bv3::=b]τb[x3::=v]τv, \text{c-of } \tau3[bv3::=b]τb[x3::=v]τv \ x2) \#_{\Gamma} GNil$ 
 $;\Delta \vdash s2 \Leftarrow \tau \rangle$ 
  using check-letI ** b-of.simps c-of.simps subst-defs by metis
qed

```

```

moreover have AS-let2 x2 τ3[bv3::=b]τb[x3::=v]τv (s3[bv3::=b]sb[x3::=v]sv) s2 = AS-let2 x
 $(\tau1'[bv1::=b]τb[x1::=v]τv) (s1'[bv1::=b]sb[x1::=v]sv) s$  proof –
  have *: [[atom x2]]lst. s2 = [[atom x]]lst. s using check-letI s-branch-s-branch-list.eq-iff by auto
  moreover have  $\tau3[bv3::=b]τb[x3::=v]τv = \tau1'[bv1::=b]τb[x1::=v]τv$  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  $\Theta; \Phi; \{\|\}; GNil; \Delta \vdash LET \ x = (AE\text{-op } Plus (V\text{-lit } (L\text{-num } n1)) (V\text{-lit } (L\text{-num } n2)))$  IN
 $s' \Leftarrow \tau$ 
shows  $\Theta; \Phi; \{\|\}; GNil; \Delta \vdash LET \ x = (AE\text{-val } (V\text{-lit } (L\text{-num } (n1+n2))))$  IN  $s' \Leftarrow \tau$ 
proof –
  obtain t1 where  $1: \Theta; \Phi; \{\|\}; GNil; \Delta \vdash AE\text{-op } Plus (V\text{-lit } (L\text{-num } n1)) (V\text{-lit } (L\text{-num } n2)) \Rightarrow$ 
 $t1$ 
  using assms check-s-elim by metis

```

then obtain $z1$ **where** $2: t1 = \{ z1 : B\text{-int} \mid CE\text{-val} (V\text{-var } z1) == CE\text{-op Plus} ([V\text{-lit} (L\text{-num } n1)]^{ce}) ([V\text{-lit} (L\text{-num } n2)]^{ce}) \}$
using *infer-e-plus* **by** *metis*

obtain $z2$ **where** $3: \langle \Theta ; \Phi ; \{||\} ; GNil ; \Delta \vdash AE\text{-val} (V\text{-lit} (L\text{-num} (n1+n2))) \Rightarrow \{ z2 : B\text{-int} \mid CE\text{-val} (V\text{-var } z2) == CE\text{-val} (V\text{-lit} (L\text{-num} (n1+n2))) \} \rangle$
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 $\Theta ; \Phi ; \{||\} ; GNil ; \Delta \vdash LET\ x = (AE\text{-op LEq} (V\text{-lit} (L\text{-num } n1)) (V\text{-lit} (L\text{-num } n2)))$
 $IN\ s' \Leftarrow \tau$
shows $\Theta ; \Phi ; \{||\} ; GNil ; \Delta \vdash LET\ x = (AE\text{-val} (V\text{-lit} (if\ (n1 \leq n2)\ then\ L\text{-true}\ else\ L\text{-false})))$ $IN\ s' \Leftarrow \tau$

proof –

obtain $t1$ **where** $1: \langle \Theta ; \Phi ; \{||\} ; GNil ; \Delta \vdash AE\text{-op LEq} (V\text{-lit} (L\text{-num } n1)) (V\text{-lit} (L\text{-num } n2)) \Rightarrow t1 \rangle$
using *assms check-s-elim* **by** *metis*
then obtain $z1$ **where** $2: t1 = \{ z1 : B\text{-bool} \mid CE\text{-val} (V\text{-var } z1) == CE\text{-op LEq} ([V\text{-lit} (L\text{-num } n1)]^{ce}) ([V\text{-lit} (L\text{-num } n2)]^{ce}) \}$
using *infer-e-leq* **by** *auto*

obtain $z2$ **where** $3: \langle \Theta ; \Phi ; \{||\} ; GNil ; \Delta \vdash AE\text{-val} (V\text{-lit} ((if\ (n1 \leq n2)\ then\ L\text{-true}\ else\ L\text{-false}))) \Rightarrow \{ z2 : B\text{-bool} \mid CE\text{-val} (V\text{-var } z2) == CE\text{-val} (V\text{-lit} ((if\ (n1 \leq n2)\ then\ L\text{-true}\ else\ L\text{-false}))) \} \rangle$

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 $\langle \Theta ; \Phi ; \{||\} ; GNil ; \Delta \vdash AS\text{-let } x\ (AE\text{-op LEq} [L\text{-num } n1]^v [L\text{-num } n2]^v)\ s' \Leftarrow \tau \rangle$ **using** *assms* **by** *auto*

show $\langle \Theta ; \Phi ; \{||\} ; GNil ; \Delta \vdash AE\text{-op LEq} [L\text{-num } n1]^v [L\text{-num } n2]^v \Rightarrow t1 \rangle$ **using** *1* **by** *auto*

show $\langle \Theta ; \Phi ; \{||\} ; GNil ; \Delta \vdash [[if\ n1 \leq n2\ then\ L\text{-true}\ else\ L\text{-false}]^v]^e \Rightarrow \{ z2 : B\text{-bool} \mid CE\text{-val} (V\text{-var } z2) == CE\text{-val} (V\text{-lit} ((if\ (n1 \leq n2)\ then\ L\text{-true}\ else\ L\text{-false}))) \} \rangle$ **using** *3* **by** *auto*

show $\langle \Theta ; \{||\} ; GNil \vdash \{ z2 : B\text{-bool} \mid CE\text{-val} (V\text{-var } z2) == CE\text{-val} (V\text{-lit} ((if\ (n1 \leq n2)\ then\ L\text{-true}\ else\ L\text{-false}))) \} \lesssim t1 \rangle$

using *subtype-bop[where opp=LEq] check-s-wf assms 2*

by (*metis opp.distinct(1) subtype-bop type-l-eq*)

qed

qed

16.2.2 Operators

lemma *preservation-plus*:

assumes $\Theta ; \Phi ; \Delta \vdash \langle \delta , LET\ x = (AE\text{-op Plus} (V\text{-lit} (L\text{-num } n1)) (V\text{-lit} (L\text{-num } n2))) \rangle\ IN\ s' \Leftarrow \tau$

shows $\Theta ; \Phi ; \Delta \vdash \langle \delta , LET\ x = (AE\text{-val} (V\text{-lit} (L\text{-num} (n1+n2)))) \rangle\ IN\ s' \Leftarrow \tau$

proof –

have $tt: \Theta; \Phi; \{\|\}; GNil; \Delta \vdash AS\text{-}let\ x\ (AE\text{-}op\ Plus\ (V\text{-}lit\ (L\text{-}num\ n1))\ (V\text{-}lit\ (L\text{-}num\ n2)))\ s' \Leftarrow \tau$
and $dsim: \Theta \vdash \delta \sim \Delta$ **and** $fd: (\forall fd \in set\ \Phi. check\text{-}fundef\ \Theta\ \Phi\ fd)$
using *assms config-type-elim* **by** *blast+*

hence $\Theta; \Phi; \{\|\}; GNil; \Delta \vdash AS\text{-}let\ x\ (AE\text{-}val\ (V\text{-}lit\ (L\text{-}num\ (n1+n2))))\ s' \Leftarrow \tau$ **using** *check-s-plus assms* **by** *auto*

hence $\Theta; \Phi; \Delta \vdash \langle \delta, AS\text{-}let\ x\ (AE\text{-}val\ (V\text{-}lit\ (L\text{-}num\ (n1+n2))))\ s' \rangle \Leftarrow \tau$ **using** *dsim config-typeI*
fd **by** *presburger*

then show *?thesis* **using** *dsim config-typeI*
by (*meson order-refl*)
qed

lemma *preservation-leq*:

assumes $\Theta; \Phi; \Delta \vdash \langle \delta, AS\text{-}let\ x\ (AE\text{-}op\ LEq\ (V\text{-}lit\ (L\text{-}num\ n1))\ (V\text{-}lit\ (L\text{-}num\ n2)))\ s' \rangle \Leftarrow \tau$

shows $\Theta; \Phi; \Delta \vdash \langle \delta, AS\text{-}let\ x\ (AE\text{-}val\ (V\text{-}lit\ (((if\ (n1 \leq n2)\ then\ L\text{-}true\ else\ L\text{-}false))))\ s' \rangle \Leftarrow \tau$
proof –

have $tt: \Theta; \Phi; \{\|\}; GNil; \Delta \vdash AS\text{-}let\ x\ (AE\text{-}op\ LEq\ (V\text{-}lit\ (L\text{-}num\ n1))\ (V\text{-}lit\ (L\text{-}num\ n2)))\ s' \Leftarrow \tau$
and $dsim: \Theta \vdash \delta \sim \Delta$ **and** $fd: (\forall fd \in set\ \Phi. check\text{-}fundef\ \Theta\ \Phi\ fd)$
using *assms config-type-elim* **by** *blast+*

hence $\Theta; \Phi; \{\|\}; GNil; \Delta \vdash AS\text{-}let\ x\ (AE\text{-}val\ (V\text{-}lit\ (((if\ (n1 \leq n2)\ then\ L\text{-}true\ else\ L\text{-}false))))\ s' \rangle \Leftarrow \tau$
using *check-s-leq assms* **by** *auto*

hence $\Theta; \Phi; \Delta \vdash \langle \delta, AS\text{-}let\ x\ (AE\text{-}val\ (V\text{-}lit\ (((if\ (n1 \leq n2)\ then\ L\text{-}true\ else\ L\text{-}false))))\ s' \rangle \Leftarrow \tau$
using *dsim config-typeI fd* **by** *presburger*

then show *?thesis* **using** *dsim config-typeI*
by (*meson order-refl*)
qed

16.2.3 Let Statements

lemma *subst-s-abs-lst*:

fixes $s::s$ **and** $sa::s$ **and** $v'::v$

assumes $[[atom\ x]]lst. s = [[atom\ xa]]lst. sa$ **and** $atom\ xa \# v \wedge atom\ x \# v$

shows $s[x::=v]_{sv} = sa[xa::=v]_{sv}$

proof –

obtain $c'::x$ **where** $cdash: atom\ c' \# (v, x, xa, s, sa)$ **using** *obtain-fresh* **by** *blast*

moreover have $(x \leftrightarrow c') \cdot s = (xa \leftrightarrow c') \cdot sa$ **proof** –

have $atom\ c' \# (s, sa) \wedge atom\ c' \# (x, xa, s, sa)$ **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 $\Theta; \Phi; B; G; \Delta \vdash ss \Leftarrow \tau \implies B = \{\|\} \implies G = GNil \implies$

$ss = AS\text{-}let\ x\ (AE\text{-}val\ v)\ s \vee ss = AS\text{-}let2\ x\ t\ (AS\text{-}val\ v)\ s \implies \Theta; \Phi; \{\|\}; GNil; \Delta \vdash$
 $(s[x ::= v]_{sv}) \Leftarrow \tau$ **and**
 $check\text{-}branch\text{-}s\ \Theta\ \Phi\ \mathcal{B}\ GNil\ \Delta\ tid\ dc\ const\ v\ cs\ \tau \implies True$ **and**
 $check\text{-}branch\text{-}list\ \Theta\ \Phi\ \mathcal{B}\ \Gamma\ \Delta\ tid\ dclist\ v\ css\ \tau \implies True$
proof(*nominal-induct* τ **and** τ **and** τ *avoiding: v rule: check-s-check-branch-s-check-branch-list.strong-induct*)
case ($check\text{-}letI\ x1\ \Theta\ \Phi\ \mathcal{B}\ \Gamma\ \Delta\ e\ \tau\ z\ s1\ b\ c$)
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 $**:(\Theta; \mathcal{B}; \Gamma \vdash v \Rightarrow \llbracket z : b \mid c \rrbracket)$ **using** *infer-e-elim*s $check\text{-}letI\ *$ **by** *fast*
thus $\langle \Theta; \mathcal{B}; \Gamma \vdash \llbracket z : b \mid c \rrbracket \lesssim \llbracket z : b \mid c \rrbracket \rangle$ **using** *subtype-refl* *infer-v-wf* **by** *metis*
show $\langle atom\ z \# (x1, v) \rangle$ **using** $check\text{-}letI\ fresh\text{-}Pair$ **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** $check\text{-}letI\ subst\text{-}defs$ **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** $check\text{-}letI$ **by** *auto*
moreover **have** $s1[x1 ::= v]_{sv} = s[x ::= v]_{sv}$
by (*metis* (*full-types*) $check\text{-}letI\ fresh\text{-}GNil\ infer\text{-}e\text{-}elim$ s(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** $check\text{-}letI$ **by** *simp*
next
case ($check\text{-}let2I\ x1\ \Theta\ \Phi\ \mathcal{B}\ \Gamma\ \Delta\ t\ s1\ \tau\ s2$)

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** $*:atom\ z \# (x1, v, t)$ **using** $obtain\text{-}fresh\text{-}z$ **by** *metis*
hence $teq:t = \llbracket z : b\text{-}of\ t \mid c\text{-}of\ t\ z \rrbracket$ **using** $b\text{-}of\text{-}c\text{-}of\text{-}eq$ **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** $check\text{-}s\text{-}elim$ s(1) $check\text{-}let2I$ (10)
s1eq **by** *auto*
thus $**:(\Theta; \mathcal{B}; \Gamma \vdash v \Leftarrow \llbracket z : b\text{-}of\ t \mid c\text{-}of\ t\ z \rrbracket)$ **using** $check\text{-}v.intros\ teq$ **by** *auto*
show $atom\ z \# (x1, v)$ **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** $check\text{-}let2I$ **by** *auto*
show $(x1, b\text{-}of\ t, c\text{-}of\ t\ x1) \#_{\Gamma} \Gamma = GNil @ (x1, b\text{-}of\ t, (c\text{-}of\ t\ z)[z ::= V\text{-}var\ x1]_{cv}) \#_{\Gamma} \Gamma$ **using**
append-g.simps *c-of-switch* $*\ fresh\text{-}prodN$ **by** *metis*
qed

hence $\Theta; \Phi; \mathcal{B}; \Gamma; \Delta \vdash s2[x1 ::= v]_{sv} \Leftarrow \tau$ **using** $check\text{-}let2I$ **by** *auto*
moreover **have** $s2[x1 ::= v]_{sv} = s[x ::= v]_{sv}$ **using**
 $check\text{-}let2I\ fresh\text{-}GNil\ check\text{-}s\text{-}elim$ s *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 $AS\text{-}let2\ x\ t\ (AS\text{-}val\ v)\ s = AS\text{-}let2\ x1\ t\ s1\ s2$
by (*metis* $check\text{-}let2I.prem$ s(3) *s-branch-s-branch-list.distinct* *s-branch-s-branch-list.eq-iff*(3))
then show $?thesis$
by (*metis* (*no-types*) $check\text{-}let2I\ check\text{-}let2I.prem$ s(2) $check\text{-}s\text{-}elim$ s(1) *fresh-GNil* *s-branch-s-branch-list.eq-iff*(3))

subst-s-abs-let wfG-x-fresh-in-v-simple)

qed

ultimately show *?case* **using** *check-let2I* **by** *simp*

qed(*auto*+))

lemma *preservation-let-val*:

assumes $\Theta; \Phi; \Delta \vdash \langle \delta, AS\text{-}let\ x\ (AE\text{-}val\ v)\ s \rangle \Leftarrow \tau \vee \Theta; \Phi; \Delta \vdash \langle \delta, AS\text{-}let2\ x\ t\ (AS\text{-}val\ v)\ s \rangle \Leftarrow \tau$ (**is** *?A* \vee *?B*)

shows $\exists \Delta'. \Theta; \Phi; \Delta' \vdash \langle \delta, s[x::=v]_{sv} \rangle \Leftarrow \tau \wedge \Delta \sqsubseteq \Delta'$

proof –

have *tt*: $\Theta \vdash \delta \sim \Delta$ **and** *fd*: $(\forall fd \in set\ \Phi. check\text{-}fundef\ \Theta\ \Phi\ fd)$

using *assms* **by** *blast*+

have *?A* \vee *?B* **using** *assms* **by** *auto*

then have $\Theta; \Phi; \{\|\}; GNil; \Delta \vdash s[x::=v]_{sv} \Leftarrow \tau$

proof

assume $\Theta; \Phi; \Delta \vdash \langle \delta, AS\text{-}let\ x\ (AE\text{-}val\ v)\ s \rangle \Leftarrow \tau$

hence $*$: $\Theta; \Phi; \{\|\}; GNil; \Delta \vdash AS\text{-}let\ x\ (AE\text{-}val\ v)\ s \Leftarrow \tau$ **by** *blast*

thus *?thesis* **using** *check-let-val* **by** *simp*

next

assume $\Theta; \Phi; \Delta \vdash \langle \delta, AS\text{-}let2\ x\ t\ (AS\text{-}val\ v)\ s \rangle \Leftarrow \tau$

hence $*$: $\Theta; \Phi; \{\|\}; GNil; \Delta \vdash AS\text{-}let2\ x\ t\ (AS\text{-}val\ v)\ s \Leftarrow \tau$ **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 $fst\text{-}snd = AE\text{-}fst \wedge v=v1 \vee fst\text{-}snd = AE\text{-}snd \wedge v=v2$

and $\Theta; \Phi; \{\|\}; GNil; \Delta \vdash AS\text{-}let\ x\ (fst\text{-}snd\ (V\text{-}pair\ v1\ v2))\ s' \Leftarrow \tau$

shows $\Theta; \Phi; \{\|\}; GNil; \Delta \vdash AS\text{-}let\ x\ (AE\text{-}val\ v)\ s' \Leftarrow \tau$

proof –

have *1*: $\langle \Theta; \Phi; \{\|\}; GNil; \Delta \vdash AS\text{-}let\ x\ (fst\text{-}snd\ (V\text{-}pair\ v1\ v2))\ s' \Leftarrow \tau \rangle$ **using** *assms* **by** *auto*

then obtain *t1* **where** $2:\Theta; \Phi; \{\|\}; GNil; \Delta \vdash (fst\text{-}snd\ (V\text{-}pair\ v1\ v2)) \Rightarrow t1$ **using** *check-s-elim* **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 $\Theta; \Phi; \Delta \vdash \langle \delta, LET\ x = (fst\text{-}snd\ (V\text{-}pair\ v1\ v2))\ IN\ s' \rangle \Leftarrow \tau$ **and**
 $fst\text{-}snd = AE\text{-}fst \wedge v=v1 \vee fst\text{-}snd = AE\text{-}snd \wedge v=v2$

shows $\exists \Delta'. \Theta; \Phi; \Delta \vdash \langle \delta, LET\ x = (AE\text{-}val\ v)\ IN\ s' \rangle \Leftarrow \tau \wedge \Delta \sqsubseteq \Delta'$

proof –

have $tt: \Theta; \Phi; \{\|\}; GNil; \Delta \vdash AS\text{-}let\ x\ (fst\text{-}snd\ (V\text{-}pair\ v1\ v2))\ s' \Leftarrow \tau \wedge \Theta \vdash \delta \sim \Delta$ **using** *assms*
by *blast*

hence $t2: \Theta; \Phi; \{\|\}; GNil; \Delta \vdash AS\text{-}let\ x\ (fst\text{-}snd\ (V\text{-}pair\ v1\ v2))\ s' \Leftarrow \tau$ **by** *auto*

moreover have $\forall fd \in set\ \Phi. check\text{-}fundef\ \Theta\ \Phi\ fd$ **using** *assms config-type-elim* **by** *auto*
ultimately show *?thesis* **using** *config-typeI order-refl tt assms check-s-fst-snd* **by** *metis*
qed

inductive-cases *check-branch-s-elim2[elim!]*:

$\Theta; \Phi; \mathcal{B}; \Gamma; \Delta; tid; cons; const; v \vdash cs \Leftarrow \tau$

lemmas *freshers = freshers atom-dom.simps toSet.simps fresh-def x-not-in-b-set*
declare *freshers [simp]*

lemma *subtype-eq-if*:

fixes $t::\tau$ **and** $va::v$

assumes $\Theta; \mathcal{B}; \Gamma \vdash_{wf} \{z : b\text{-of}\ t \mid c\text{-of}\ t\ z\}$ **and** $\Theta; \mathcal{B}; \Gamma \vdash_{wf} \{z : b\text{-of}\ t \mid c\ IMP\ c\text{-of}\ t\ z\}$
shows $\Theta; \mathcal{B}; \Gamma \vdash \{z : b\text{-of}\ t \mid c\text{-of}\ t\ z\} \lesssim \{z : b\text{-of}\ t \mid c\ IMP\ c\text{-of}\ t\ z\}$

proof –

obtain $x::x$ **where** $xf:atom\ x \# ((\Theta, \mathcal{B}, \Gamma, z, c\text{-of}\ t\ z, z, c\ IMP\ c\text{-of}\ t\ z), c)$ **using** *obtain-fresh* **by** *metis*

moreover have $\Theta; \mathcal{B}; (x, b\text{-of}\ t, (c\text{-of}\ t\ z)[z::=[x]^v]_{cv}) \#_{\Gamma} \Gamma \models (c\ IMP\ c\text{-of}\ t\ z)[z::=[x]^v]_{cv}$

unfolding *subst-cv.simps*

proof(*rule valid-eq-imp*)

have $\Theta; \mathcal{B}; (x, b\text{-of}\ t, (c\text{-of}\ t\ z)[z::=[x]^v]_v) \#_{\Gamma} \Gamma \vdash_{wf} (c\ IMP\ (c\text{-of}\ t\ z))[z::=[x]^v]_v$

apply(*rule wfT-wfC-cons*)

apply(*simp add: assms, simp add: assms, unfold fresh-prodN*)

using *xf fresh-prodN* **by** *metis+*

thus $\Theta; \mathcal{B}; (x, b\text{-of}\ t, (c\text{-of}\ t\ z)[z::=[x]^v]_{cv}) \#_{\Gamma} \Gamma \vdash_{wf} c[z::=[x]^v]_{cv} IMP\ (c\text{-of}\ t\ z)[z::=[x]^v]_{cv}$

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** $atom\ z \# t$

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 $\langle \Theta ; \mathcal{B} ; \Gamma \vdash_{wf} c1 \text{ AND } c2 \rangle$ **using** *valid.simps wfC-conjI assms* **by** *auto*

show $\langle \forall i. \Theta ; \Gamma \vdash i \wedge i \models \Gamma \longrightarrow i \models c1 \text{ AND } c2 \rangle$

proof(*rule+*)

fix *i*

assume $\ast : \Theta ; \Gamma \vdash i \wedge i \models \Gamma$

thus $i \models c1 \sim \text{True}$ **using** *assms valid.simps*

using *is-satis.cases* **by** *blast*

show $i \models c2 \sim \text{True}$ **using** *assms valid.simps*

using *is-satis.cases* \ast **by** *blast*

qed

qed

16.2.4 Other Statements

lemma *check-if*:

fixes $s'::s$ **and** $cs::\text{branch-}s$ **and** $css::\text{branch-list}$ **and** $v::v$

shows $\Theta ; \Phi ; B ; G ; \Delta \vdash s' \Leftarrow \tau \Longrightarrow s' = \text{IF } (V\text{-lit } ll) \text{ THEN } s1 \text{ ELSE } s2 \Longrightarrow$

$\Theta ; \{\|\} ; GNil \vdash_{wf} \tau \Longrightarrow G = GNil \Longrightarrow B = \{\|\} \Longrightarrow ll = L\text{-true} \wedge s = s1 \vee ll = L\text{-false} \wedge s = s2 \Longrightarrow$

$\Theta ; \Phi ; \{\|\} ; GNil ; \Delta \vdash s \Leftarrow \tau$ **and**

check-branch-} s $\Theta \Phi \{\|\} GNil \Delta \text{ tid dc const } v \text{ cs } \tau \Longrightarrow \text{True}$ **and**

check-branch-list $\Theta \Phi \{\|\} \Gamma \Delta \text{ tid dclist } v \text{ css } \tau \Longrightarrow \text{True}$

proof(*nominal-induct* τ **and** τ **and** τ *rule: check-s-check-branch-s-check-branch-list.strong-induct*)

case (*check-ifI* $z \Theta \Phi \mathcal{B} \Gamma \Delta v s1 s2 \tau$)

obtain z' **where** *teq*: $\tau = \llbracket z' : b\text{-of } \tau \mid c\text{-of } \tau z' \rrbracket \wedge \text{atom } z' \nmid (z, \tau)$ **using** *obtain-fresh-z-c-of* **by** *metis*

hence *ceq*: $(c\text{-of } \tau z') [z'::[z]^v]_{cv} = (c\text{-of } \tau z)$ **using** *c-of-switch fresh-Pair* **by** *metis*

have *zf*: $\text{atom } z \nmid c\text{-of } \tau z'$

using *c-of-fresh check-ifI teq fresh-Pair fresh-at-base* **by** *metis*

hence $1 : \Theta ; \Phi ; \{\|\} ; GNil ; \Delta \vdash s \Leftarrow \llbracket z : b\text{-of } \tau \mid CE\text{-val } (V\text{-lit } ll) \rrbracket == CE\text{-val } (V\text{-lit } ll) \text{ IMP } c\text{-of } \tau z \rrbracket$ **using** *check-ifI* **by** *auto*

moreover **have** $2 : \Theta ; \{\|\} ; GNil \vdash (\llbracket z : b\text{-of } \tau \mid CE\text{-val } (V\text{-lit } ll) \rrbracket == CE\text{-val } (V\text{-lit } ll) \text{ IMP } c\text{-of } \tau z \rrbracket) \lesssim \tau$

proof –

have $\Theta ; \{\|\} ; GNil \vdash_{wf} (\llbracket z : b\text{-of } \tau \mid CE\text{-val } (V\text{-lit } ll) \rrbracket == CE\text{-val } (V\text{-lit } ll) \text{ IMP } c\text{-of } \tau z \rrbracket)$ **using** *check-ifI check-s-wf* **by** *auto*

moreover **have** $\Theta ; \{\|\} ; GNil \vdash_{wf} \tau$ **using** *check-s-wf check-ifI* **by** *auto*

ultimately show *?thesis* **using** *subtype-if-simp*[*of* $\Theta \{\|\} z b\text{-of } \tau ll c\text{-of } \tau z' z'$] **using** *teq ceq zf subst-defs* **by** *metis*

qed

ultimately show *?case* **using** *check-s-supertype(1) check-ifI* **by** *metis*

qed(*auto+*)

lemma *preservation-if*:

assumes $\Theta ; \Phi ; \Delta \vdash \langle \delta , \text{IF } (V\text{-lit } ll) \text{ THEN } s1 \text{ ELSE } s2 \rangle \Leftarrow \tau$ **and**

$ll = L\text{-true} \wedge s = s1 \vee ll = L\text{-false} \wedge s = s2$

shows $\Theta ; \Phi ; \Delta \vdash \langle \delta , s \rangle \Leftarrow \tau \wedge \text{setD } \Delta \subseteq \text{setD } \Delta$

proof –

have $\ast : \Theta ; \Phi ; \{\|\} ; GNil ; \Delta \vdash AS\text{-if } (V\text{-lit } ll) s1 s2 \Leftarrow \tau \wedge (\forall fd \in \text{set } \Phi. \text{check-fundef } \Theta \Phi fd)$

using *assms config-type-elim*s **by** *metis*

hence $\Theta ; \Phi ; \{\|\} ; GNil ; \Delta \vdash s \Leftarrow \tau$ **using** *check-s-wf check-if assms* **by** *metis*

hence $\Theta; \Phi; \Delta \vdash \langle \delta, s \rangle \Leftarrow \tau \wedge \text{setD } \Delta \subseteq \text{setD } \Delta$ **using** *config-typeI* *
using *assms(1)* **by** *blast*
thus *?thesis* **by** *blast*
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 $\Theta; \mathcal{B}; GNil \vdash_{wf} \{ z : b \mid c1 \text{ AND } c2 \}$

proof

show $\langle \text{atom } z \# (\Theta, \mathcal{B}, GNil) \rangle$
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 $\langle \Theta; \mathcal{B} \vdash_{wf} b \rangle$ **using** *wfT-elim assms* **by** *metis*
have $\Theta; \mathcal{B}; (z, b, \text{TRUE}) \#_{\Gamma} GNil \vdash_{wf} c1$ **using** *wfT-wfC fresh-GNil assms* **by** *auto*
moreover **have** $\Theta; \mathcal{B}; (z, b, \text{TRUE}) \#_{\Gamma} GNil \vdash_{wf} c2$ **using** *wfT-wfC fresh-GNil assms* **by** *auto*
ultimately **show** $\langle \Theta; \mathcal{B}; (z, b, \text{TRUE}) \#_{\Gamma} GNil \vdash_{wf} c1 \text{ AND } c2 \rangle$ **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 $\Theta; \mathcal{B}; GNil \vdash \{ z : b \mid c\text{-of } t \ z \} \lesssim \{ z : b \mid c1 \text{ AND } c2 \}$

proof –

have *beq: b-of t = b* **using** *subtype-eq-base2 b-of.simps assms* **by** *metis*
obtain $x::x$ **where** $x:\langle \text{atom } x \# (\Theta, \mathcal{B}, GNil, z, c\text{-of } t \ z, z, c1 \text{ AND } c2) \rangle$ **using** *obtain-fresh* **by** *metis*

thus *?thesis* **proof**

have $\text{atom } z \# t$ **using** *subtype-wf wfT-supp fresh-def x-not-in-b-set atom-dom.simps toSet.simps*
assms dom.simps **by** *fastforce*

hence $t:t = \{ z : b\text{-of } t \mid c\text{-of } t \ z \}$ **using** *b-of-c-of-eq* **by** *auto*

thus $\langle \Theta; \mathcal{B}; GNil \vdash_{wf} \{ z : b \mid c\text{-of } t \ z \} \rangle$ **using** *subtype-wf beq assms* **by** *auto*

show $\langle \Theta; \mathcal{B}; (x, b, (c\text{-of } t \ z)[z::=[x]^v]_v) \#_{\Gamma} GNil \models (c1 \text{ AND } c2)[z::=[x]^v]_v \rangle$

proof –

have $\langle \Theta; \mathcal{B}; (x, b, (c\text{-of } t \ z)[z::=[x]^v]_v) \#_{\Gamma} GNil \models c1[z::=[x]^v]_v \rangle$

proof(*rule subtype-valid*)

show $\langle \Theta; \mathcal{B}; GNil \vdash t \lesssim \{ z : b \mid c1 \} \rangle$ **using** *assms* **by** *auto*

show $\langle \text{atom } x \# GNil \rangle$ **using** *fresh-GNil* **by** *auto*

show $\langle t = \{ z : b \mid c\text{-of } t \ z \} \rangle$ **using** *t beq* **by** *auto*

show $\langle \{ z : b \mid c1 \} = \{ z : b \mid c1 \} \rangle$ **by** *auto*

qed

moreover **have** $\langle \Theta; \mathcal{B}; (x, b, (c\text{-of } t \ z)[z::=[x]^v]_v) \#_{\Gamma} GNil \models c2[z::=[x]^v]_v \rangle$

proof(*rule subtype-valid*)

show $\langle \Theta; \mathcal{B}; GNil \vdash t \lesssim \{ z : b \mid c2 \} \rangle$ **using** *assms* **by** *auto*

show $\langle \text{atom } x \# GNil \rangle$ **using** *fresh-GNil* **by** *auto*

show $\langle t = \{ z : b \mid c\text{-of } t \ z \} \rangle$ **using** *t beq* **by** *auto*

show $\langle \{ z : b \mid c2 \} = \{ z : b \mid c2 \} \rangle$ **by** *auto*

qed

ultimately **show** *?thesis* **unfolding** *subst-cv.simps subst-v-c-def* **using** *valid-conj* **by** *metis*

qed

thus $\langle \Theta; \mathcal{B}; GNil \vdash_{wf} \{ z : b \mid c1 \text{ AND } c2 \} \rangle$ **using** *subtype-wf wfT-conj assms* **by** *auto*

qed
qed

lemma *infer-v-conj*:

assumes $\Theta ; \mathcal{B} ; GNil \vdash v \Leftarrow \llbracket z : b \mid c1 \rrbracket$ and $\Theta ; \mathcal{B} ; GNil \vdash v \Leftarrow \llbracket z : b \mid c2 \rrbracket$
shows $\Theta ; \mathcal{B} ; GNil \vdash v \Leftarrow \llbracket z : b \mid c1 \text{ AND } c2 \rrbracket$

proof –

obtain $t1$ where $t1: \Theta ; \mathcal{B} ; GNil \vdash v \Rightarrow t1 \wedge \Theta ; \mathcal{B} ; GNil \vdash t1 \lesssim \llbracket z : b \mid c1 \rrbracket$

using *assms check-v-elim* by *metis*

obtain $t2$ where $t2: \Theta ; \mathcal{B} ; GNil \vdash v \Rightarrow t2 \wedge \Theta ; \mathcal{B} ; GNil \vdash t2 \lesssim \llbracket z : b \mid c2 \rrbracket$

using *assms check-v-elim* by *metis*

have *teq*: $t1 = \llbracket z : b \mid c\text{-of } t1 \ z \rrbracket$ 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 $\Theta ; \mathcal{B} ; GNil \vdash \llbracket z : b \mid c\text{-of } t1 \ z \rrbracket \lesssim \llbracket z : b \mid c1 \text{ AND } c2 \rrbracket$ using *subtype-conj t1 t2* by *simp*

hence $\Theta ; \mathcal{B} ; GNil \vdash t1 \lesssim \llbracket z : b \mid c1 \text{ AND } c2 \rrbracket$ using *teq* by *auto*

thus *?thesis* using *t1* using *check-v.intros* by *auto*

qed

lemma *wfG-conj*:

fixes $c1::c$

assumes $\Theta ; \mathcal{B} \vdash_{wf} (x, b, c1 \text{ AND } c2) \#_{\Gamma} \Gamma$

shows $\Theta ; \mathcal{B} \vdash_{wf} (x, b, c1) \#_{\Gamma} \Gamma$

proof(*cases c1* $\in \{TRUE, FALSE\}$)

case *True*

then show *?thesis* using *assms wfG-cons2I wfG-elim wfX-wfY* by *metis*

next

case *False*

then show *?thesis* using *assms wfG-cons1I wfG-elim wfX-wfY wfC-elim*

by (*metis wfG-elim2*)

qed

lemma *check-match*:

fixes $s'::s$ and $s::s$ and $css::branch\text{-}list$ and $cs::branch\text{-}s$

shows $\Theta ; \Phi ; \mathcal{B} ; \Gamma ; \Delta \vdash s \Leftarrow \tau \Longrightarrow \text{True}$ and

$\Theta ; \Phi ; \mathcal{B} ; \Gamma ; \Delta ; tid ; dc ; const ; vcons \vdash cs \Leftarrow \tau \Longrightarrow$

$vcons = V\text{-cons } tid \ dc \ v \Longrightarrow B = \{\mid\} \Longrightarrow G = GNil \Longrightarrow cs = (dc \ x' \Rightarrow s') \Longrightarrow$

$\Theta ; \{\mid\} ; GNil \vdash v \Leftarrow const \Longrightarrow$

$\Theta ; \Phi ; \{\mid\} ; GNil ; \Delta \vdash s'[x'::v]_{sv} \Leftarrow \tau$ and

$\Theta ; \Phi ; \mathcal{B} ; \Gamma ; \Delta ; tid ; dclist ; vcons \vdash css \Leftarrow \tau \Longrightarrow distinct \ (map \ fst \ dclist) \Longrightarrow$

$vcons = V\text{-cons } tid \ dc \ v \Longrightarrow B = \{\mid\} \Longrightarrow (dc, const) \in set \ dclist \Longrightarrow G = GNil \Longrightarrow$

$Some \ (AS\text{-branch } dc \ x' \ s') = lookup\text{-branch } dc \ css \Longrightarrow \Theta ; \{\mid\} ; GNil \vdash v \Leftarrow const \Longrightarrow$

$\Theta ; \Phi ; \{\mid\} ; GNil ; \Delta \vdash s'[x'::v]_{sv} \Leftarrow \tau$

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* $\Theta \ \Phi \ \mathcal{B} \ \Gamma \ \Delta \ tid \ consa \ consta \ va \ cs \ \tau \ dclist \ cssa$)

then obtain xa and sa where $cseq:cs = AS\text{-branch } consa \ xa \ sa$ using *check-branch-s-elim2[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) ∈ 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 Θ Φ B Γ Δ 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 Θ B Γ Δ τ const x Φ tid cons va s)

```

Supporting facts here to make the main body of the proof concise

```

have xf:atom x ≠ τ proof -
  have supp τ ⊆ supp B using wf-sup(4) check-branch-s-branchI atom-dom.simps toSet.simps
  dom.simps by fastforce
  thus ?thesis using fresh-def x-not-in-b-set by blast
qed

```

```

hence τ[x::=v]τv = τ using forget-subst-v subst-v-τ-def by auto
have Δ[x::=v]Δv = Δ 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-sup-nil by metis
hence supp va = {} using ⟨ va = V-cons tid cons v ⟩ v.supp pure-sup by auto

```

```

let ?G = (x, b-of const, [ va ]ce == [ V-cons tid cons [ x ]v ]ce AND c-of const x ) #Γ 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 ; 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 ; B ; GNil ⊢ v ⇒ t' ∧ Θ ; B ; GNil ⊢ t' ⋲ const
  using check-v-elim check-branch-s-branchI by metis
  show Θ ; B ; GNil ⊢ v ⇐ { z : b-of const | [ va ]ce == [ V-cons tid cons [ z ]v ]ce }
  proof(rule check-v-top)

```

show $\Theta ; \mathcal{B} ; GNil \vdash_{wf} \{ z : b\text{-of } const \mid [va]^{ce} == [V\text{-cons } tid \text{ cons } [z]^v]^{ce} \}$
proof(*rule wfG-wfT*)
show $\langle \Theta ; \mathcal{B} \vdash_{wf} (x, b\text{-of } const, ([va]^{ce} == [V\text{-cons } tid \text{ cons } [z]^v]^{ce})) [z::=[x]^v]_{cv} \rangle \#_{\Gamma} GNil$
proof –
have $1: va[z::=[x]^v]_{vv} = va$ **using** *forget-subst-v subst-v-v-def z fresh-prodN by metis*
moreover have $2: \Theta ; \mathcal{B} \vdash_{wf} (x, b\text{-of } const, [va]^{ce} == [V\text{-cons } tid \text{ cons } [x]^v]^{ce} \text{ AND } c\text{-of } const \ x) \#_{\Gamma} GNil$
using *check-branch-s-branchI(17)[THEN check-s-wf] $\langle \Gamma = GNil \rangle$ by auto*
moreover hence $\Theta ; \mathcal{B} \vdash_{wf} (x, b\text{-of } const, [va]^{ce} == [V\text{-cons } tid \text{ cons } [x]^v]^{ce}) \#_{\Gamma} GNil$
using *wfG-conj by metis*
ultimately show *?thesis*
unfolding *subst-cv.simps subst-cev.simps subst-vv.simps by auto*
qed
show $\langle atom \ x \ \# \ ([va]^{ce} == [V\text{-cons } tid \text{ cons } [z]^v]^{ce}) \rangle$ **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 $\langle [va]^{ce} = [V\text{-cons } tid \text{ cons } [z]^v]^{ce} [z::=v]_{cev} \rangle$
using $\langle va = V\text{-cons } tid \text{ cons } v \rangle$ *subst-cev.simps subst-vv.simps by auto*
show $\langle \Theta ; \mathcal{B} ; GNil \vdash v \Leftarrow const \rangle$ **using** *check-branch-s-branchI by auto*
show $supp \ [va]^{ce} \subseteq supp \ \mathcal{B}$ **using** $\langle supp \ va = \{ \} \rangle$ *ce.supp by simp*
qed
show $\Theta ; \mathcal{B} ; GNil \vdash v \Leftarrow \{ z : b\text{-of } const \mid c\text{-of } const \ z \}$
using *check-branch-s-branchI z by auto*
qed

Main body of proof for this case

have $\Theta ; \Phi ; \mathcal{B} ; (?G)[x::=v]_{\Gamma v} ; \Delta[x::=v]_{\Delta v} \vdash s[x::=v]_{sv} \Leftarrow \tau[x::=v]_{\tau v}$
proof(*rule subst-check-check-s*)
show $\langle \Theta ; \mathcal{B} ; GNil \vdash v \Leftarrow \{ z : b\text{-of } const \mid [va]^{ce} == [V\text{-cons } tid \text{ cons } [z]^v]^{ce} \text{ AND } c\text{-of } const \ z \} \rangle$ **using** *vt by auto*
show $\langle atom \ z \ \# \ (x, v) \rangle$ **using** *z fresh-prodN by auto*
show $\langle \Theta ; \Phi ; \mathcal{B} ; ?G ; \Delta \vdash s \Leftarrow \tau \rangle$
using *check-branch-s-branchI by auto*

show $\langle ?G = GNil @ (x, b\text{-of } const, ([va]^{ce} == [V\text{-cons } tid \text{ cons } [z]^v]^{ce} \text{ AND } c\text{-of } const \ z) [z::=[x]^v]_{cv}) \rangle \#_{\Gamma} GNil$
proof –
have $va[z::=[x]^v]_{vv} = va$ **using** *forget-subst-v subst-v-v-def z fresh-prodN by metis*
moreover have $(c\text{-of } const \ z) [z::=[x]^v]_{cv} = c\text{-of } const \ x$
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* $\langle \tau[x::=v]_{\tau v} = \tau \rangle \langle \Delta[x::=v]_{\Delta v} = \Delta \rangle$ **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 $\tau::\tau$ **and** $\Phi::\Phi$ **and** $\Delta::\Delta$ **and** $G::\Gamma$
assumes $\Theta ; \{||\} ; GNil \vdash \llbracket z : B\text{-unit} \mid TRUE \rrbracket \lesssim \tau'$ **and** $\Theta ; \{||\} ; GNil \vdash_{wf} \Delta$ **and** $\Theta \vdash_{wf} \Phi$
and $\Theta ; \{||\} \vdash_{wf} G$
shows $\langle \Theta ; \Phi ; \{||\} ; G ; \Delta \vdash \llbracket L\text{-unit} \rrbracket^v \rangle \Leftarrow \tau'$
proof –
have $*:\Theta ; \{||\} ; GNil \vdash \llbracket L\text{-unit} \rrbracket^v \Rightarrow \llbracket z : B\text{-unit} \mid \llbracket z \rrbracket^v \rrbracket^{ce} == \llbracket \llbracket L\text{-unit} \rrbracket^v \rrbracket^{ce} \rrbracket$
using *infer-l.intros(4) infer-v-litI fresh-GNil assms wfX-wfY* **by** (*meson subtype-g-wf*)
moreover have $\Theta ; \{||\} ; GNil \vdash \llbracket z : B\text{-unit} \mid \llbracket z \rrbracket^v \rrbracket^{ce} == \llbracket \llbracket L\text{-unit} \rrbracket^v \rrbracket^{ce} \rrbracket \lesssim \tau'$
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 toSet.simps*

by (*metis bot.extremum infer-v-g-weakening subtype-weakening wfD-wf*)
qed

lemma *preservation-var*:

shows $\Theta ; \Phi ; \{||\} ; GNil ; \Delta \vdash VAR u : \tau' = v IN s \Leftarrow \tau \Rightarrow \Theta \vdash \delta \sim \Delta \Rightarrow atom u \nVdash \delta \Rightarrow atom u \nVdash \Delta \Rightarrow$

$\Theta ; \Phi ; \{||\} ; GNil ; (u, \tau') \#_{\Delta} \Delta \vdash s \Leftarrow \tau \wedge \Theta \vdash (u, v) \# \delta \sim (u, \tau') \#_{\Delta} \Delta$

and

check-branch-s $\Theta \Phi \{||\} GNil \Delta tid dc const v cs \tau \Rightarrow True$ **and**

check-branch-list $\Theta \Phi \{||\} \Gamma \Delta tid dclist v css \tau \Rightarrow True$

proof(*nominal-induct* $\{||\}::bv fset GNil \Delta VAR u : \tau' = v IN s \tau$ **and** τ **and** τ *rule: check-s-check-branch-s-check-branch-*

case (*check-varI* $u' \Theta \Phi \Delta \tau s'$)

hence $\Theta ; \Phi ; \{||\} ; GNil ; (u, \tau') \#_{\Delta} \Delta \vdash s \Leftarrow \tau$ **using** *check-s-abs-u check-v-wf* **by** *metis*

moreover have $\Theta \vdash ((u, v) \# \delta) \sim ((u, \tau') \#_{\Delta} \Delta)$ **proof**

show $\langle \Theta \vdash \delta \sim \Delta \rangle$ **using** *check-varI* **by** *auto*

show $\langle \Theta ; \{||\} ; GNil \vdash v \Leftarrow \tau' \rangle$ **using** *check-varI* **by** *auto*

show $\langle u \notin fst \text{ ‘ set } \delta \rangle$ **using** *check-varI fresh-d-fst-d* **by** *auto*

qed

ultimately show $?case$ **by** *simp*

qed(*auto+*)

lemma *check-while*:

shows $\Theta ; \Phi ; \{||\} ; GNil ; \Delta \vdash WHILE s1 DO \{ s2 \} \Leftarrow \tau \Rightarrow atom x \nVdash (s1, s2) \Rightarrow atom z' \nVdash x \Rightarrow$

$\Theta ; \Phi ; \{||\} ; GNil ; \Delta \vdash LET x : (\llbracket z' : B\text{-bool} \mid TRUE \rrbracket) = s1 IN (IF (V\text{-var } x) THEN (s2 ;; (WHILE s1 DO \{s2\})))$

$ELSE (\llbracket V\text{-lit } L\text{-unit} \rrbracket^s)) \Leftarrow \tau$ **and**

$check\text{-}branch\text{-}s \ \Theta \ \Phi \ \{\|\} \ GNil \ \Delta \ tid \ dc \ const \ v \ cs \ \tau \implies True$ **and**
 $check\text{-}branch\text{-}list \ \Theta \ \Phi \ \{\|\} \ \Gamma \ \Delta \ tid \ dclist \ v \ css \ \tau \implies True$
proof(*nominal-induct* $\{\|\}::bv \ fset \ GNil \ \Delta \ AS\text{-}while \ s1 \ s2 \ \tau$ **and** τ **and** τ *avoiding: s1 s2 x z' rule:*
check-s-check-branch-s-check-branch-list.strong-induct)
case ($check\text{-}whileI \ \Theta \ \Phi \ \Delta \ s1 \ z \ s2 \ \tau'$)
have $teq:\{\|\ z' : B\text{-}bool \mid TRUE \} = \{\|\ z : B\text{-}bool \mid TRUE \}$ **using** $\tau.eq\text{-}iff$ **by** *auto*

show *?case proof*
have $atom \ x \ \# \ \tau'$ **using** *wfT-nil-supp fresh-def subtype-wfT check-whileI(5)* **by** *fast*
moreover **have** $atom \ x \ \# \ \{\|\ z' : B\text{-}bool \mid TRUE \}$ **using** $\tau.fresh \ c.fresh \ b.fresh$ **by** *metis*
ultimately **show** $\langle atom \ x \ \# \ (\Theta, \Phi, \{\|\}, GNil, \Delta, \{\|\ z' : B\text{-}bool \mid TRUE \}, s1, \tau') \rangle$
apply(*unfold fresh-prodN*)
using *check-whileI wb-x-fresh check-s-wf wfD-x-fresh fresh-empty-fset fresh-GNil fresh-Pair* $\langle atom \ x \ \# \ \tau' \rangle$ **by** *metis*

show $\langle \Theta ; \Phi ; \{\|\} ; GNil ; \Delta \vdash s1 \Leftarrow \{\|\ z' : B\text{-}bool \mid TRUE \} \rangle$ **using** *check-whileI teq* **by** *metis*

let $?G = (x, b\text{-}of \ \{\|\ z' : B\text{-}bool \mid TRUE \}, c\text{-}of \ \{\|\ z' : B\text{-}bool \mid TRUE \} \ x) \ \#_{\Gamma} \ GNil$

have $c\text{-}of : (c\text{-}of \ \{\|\ z' : B\text{-}bool \mid TRUE \} \ x) = C\text{-}true$ **using** *c-of.simps check-whileI subst-cv.simps*
by *metis*
have $wfg : \Theta ; \{\|\} \vdash_{wf} ?G$ **proof**
show $c\text{-}of \ \{\|\ z' : B\text{-}bool \mid TRUE \} \ x \in \{TRUE, FALSE\}$ **using** *subst-cv.simps cof* **by** *auto*
show $\Theta ; \{\|\} \vdash_{wf} GNil$ **using** *wfG-nilI check-whileI wfX-wfY check-s-wf* **by** *metis*
show $atom \ x \ \# \ GNil$ **using** *fresh-GNil* **by** *auto*
show $\Theta ; \{\|\} \vdash_{wf} b\text{-}of \ \{\|\ z' : B\text{-}bool \mid TRUE \}$ **using** *wfB-boolI wfX-wfY check-s-wf b-of.simps*
by (*metis* $\langle \Theta ; \{\|\} \vdash_{wf} GNil \rangle$)
qed

obtain $zz::x$ **where** $zf:\langle atom \ zz \ \# \ ((\Theta, \Phi, \{\|\}::bv \ fset, ?G, \Delta, [x]^v,$
 $AS\text{-}seq \ s2 \ (AS\text{-}while \ s1 \ s2), AS\text{-}val \ [L\text{-}unit]^v, \tau'), x, ?G \rangle$
using *obtain-fresh* **by** *blast*
show $\langle \Theta ; \Phi ; \{\|\} ; ?G ; \Delta \vdash$
 $AS\text{-}if \ [x]^v \ (AS\text{-}seq \ s2 \ (AS\text{-}while \ s1 \ s2)) \ (AS\text{-}val \ [L\text{-}unit]^v) \Leftarrow \tau' \rangle$
proof
show $atom \ zz \ \# \ (\Theta, \Phi, \{\|\}::bv \ fset, ?G, \Delta, [x]^v, AS\text{-}seq \ s2 \ (AS\text{-}while \ s1 \ s2), AS\text{-}val \ [L\text{-}unit]^v,$
 $\tau') \text{ using } zf \text{ by } auto$
show $\langle \Theta ; \{\|\} ; ?G \vdash [x]^v \Leftarrow \{\|\ zz : B\text{-}bool \mid TRUE \} \rangle$ **proof**
have $atom \ zz \ \# \ x \wedge atom \ zz \ \# \ (\Theta, \{\|\}::bv \ fset, ?G)$ **using** *zf fresh-prodN* **by** *metis*
thus $\langle \Theta ; \{\|\} ; ?G \vdash [x]^v \Rightarrow \{\|\ zz : B\text{-}bool \mid [[zz]^v]^{ce} == [[x]^v]^{ce} \} \rangle$
using *infer-v-varI lookup.simps wfg b-of.simps* **by** *metis*
thus $\langle \Theta ; \{\|\} ; ?G \vdash \{\|\ zz : B\text{-}bool \mid [[zz]^v]^{ce} == [[x]^v]^{ce} \} \lesssim \{\|\ zz : B\text{-}bool \mid TRUE \} \rangle$
using *subtype-top infer-v-wf* **by** *metis*
qed
show $\langle \Theta ; \Phi ; \{\|\} ; ?G ; \Delta \vdash AS\text{-}seq \ s2 \ (AS\text{-}while \ s1 \ s2) \Leftarrow \{\|\ zz : B\text{-}unit \mid TRUE \}$
 $L\text{-}true]^v]^{ce} \ IMP \ c\text{-}of \ \tau' \ zz \ \} \rangle$
proof
have $\{\|\ zz : B\text{-}unit \mid TRUE \} = \{\|\ z : B\text{-}unit \mid TRUE \}$ **using** $\tau.eq\text{-}iff$ **by** *auto*
thus $\langle \Theta ; \Phi ; \{\|\} ; ?G ; \Delta \vdash s2 \Leftarrow \{\|\ zz : B\text{-}unit \mid TRUE \} \rangle$ **using** *check-s-g-weakening(1)*
 $[OF \ check\text{-}whileI(3) - wfg] \ toSet.simps$
by (*simp add: $\langle \{\|\ zz : B\text{-}unit \mid TRUE \} = \{\|\ z : B\text{-}unit \mid TRUE \} \rangle$*)

```

    show ⟨Θ; Φ; {||}; ?G; Δ ⊢ AS-while s1 s2 ⇐ {||} 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 ⊢ AS-while s1 s2 ⇐ τ'⟩ using check-whileI by auto
    thus *:⟨Θ; Φ; {||}; ?G; Δ ⊢ AS-while s1 s2 ⇐ τ'⟩ using check-s-g-weakening(1)[OF - -
wfg] toSet.simps by auto

  show ⟨Θ; {||}; ?G ⊢ τ' ≲ {||} 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 ⊢wf τ'⟩ using * check-s-wf by auto
    thm wfT-wfT-if-rev
    show ⟨Θ; {||}; ?G ⊢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 # τ'⟩ using zf fresh-prodN by metis
  qed
  qed

  qed
  show ⟨Θ; Φ; {||}; ?G; Δ ⊢ AS-val [ L-unit ]v ⇐ {||} 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 ⊢ AS-val [ L-unit ]v ⇐ τ'⟩
    using check-unit[OF check-whileI(5) - - wfg] using check-whileI wfg wfX-wfY check-s-wf by
  metis

  show ⟨Θ; {||}; ?G ⊢ τ' ≲ {||} 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+)

```

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 $?v = V\text{-lit } L\text{-true}$
obtain $z::x$ **where** z : $\text{atom } z \# (x, [L\text{-true}]^v, c)$ **using** *obtain-fresh by metis*
have $\text{atom } z \# c$ **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 $\Theta ; \Phi ; \mathcal{B} ; ((x, B\text{-bool}, c) \#_{\Gamma} \Gamma)[x::=?v]_{\Gamma v} ; \Delta[x::=?v]_{\Delta v} \vdash s[x::=?v]_{sv} \Leftarrow \tau[x::=?v]_{\tau v}$
proof(*rule subst-infer-check-s(1)*)
show $\langle \Theta ; \mathcal{B} ; \Gamma \vdash [L\text{-true}]^v \Rightarrow \{ z : B\text{-bool} \mid (CE\text{-val } (V\text{-var } z)) = (CE\text{-val } (V\text{-lit } L\text{-true})) \} \rangle$
using *infer-v-litI check-s-wf wfG-elim(2) infer-trueI assms by metis*
show $\langle \Theta ; \mathcal{B} ; \Gamma \vdash \{ z : B\text{-bool} \mid (CE\text{-val } (V\text{-var } z)) = (CE\text{-val } (V\text{-lit } L\text{-true})) \} \lesssim \{ z : B\text{-bool} \mid c[x::=[z]^v]_v \} \rangle$ **proof**
show $\langle \text{atom } x \# (\Theta, \mathcal{B}, \Gamma, z, [[z]^v]^{ce} == [[L\text{-true}]^v]^{ce}, z, c[x::=[z]^v]_v) \rangle$
apply(*unfold fresh-prodN, intro conjI*)
prefer 5
using $c.\text{fresh } ce.\text{fresh } v.\text{fresh } z$ *fresh-prodN* **apply** $\text{auto}[1]$
prefer 6
using $\text{fresh-subst-v-if}[of \text{atom } x \ c \ x]$ *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 $\langle \Theta ; \mathcal{B} ; \Gamma \vdash_{wf} \{ z : B\text{-bool} \mid [[z]^v]^{ce} == [[L\text{-true}]^v]^{ce} \} \rangle$ **using** vt *infer-v-wf by simp*
show $\langle \Theta ; \mathcal{B} ; \Gamma \vdash_{wf} \{ z : B\text{-bool} \mid c[x::=[z]^v]_v \} \rangle$ **proof**(*rule wfG-wfT*)
show $\langle \Theta ; \mathcal{B} \vdash_{wf} (x, B\text{-bool}, c[x::=[z]^v]_v[z::=[x]^v]_{cv}) \#_{\Gamma} \Gamma \rangle$ **using** c *check-s-wf assms by metis*
have $\text{atom } x \# [z]^v$ **using** $v.\text{fresh } z$ *fresh-at-base by auto*
thus $\langle \text{atom } x \# c[x::=[z]^v]_v \rangle$ **using** $\text{fresh-subst-v-if}[of \text{atom } x \ c]$ **by** *auto*
qed
have $wfg: \Theta ; \mathcal{B} \vdash_{wf} (x, B\text{-bool}, ([z]^v]^{ce} == [[L\text{-true}]^v]^{ce})[z::=[x]^v]_v) \#_{\Gamma} \Gamma$
using $wfT\text{-wfG } vt$ *infer-v-wf fresh-prodN assms by simp*
show $\langle \Theta ; \mathcal{B} ; (x, B\text{-bool}, ([z]^v]^{ce} == [[L\text{-true}]^v]^{ce})[z::=[x]^v]_v) \#_{\Gamma} \Gamma \models c[x::=[z]^v]_v[z::=[x]^v]_{cv} \rangle$
using c *valid-weakening[OF assms(3) - wfg] toSet.simps*
using *subst-v-c-def by auto*
qed
show $\langle \text{atom } z \# (x, [L\text{-true}]^v) \rangle$ **using** z *fresh-prodN by metis*
show $\langle \Theta ; \Phi ; \mathcal{B} ; (x, B\text{-bool}, c) \#_{\Gamma} \Gamma ; \Delta \vdash s \Leftarrow \tau \rangle$ **using** *assms by auto*
thus $\langle (x, B\text{-bool}, c) \#_{\Gamma} \Gamma = GNil @ (x, B\text{-bool}, c[x::=[z]^v]_v[z::=[x]^v]_{cv}) \#_{\Gamma} \Gamma \rangle$ **using** *append-g.simps*
c by auto
qed
moreover **have** $((x, B\text{-bool}, c) \#_{\Gamma} \Gamma)[x::=?v]_{\Gamma v} = \Gamma$ **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 $\Theta; \Phi; \{\|\}; GNil; \Delta \vdash AS\text{-}assert\ c\ s \Leftarrow \tau$
shows $\Theta; \Phi; \{\|\}; GNil; \Delta \vdash s \Leftarrow \tau \wedge \Theta; \{\|\}; GNil \models c$
proof –
let $?B = (\{\|\}::bv\ fset)$
let $?v = V\text{-}lit\ L\text{-}true$

obtain x **where** $x: \Theta; \Phi; ?B; (x, B\text{-}bool, c) \#_{\Gamma} GNil; \Delta \vdash s \Leftarrow \tau \wedge atom\ x \# (\Theta, \Phi, ?B, GNil, \Delta, c, \tau, s) \wedge \Theta; ?B; GNil \models c$
using $check\text{-}s\text{-}elims(10)[OF\ (\Theta; \Phi; ?B; GNil; \Delta \vdash AS\text{-}assert\ c\ s \Leftarrow \tau)]\ valid.simps$ **by** $metis$

show $?thesis$ **using** $assms\ check\text{-}s\text{-}narrow\ x$ **by** $metis$
qed

16.2.5 Main Lemma

thm $infer\text{-}v\text{-}pairI$

lemma $infer\text{-}v\text{-}pair2I$:

$atom\ z \# (v1, v2) \implies$
 $atom\ z \# (\Theta, \mathcal{B}, \Gamma) \implies$
 $\Theta; \mathcal{B}; \Gamma \vdash v1 \Rightarrow t1 \implies$
 $\Theta; \mathcal{B}; \Gamma \vdash v2 \Rightarrow t2 \implies$
 $b1 = b\text{-}of\ t1 \implies b2 = b\text{-}of\ t2 \implies$
 $\Theta; \mathcal{B}; \Gamma \vdash [v1, v2]^v \Rightarrow \llbracket z : [b1, b2]^b \mid [[z]^v]^{ce} == [[v1, v2]^v]^{ce} \rrbracket$
using $infer\text{-}v\text{-}pairI$ **by** $simp$

lemma $preservation$:

assumes $\Phi \vdash \langle \delta, s \rangle \longrightarrow \langle \delta', s' \rangle$ **and** $\Theta; \Phi; \Delta \vdash \langle \delta, s \rangle \Leftarrow \tau$
shows $\exists \Delta'. \Theta; \Phi; \Delta' \vdash \langle \delta', s' \rangle \Leftarrow \tau \wedge \Delta \sqsubseteq \Delta'$
using $assms$

proof($induct\ arbitrary: \tau\ rule: reduce\text{-}stmt.induct$)

case ($reduce\text{-}let\text{-}plusI\ \delta\ x\ n1\ n2\ s'$)
then show $?case$ **using** $preservation\text{-}plus$
by ($metis\ order\text{-}refl$)

next

case ($reduce\text{-}let\text{-}leqI\ b\ n1\ n2\ \delta\ x\ s$)
then show $?case$ **using** $preservation\text{-}leq$ **by** ($metis\ order\text{-}refl$)

next

case ($reduce\text{-}let\text{-}appI\ f\ z\ b\ c\ \tau'\ s'\ \Phi\ \delta\ x\ v\ s$)
hence $tt: \Theta; \Phi; \{\|\}; GNil; \Delta \vdash AS\text{-}let\ x\ (AE\text{-}app\ f\ v)\ s \Leftarrow \tau \wedge \Theta \vdash \delta \sim \Delta \wedge (\forall fd \in set\ \Phi. check\text{-}fundef\ \Theta\ \Phi\ fd)$ **using** $config\text{-}type\text{-}elims[OF\ reduce\text{-}let\text{-}appI(2)]$ **by** $metis$
hence $*:\Theta; \Phi; \{\|\}; GNil; \Delta \vdash AS\text{-}let\ x\ (AE\text{-}app\ f\ v)\ s \Leftarrow \tau$ **by** $auto$

hence $\Theta; \Phi; \{\|\}; GNil; \Delta \vdash AS\text{-}let2\ x\ (\tau'[z::=v]_{\tau v})\ (s'[z::=v]_{sv})\ s \Leftarrow \tau$
using $preservation\text{-}app\ reduce\text{-}let\text{-}appI\ tt$ **by** $auto$

hence $\Theta; \Phi; \Delta \vdash \langle \delta, AS\text{-}let2\ x\ (\tau'[z::=v]_{\tau v})\ s'[z::=v]_{sv}\ s \rangle \Leftarrow \tau$ **using** $config\text{-}typeI\ tt$ **by** $auto$
then show $?case$ **using** $tt\ order\text{-}refl\ reduce\text{-}let\text{-}appI$ **by** $metis$

next

case ($reduce\text{-}let\text{-}appPI\ f\ bv\ z\ b\ c\ \tau'\ s'\ \Phi\ \delta\ x\ b'\ v\ s$)

hence $tt: \Theta; \Phi; \{\|\}; GNil; \Delta \vdash AS\text{-}let\ x\ (AE\text{-}appP\ f\ b'\ v)\ s \Leftarrow \tau \wedge \Theta \vdash \delta \sim \Delta \wedge (\forall fd \in set\ \Phi.$

$check_fundef \ \Theta \ \Phi \ fd$ **using** $config_type_elims[OF \ reduce_let_appPI(2)]$ **by** $metis$
hence $*, \Theta; \Phi; \{\|\}; GNil; \Delta \vdash AS\text{-}let \ x \ (AE\text{-}appP \ f \ b' \ v) \ s \Leftarrow \tau$ **by** $auto$

have $\Theta; \Phi; \{\|\}; GNil; \Delta \vdash AS\text{-}let2 \ x \ (\tau'[bv::=b]_{\tau b}[z::=v]_{\tau v}) \ (s'[bv::=b]_{sb}[z::=v]_{sv}) \ s \Leftarrow \tau$
proof($rule \ preservation\text{-}poly\text{-}app$)
show $\langle Some \ (AF\text{-}fundef \ f \ (AF\text{-}fun\text{-}typ\text{-}some \ bv \ (AF\text{-}fun\text{-}typ \ z \ b \ c \ \tau' \ s')) \rangle = lookup\text{-}fun \ \Phi \ f$ **using**
 $reduce_let_appPI$ **by** $metis$
show $\langle \forall fd \in set \ \Phi. \ check_fundef \ \Theta \ \Phi \ fd \rangle$ **using** $tt \ lookup\text{-}fun\text{-}member$ **by** $metis$
show $\langle \Theta; \Phi; \{\|\}; GNil; \Delta \vdash AS\text{-}let \ x \ (AE\text{-}appP \ f \ b' \ v) \ s \Leftarrow \tau \rangle$ **using** $*$ **by** $auto$
show $\langle \Theta; \{\|\} \vdash_{wf} b' \rangle$ **using** $check\text{-}s\text{-}elims \ infer\text{-}e\text{-}wf \ wfE\text{-}elims \ *$ **by** $metis$
qed($auto+$)

hence $\Theta; \Phi; \Delta \vdash \langle \delta, AS\text{-}let2 \ x \ (\tau'[bv::=b]_{\tau b}[z::=v]_{\tau v}) \ s'[bv::=b]_{sb}[z::=v]_{sv} \ s \rangle \Leftarrow \tau$ **using**
 $config_typeI \ tt$ **by** $auto$
then show $?case$ **using** $tt \ order\text{-}refl \ reduce_let_appI$ **by** $metis$

next
case ($reduce\text{-}if\text{-}trueI \ \delta \ s1 \ s2$)
then show $?case$ **using** $preservation\text{-}if$ **by** $metis$

next
case ($reduce\text{-}if\text{-}falseI \ uw \ \delta \ s1 \ s2$)
then show $?case$ **using** $preservation\text{-}if$ **by** $metis$

next
case ($reduce\text{-}let\text{-}valI \ \delta \ x \ v \ s$)
then show $?case$ **using** $preservation\text{-}let\text{-}val$ **by** $presburger$

next
case ($reduce\text{-}let\text{-}mvar \ u \ v \ \delta \ \Phi \ x \ s$)
hence $*, \Theta; \Phi; \{\|\}; GNil; \Delta \vdash AS\text{-}let \ x \ (AE\text{-}mvar \ u) \ s \Leftarrow \tau \wedge \Theta \vdash \delta \sim \Delta \wedge (\forall fd \in set \ \Phi. \ check_fundef \ \Theta \ \Phi \ fd)$
using $config_type_elims$ **by** $blast$

hence $*, \Theta; \Phi; \{\|\}; GNil; \Delta \vdash AS\text{-}let \ x \ (AE\text{-}mvar \ u) \ s \Leftarrow \tau$ **by** $auto$
obtain $xa::x$ **and** $za::x$ **and** $ca::c$ **and** $ba::b$ **and** $sa::s$ **where**
 $sa1: atom \ xa \ \# \ (\Theta, \Phi, \{\|\}::bv \ fset, GNil, \Delta, AE\text{-}mvar \ u, \tau) \wedge atom \ za \ \# \ (xa, \Theta, \Phi, \{\|\}::bv \ fset, GNil, \Delta, AE\text{-}mvar \ u, \tau, sa) \wedge$
 $\Theta; \Phi; \{\|\}; GNil; \Delta \vdash AE\text{-}mvar \ u \Rightarrow \llbracket za : ba \mid ca \rrbracket \wedge$
 $\Theta; \Phi; \{\|\}; (xa, ba, ca[z::=V\text{-}var \ xa]_{cv}) \#_{\Gamma} GNil; \Delta \vdash sa \Leftarrow \tau \wedge$
 $(\forall c. atom \ c \ \# \ (s, sa) \longrightarrow atom \ c \ \# \ (x, xa, s, sa) \longrightarrow (x \leftrightarrow c) \cdot s = (xa \leftrightarrow c) \cdot sa)$
using $check\text{-}s\text{-}elims(2)[OF \ **] \ subst\text{-}defs$ **by** $metis$

have $\Theta; \{\|\}; GNil \vdash v \Leftarrow \llbracket za : ba \mid ca \rrbracket$ **proof** –
have $(u, \llbracket za : ba \mid ca \rrbracket) \in setD \ \Delta$ **using** $infer\text{-}e\text{-}elims(11) \ sa1$ **by** $fast$
thus $?thesis$ **using** $\delta\text{-}sim\text{-}v \ reduce_let\text{-}mvar \ config_type_elims \ check\text{-}s\text{-}wf$ **by** $metis$
qed

then obtain τ' **where** $vst: \Theta; \{\|\}; GNil \vdash v \Rightarrow \tau' \wedge$
 $\Theta; \{\|\}; GNil \vdash \tau' \lesssim \llbracket za : ba \mid ca \rrbracket$ **using** $check\text{-}v\text{-}elims$ **by** $blast$

obtain $za2$ **and** $ba2$ **and** $ca2$ **where** $zbc: \tau' = (\llbracket za2 : ba2 \mid ca2 \rrbracket) \wedge atom \ za2 \ \# \ (xa, (xa, \Theta, \Phi, \{\|\}::bv \ fset, GNil, \Delta, AE\text{-}val \ v, \tau, sa))$
using $obtain\text{-}fresh\text{-}z$ **by** $blast$
have $beq: ba=ba2$ **using** $subtype\text{-}eq\text{-}base \ vst \ zbc$ **by** $blast$

moreover have $xaf : atom\ x \# (za, za2)$
apply(*unfold fresh-prodN, intro conjI*)
using $sa1\ zbc\ fresh-prodN\ fresh-x-neq$ **by** *metis+*

have $sat2 : \Theta ; \Phi ; \{\|\}; GNil @ (xa, ba, ca2[za2::=V-var\ xa]_{cv}) \#_{\Gamma} GNil ; \Delta \vdash sa \Leftarrow \tau$ **proof**(*rule ctx-subtype-s*)
show $\Theta ; \Phi ; \{\|\}; GNil @ (xa, ba, ca[za::=V-var\ xa]_{cv}) \#_{\Gamma} GNil ; \Delta \vdash sa \Leftarrow \tau$ **using** $sa1$ **by** *auto*
show $\Theta ; \{\|\}; GNil \vdash \llbracket za2 : ba \mid ca2 \rrbracket \lesssim \llbracket za : ba \mid ca \rrbracket$ **using** $beq\ zbc\ vst$ **by** *fast*
show $atom\ xa \# (za, za2, ca, ca2)$ **proof** –
have $* : \Theta ; \{\|\}; GNil \vdash_{wf} (\llbracket za2 : ba2 \mid ca2 \rrbracket)$ **using** $zbc\ vst\ subtype-wf$ **by** *auto*
hence $supp\ ca2 \subseteq \{ atom\ za2 \}$ **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 $* : \Theta ; \{\|\}; GNil \vdash_{wf} (\llbracket za : ba \mid ca \rrbracket)$ **using** $zbc\ vst\ subtype-wf$ **by** *auto*
hence $supp\ ca \subseteq \{ atom\ za \}$ **using** $wfT-supp\ \tau.supp$ **by** *force*
moreover have $xa \neq za$ **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 $dwf : \Theta ; \{\|\}; GNil \vdash_{wf} \Delta$ **using** $sa1\ infer-e-wf$ **by** *meson*

have $\Theta ; \Phi ; \{\|\}; GNil ; \Delta \vdash AS-let\ xa\ (AE-val\ v)\ sa \Leftarrow \tau$ **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 $atom\ xa \# (\Theta, \Phi, \{\|\}::bv\ fset, GNil, \Delta, AE-val\ v, \tau)$ **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 $atom\ za2 \# (xa, \Theta, \Phi, \{\|\}::bv\ fset, GNil, \Delta, AE-val\ v, \tau, sa)$ **using** $zbc\ freshers\ fresh-prodN$
by *auto*
have $\Theta \vdash_{wf} \Phi$ **using** $sa1\ infer-e-wf$ **by** *auto*
thus $\Theta ; \Phi ; \{\|\}; GNil ; \Delta \vdash AE-val\ v \Rightarrow \llbracket za2 : ba \mid ca2 \rrbracket$
using $zbc\ vst\ beq\ dwf\ infer-e-valI$ **by** *blast*
show $\Theta ; \Phi ; \{\|\}; (xa, ba, ca2[za2::=V-var\ xa]_v) \#_{\Gamma} GNil ; \Delta \vdash sa \Leftarrow \tau$ **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 $\Theta ; \Phi ; \{\|\}; GNil ; \Delta \vdash AS-let\ x\ (AE-val\ v)\ s \Leftarrow \tau$ **by** *auto*

then show *?case* **using** $reduce-let-mvar\ * \ config-typeI$
by (*meson order-refl*)
next
case ($reduce-let2I\ \Phi\ \delta\ s1\ \delta'\ s1'\ x\ t\ s2$)

hence $** : \Theta; \Phi; \{\|\}; GNil; \Delta \vdash AS\text{-}let2\ x\ t\ s1\ s2 \Leftarrow \tau \wedge \Theta \vdash \delta \sim \Delta \wedge (\forall fd \in set\ \Phi. check\text{-}fundef\ \Theta\ \Phi\ fd)$ **using** *config-type-elim*_[OF reduce-let2I(3)] **by** *blast*
hence $* : \Theta; \Phi; \{\|\}; GNil; \Delta \vdash AS\text{-}let2\ x\ t\ s1\ s2 \Leftarrow \tau$ **by** *auto*

obtain $xa :: x$ **and** $z :: x$ **and** c **and** b **and** $s2a :: s$ **where** $st : atom\ xa \# (\Theta, \Phi, \{\|\} :: bv\ fset, GNil, \Delta, t, s1, \tau) \wedge$

$\Theta; \Phi; \{\|\}; GNil; \Delta \vdash s1 \Leftarrow t \wedge$
 $\Theta; \Phi; \{\|\}; (xa, b\text{-}of\ t, c\text{-}of\ t\ xa) \#_{\Gamma} GNil; \Delta \vdash s2a \Leftarrow \tau \wedge ([[atom\ x]]lst. s2 = [[atom\ xa]]lst. s2a)$

using *check-s-elim*₍₄₎_[OF *] *Abs1-eq-iff-all(3)* **by** *metis*

hence $\Theta; \Phi; \Delta \vdash \langle \delta, s1 \rangle \Leftarrow t$ **using** *config-typeI* ****** **by** *auto*

then obtain Δ' **where** $s1r : \Theta; \Phi; \Delta' \vdash \langle \delta', s1' \rangle \Leftarrow t \wedge \Delta \sqsubseteq \Delta'$ **using** *reduce-let2I* **by** *presburger*

have $\Theta; \Phi; \{\|\}; GNil; \Delta' \vdash AS\text{-}let2\ xa\ t\ s1'\ s2a \Leftarrow \tau$

proof(*rule check-let2I*)

show $* : \Theta; \Phi; \{\|\}; GNil; \Delta' \vdash s1' \Leftarrow t$ **using** *config-type-elim*_{st s1r} **by** *metis*

show $atom\ xa \# (\Theta, \Phi, \{\|\} :: bv\ fset, GNil, \Delta', t, s1', \tau)$ **proof** $-$

have $atom\ xa \# s1'$ **using** *check-s-x-fresh ** **by** *auto*

moreover **have** $atom\ xa \# \Delta'$ **using** *check-s-x-fresh ** **by** *auto*

ultimately show *?thesis* **using** *st fresh-prodN* **by** *metis*

qed

show $\Theta; \Phi; \{\|\}; (xa, b\text{-}of\ t, c\text{-}of\ t\ xa) \#_{\Gamma} GNil; \Delta' \vdash s2a \Leftarrow \tau$ **proof** $-$

have $\Theta; \{\|\}; GNil \vdash_{wf} \Delta'$ **using** ** check-s-wf* **by** *auto*

moreover **have** $\Theta; \{\|\} \vdash_{wf} ((xa, b\text{-}of\ t, c\text{-}of\ t\ xa) \#_{\Gamma} GNil)$ **using** *st check-s-wf* **by** *auto*

ultimately have $\Theta; \{\|\}; ((xa, b\text{-}of\ t, c\text{-}of\ t\ xa) \#_{\Gamma} GNil) \vdash_{wf} \Delta'$ **using** *wf-weakening* **by** *auto*

thus *?thesis* **using** *check-s-d-weakening check-s-wf st s1r* **by** *metis*

qed

qed

moreover **have** $AS\text{-}let2\ xa\ t\ s1'\ s2a = AS\text{-}let2\ x\ t\ s1'\ s2$ **using** *st s-branch-s-branch-list.eq-iff* **by** *metis*

ultimately have $\Theta; \Phi; \{\|\}; GNil; \Delta' \vdash AS\text{-}let2\ x\ t\ s1'\ s2 \Leftarrow \tau$ **using** *st* **by** *argo*

moreover **have** $\Theta \vdash \delta' \sim \Delta'$ **using** *config-type-elim 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 $** : \Theta; \Phi; \{\|\}; GNil; \Delta \vdash AS\text{-}var\ u\ \tau'\ v\ s \Leftarrow \tau \wedge \Theta \vdash \delta \sim \Delta \wedge (\forall fd \in set\ \Phi. check\text{-}fundef\ \Theta\ \Phi\ fd)$

using *config-type-elim* **by** *meson*

have $uf : atom\ u \# \Delta$ **using** *reduce-varI delta-sim-fresh* **by** *force*

hence $* : \Theta; \Phi; \{\|\}; GNil; \Delta \vdash AS\text{-}var\ u\ \tau'\ v\ s \Leftarrow \tau$ **and** $\Theta \vdash \delta \sim \Delta$ **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* $\Phi \delta u v$)
hence $*$: $\Theta; \Phi; \{\|\}; GNil; \Delta \vdash AS\text{-}assign\ u\ v \Leftarrow \tau \wedge \Theta \vdash \delta \sim \Delta \wedge (\forall fd \in set\ \Phi. check\text{-}fundef\ \Theta\ \Phi\ fd)$
using *config-type-elim* **by** *meson*
then obtain z **and** τ' **where** zt : $\Theta; \{\|\}; GNil \vdash (\llbracket z : B\text{-}unit \mid TRUE \rrbracket) \lesssim \tau \wedge (u, \tau') \in setD\ \Delta \wedge \Theta; \{\|\}; GNil \vdash v \Leftarrow \tau' \wedge \Theta; \{\|\}; GNil \vdash_{wf} \Delta$
using *check-s-elim*(8) **by** *metis*
hence $\Theta \vdash update\text{-}d\ \delta\ u\ v \sim \Delta$ **using** *update-d-sim* ***** **by** *metis*
moreover have $\Theta; \Phi; \{\|\}; GNil; \Delta \vdash AS\text{-}val\ (V\text{-}lit\ L\text{-}unit) \Leftarrow \tau$ **using** $zt * check\text{-}s\text{-}v\text{-}unit\ check\text{-}s\text{-}wf$
by *auto*
ultimately show $?case$ **using** *config-typeI* ***** **by** (*meson order-refl*)
next
case (*reduce-seq1I* $\Phi \delta s$)
hence $\Theta; \Phi; \{\|\}; GNil; \Delta \vdash s \Leftarrow \tau \wedge \Theta \vdash \delta \sim \Delta \wedge (\forall fd \in set\ \Phi. check\text{-}fundef\ \Theta\ \Phi\ fd)$
using *check-s-elim config-type-elim* **by** *force*
then show $?case$ **using** *config-typeI* **by** *blast*
next
case (*reduce-seq2I* $s1\ v\ \Phi\ \delta\ s1'\ s2$)
hence tt : $\Theta; \Phi; \{\|\}; GNil; \Delta \vdash AS\text{-}seq\ s1\ s2 \Leftarrow \tau \wedge \Theta \vdash \delta \sim \Delta \wedge (\forall fd \in set\ \Phi. check\text{-}fundef\ \Theta\ \Phi\ fd)$
using *config-type-elim* **by** *blast*
then obtain z **where** zz : $\Theta; \Phi; \{\|\}; GNil; \Delta \vdash s1 \Leftarrow (\llbracket z : B\text{-}unit \mid TRUE \rrbracket) \wedge \Theta; \Phi; \{\|\}; GNil; \Delta \vdash s2 \Leftarrow \tau$
using *check-s-elim* **by** *blast*
hence $\Theta; \Phi; \Delta \vdash \langle \delta, s1 \rangle \Leftarrow (\llbracket z : B\text{-}unit \mid TRUE \rrbracket)$
using $tt\ config\text{-}typeI\ tt$ **by** *simp*
then obtain Δ' **where** $*$: $\Theta; \Phi; \Delta' \vdash \langle \delta', s1' \rangle \Leftarrow (\llbracket z : B\text{-}unit \mid TRUE \rrbracket) \wedge \Delta \sqsubseteq \Delta'$
using *reduce-seq2I* **by** *meson*
moreover hence $s't$: $\Theta; \Phi; \{\|\}; GNil; \Delta' \vdash s1' \Leftarrow (\llbracket z : B\text{-}unit \mid TRUE \rrbracket) \wedge \Theta \vdash \delta' \sim \Delta'$
using *config-type-elim* **by** *force*
moreover hence $\Theta; \{\|\}; GNil \vdash_{wf} \Delta'$ **using** *check-s-wf* **by** *meson*
moreover hence $\Theta; \Phi; \{\|\}; GNil; \Delta' \vdash s2 \Leftarrow \tau$
using *calculation*(1) $zz\ check\text{-}s\text{-}d\text{-}weakening$ ***** **by** *metis*
moreover hence $\Theta; \Phi; \{\|\}; GNil; \Delta' \vdash (AS\text{-}seq\ s1'\ s2) \Leftarrow \tau$
using *check-seqI* $zz\ s't$ **by** *meson*
ultimately have $\Theta; \Phi; \Delta' \vdash \langle \delta', AS\text{-}seq\ s1'\ s2 \rangle \Leftarrow \tau \wedge \Delta \sqsubseteq \Delta'$
using $zz\ config\text{-}typeI\ tt$ **by** *meson*
then show $?case$ **by** *meson*
next
case (*reduce-whileI* $x\ s1\ s2\ z'\ \Phi\ \delta$)
hence $*$: $\Theta; \Phi; \{\|\}; GNil; \Delta \vdash AS\text{-}while\ s1\ s2 \Leftarrow \tau \wedge \Theta \vdash \delta \sim \Delta \wedge (\forall fd \in set\ \Phi. check\text{-}fundef\ \Theta\ \Phi\ fd)$
using *config-type-elim* **by** *meson*
hence $*$: $\Theta; \Phi; \{\|\}; GNil; \Delta \vdash AS\text{-}while\ s1\ s2 \Leftarrow \tau$ **by** *auto*
hence $\Theta; \Phi; \{\|\}; GNil; \Delta \vdash AS\text{-}let2\ x\ (\llbracket z' : B\text{-}bool \mid TRUE \rrbracket)\ s1\ (AS\text{-}if\ (V\text{-}var\ x)\ (AS\text{-}seq\ s2\ (AS\text{-}while\ s1\ s2))\ (AS\text{-}val\ (V\text{-}lit\ L\text{-}unit))) \Leftarrow \tau$
using *check-while reduce-whileI* **by** *auto*
thus $?case$ **using** *config-typeI* ***** **by** (*meson subset-refl*)
next

```

case (reduce-caseI dc x' s' css  $\Phi$   $\delta$  tyid v)

hence **:  $\Theta; \Phi; \{\|\}; GNil; \Delta \vdash AS\text{-match} (V\text{-cons tyid dc v}) \text{ css} \Leftarrow \tau \wedge \Theta \vdash \delta \sim \Delta \wedge (\forall fd \in \text{set } \Phi. \text{check-fundef } \Theta \Phi fd)$ 
using config-type-elim[OF reduce-caseI(2)] by metis
hence ***:  $\Theta; \Phi; \{\|\}; GNil; \Delta \vdash AS\text{-match} (V\text{-cons tyid dc v}) \text{ css} \Leftarrow \tau$  by auto

let ?vcons = V-cons tyid dc v

obtain dclist tid and z::x where cv:  $\Theta; \{\|\}; GNil \vdash (V\text{-cons tyid dc v}) \Leftarrow (\{z : B\text{-id tid} \mid \text{TRUE}\}) \wedge$ 
 $\Theta; \Phi; \{\|\}; GNil; \Delta; tid; dclist; (V\text{-cons tyid dc v}) \vdash \text{css} \Leftarrow \tau \wedge AF\text{-typedef tid dclist} \in \text{set } \Theta \wedge$ 
 $\Theta; \{\|\}; GNil \vdash V\text{-cons tyid dc v} \Leftarrow \{z : B\text{-id tid} \mid \text{TRUE}\}$ 
using check-s-elim(9)[OF ***] by metis

hence vi:  $\Theta; \{\|\}; GNil \vdash V\text{-cons tyid dc v} \Leftarrow \{z : B\text{-id tid} \mid \text{TRUE}\}$  by auto
obtain tcons where vi2:  $\Theta; \{\|\}; GNil \vdash V\text{-cons tyid dc v} \Rightarrow tcons \wedge \Theta; \{\|\}; GNil \vdash tcons \lesssim \{z : B\text{-id tid} \mid \text{TRUE}\}$ 
using check-v-elim(1)[OF vi] by metis
hence vi1:  $\Theta; \{\|\}; GNil \vdash V\text{-cons tyid dc v} \Rightarrow tcons$  by auto

show ?case proof(rule infer-v-elim(4)[OF vi1],goal-cases)
case (1 dclist2 tc tv z2)
have tyid = tid using  $\tau.\text{eq-iff}$  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  $\Theta; \Phi; \{\|\}; GNil; \Delta \vdash s'[x'::=v]_{sv} \Leftarrow \tau$  proof(rule check-match(3))
show  $\langle \Theta; \Phi; \{\|\}; GNil; \Delta; tyid; dclist; ?vcons \vdash \text{css} \Leftarrow \tau \rangle$  using  $\langle tyid = tid \rangle$  cv by auto
show distinct (map fst dclist) using wfTh-dclist-distinct check-v-wf wfX-wfY cv by metis
show  $\langle ?vcons = V\text{-cons tyid dc v} \rangle$  by auto
show  $\langle \{\|\} = \{\|\} \rangle$  by auto
show  $\langle (dc, tc) \in \text{set dclist} \rangle$  using 1 deq by auto
show  $\langle GNil = GNil \rangle$  by auto
show  $\langle \text{Some } (AS\text{-branch dc x' s'}) = \text{lookup-branch dc css} \rangle$  using reduce-caseI by auto
show  $\langle \Theta; \{\|\}; GNil \vdash v \Leftarrow tc \rangle$  using 1 check-v.intros by auto
qed
thus ?case using config-typeI ** by blast
qed

next
case (reduce-let-fstI  $\Phi$   $\delta$  x v1 v2 s)
thus ?case using preservation-fst-snd order-refl by metis
next
case (reduce-let-sndI  $\Phi$   $\delta$  x v1 v2 s)
thus ?case using preservation-fst-snd order-refl by metis
next
case (reduce-let-concatI  $\Phi$   $\delta$  x v1 v2 s)
hence elim:  $\Theta; \Phi; \{\|\}; GNil; \Delta \vdash AS\text{-let } x (AE\text{-concat } (V\text{-lit } (L\text{-bitvec } v1)) (V\text{-lit } (L\text{-bitvec } v2)))$ 
 $s \Leftarrow \tau \wedge$ 
 $\Theta \vdash \delta \sim \Delta \wedge (\forall fd \in \text{set } \Phi. \text{check-fundef } \Theta \Phi fd)$ 
using config-type-elim by metis

obtain z::x where z: atom z  $\nmid (AE\text{-concat } (V\text{-lit } (L\text{-bitvec } v1)) (V\text{-lit } (L\text{-bitvec } v2))), GNil, CE\text{-val}$ 

```

$(V\text{-lit } (L\text{-bitvec } (v1 \text{ @ } v2))))$
using *obtain-fresh* **by** *metis*

have $\Theta ; \{\|\} \vdash_{wf} GNil$ **using** *check-s-wf elim* **by** *auto*

have $\Theta; \Phi; \{\|\}; GNil; \Delta \vdash AS\text{-let } x \text{ (} AE\text{-val } (V\text{-lit } (L\text{-bitvec } (v1 \text{ @ } v2)))) s \Leftarrow \tau$ **proof**(*rule subtype-let*)
show $\langle \Theta; \Phi; \{\|\}; GNil; \Delta \vdash AS\text{-let } x \text{ (} AE\text{-concat } (V\text{-lit } (L\text{-bitvec } v1)) \text{ (} V\text{-lit } (L\text{-bitvec } v2))) s \Leftarrow \tau \rangle$ **using** *elim* **by** *auto*
show $\langle \Theta; \Phi; \{\|\}; GNil; \Delta \vdash (AE\text{-concat } (V\text{-lit } (L\text{-bitvec } v1)) \text{ (} V\text{-lit } (L\text{-bitvec } v2))) \Rightarrow \{z : B\text{-bitvec} \mid CE\text{-val } (V\text{-var } z) == (CE\text{-concat } ([V\text{-lit } (L\text{-bitvec } v1)]^{ce}) ([V\text{-lit } (L\text{-bitvec } v2)]^{ce}))\} \rangle$
(is $\Theta; \Phi; \{\|\}; GNil; \Delta \vdash ?e1 \Rightarrow ?t1$ **)**
proof
show $\langle \Theta ; \{\|\} ; GNil \vdash_{wf} \Delta \rangle$ **using** *check-s-wf elim* **by** *auto*
show $\langle \Theta \vdash_{wf} \Phi \rangle$ **using** *check-s-wf elim* **by** *auto*
show $\langle \Theta ; \{\|\} ; GNil \vdash V\text{-lit } (L\text{-bitvec } v1) \Rightarrow \{z : B\text{-bitvec} \mid CE\text{-val } (V\text{-var } z) == CE\text{-val } (V\text{-lit } (L\text{-bitvec } v1))\} \rangle$
using *infer-v-litI infer-l.intros* $\langle \Theta ; \{\|\} \vdash_{wf} GNil \rangle$ *fresh-GNil* **by** *auto*
show $\langle \Theta ; \{\|\} ; GNil \vdash V\text{-lit } (L\text{-bitvec } v2) \Rightarrow \{z : B\text{-bitvec} \mid CE\text{-val } (V\text{-var } z) == CE\text{-val } (V\text{-lit } (L\text{-bitvec } v2))\} \rangle$
using *infer-v-litI infer-l.intros* $\langle \Theta ; \{\|\} \vdash_{wf} GNil \rangle$ *fresh-GNil* **by** *auto*
show $\langle atom \ z \ \# \ AE\text{-concat } (V\text{-lit } (L\text{-bitvec } v1)) \text{ (} V\text{-lit } (L\text{-bitvec } v2)) \rangle$ **using** *z fresh-Pair* **by** *metis*
show $\langle atom \ z \ \# \ GNil \rangle$ **using** *z fresh-Pair* **by** *auto*
qed
show $\langle \Theta; \Phi; \{\|\}; GNil; \Delta \vdash AE\text{-val } (V\text{-lit } (L\text{-bitvec } (v1 \text{ @ } v2))) \Rightarrow \{z : B\text{-bitvec} \mid CE\text{-val } (V\text{-var } z) == CE\text{-val } (V\text{-lit } (L\text{-bitvec } (v1 \text{ @ } v2)))\} \rangle$
(is $\Theta; \Phi; \{\|\}; GNil; \Delta \vdash ?e2 \Rightarrow ?t2$ **)**
using *infer-e-valI infer-v-litI infer-l.intros* $\langle \Theta ; \{\|\} \vdash_{wf} GNil \rangle$ *fresh-GNil* *check-s-wf elim* **by** *metis*
show $\langle \Theta ; \{\|\} ; GNil \vdash ?t2 \lesssim ?t1 \rangle$ **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* $\Phi \delta \ x \ v \ s$)
hence *elim*: $\Theta; \Phi; \{\|\}; GNil; \Delta \vdash AS\text{-let } x \text{ (} AE\text{-len } (V\text{-lit } (L\text{-bitvec } v))) s \Leftarrow \tau \wedge \Theta \vdash \delta \sim \Delta \wedge (\forall fd \in set \ \Phi. \text{check-fundef } \Theta \ \Phi \ fd)$
using *check-s-elims config-type-elims* **by** *metis*

then obtain *t* **where** $t: \Theta; \Phi; \{\|\}; GNil; \Delta \vdash AE\text{-len } (V\text{-lit } (L\text{-bitvec } v)) \Rightarrow t$ **using** *check-s-elims* **by** *meson*

moreover then obtain $z::x$ **where** $t = \{z : B\text{-int} \mid CE\text{-val } (V\text{-var } z) == CE\text{-len } ([V\text{-lit } (L\text{-bitvec } v)]^{ce})\}$ **using** *infer-e-elims* **by** *meson*

moreover obtain $z'::x$ **where** $atom \ z' \ \# \ v$ **using** *obtain-fresh* **by** *metis*
moreover have $\Theta; \Phi; \{\|\}; GNil; \Delta \vdash AE\text{-val } (V\text{-lit } (L\text{-num } (int \ (length \ v)))) \Rightarrow \{z' : B\text{-int} \mid CE\text{-val } (V\text{-var } z') == CE\text{-val } (V\text{-lit } (L\text{-num } (int \ (length \ v))))\}$
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 $\Theta ; \{\|\} ; GNil \vdash \{z' : B\text{-int} \mid CE\text{-val } (V\text{-var } z') == CE\text{-val } (V\text{-lit } (L\text{-num } (int$


```

(length v))))  $\mathbb{I} \lesssim$ 
 $\mathbb{I} z : B\text{-int} \mid CE\text{-val} (V\text{-var } z) == CE\text{-len} [(V\text{-lit} (L\text{-bitvec } v))]^{ce} \mathbb{I}$  using
subtyping-len check-s-wf elim by auto

ultimately have  $\Theta; \Phi; \{\|\}; GNil; \Delta \vdash AS\text{-let } x (AE\text{-val} (V\text{-lit} (L\text{-num} (int (length v)))) s \Leftarrow \tau$ 
using subtyping-let by (meson elim)
thus ?case using config-typeI elim by (meson order-refl)
next
case (reduce-let-splitI n v v1 v2  $\Phi \delta x s$ )
hence elim:  $\Theta; \Phi; \{\|\}; GNil; \Delta \vdash AS\text{-let } x (AE\text{-split} (V\text{-lit} (L\text{-bitvec } v)) (V\text{-lit} (L\text{-num } n))) s \Leftarrow \tau$ 
 $\wedge$ 
 $\Theta \vdash \delta \sim \Delta \wedge (\forall fd \in set \Phi. check\text{-fundef } \Theta \Phi fd)$ 
using config-type-elims by metis

obtain  $z::x$  where  $z: atom \ z \# (AE\text{-split} (V\text{-lit} (L\text{-bitvec } v)) (V\text{-lit} (L\text{-num } n)), GNil, CE\text{-val} (V\text{-lit} (L\text{-bitvec} (v1 @ v2))),$ 
 $([L\text{-bitvec } v1]^v, [L\text{-bitvec } v2]^v), \Theta, \{\|\}::bv \ fset)$ 
using obtain-fresh by metis

have  $*:\Theta; \{\|\} \vdash_{wf} GNil$  using check-s-wf elim by auto

have  $\Theta; \Phi; \{\|\}; GNil; \Delta \vdash AS\text{-let } x (AE\text{-val} (V\text{-pair} (V\text{-lit} (L\text{-bitvec } v1)) (V\text{-lit} (L\text{-bitvec } v2)))) s$ 
 $\Leftarrow \tau$  proof(rule subtyping-let)

show  $\langle \Theta; \Phi; \{\|\}; GNil; \Delta \vdash AS\text{-let } x (AE\text{-split} (V\text{-lit} (L\text{-bitvec } v)) (V\text{-lit} (L\text{-num } n))) s \Leftarrow \tau \rangle$ 
using elim by auto
show  $\langle \Theta; \Phi; \{\|\}; GNil; \Delta \vdash (AE\text{-split} (V\text{-lit} (L\text{-bitvec } v)) (V\text{-lit} (L\text{-num } n))) \Rightarrow \mathbb{I} z : B\text{-pair}$ 
 $B\text{-bitvec } B\text{-bitvec}$ 
 $\mid ((CE\text{-val} (V\text{-lit} (L\text{-bitvec } v))) == (CE\text{-concat} (CE\text{-fst} (CE\text{-val} (V\text{-var } z))) (CE\text{-snd}$ 
 $(CE\text{-val} (V\text{-var } z))))$ 
 $AND (((CE\text{-len} (CE\text{-fst} (CE\text{-val} (V\text{-var } z)))) == (CE\text{-val} (V\text{-lit} (L\text{-num } n)))) \mathbb{I} \rangle$ 
(is  $\Theta; \Phi; \{\|\}; GNil; \Delta \vdash ?e1 \Rightarrow ?t1$ )
proof
show  $\langle \Theta; \{\|\}; GNil \vdash_{wf} \Delta \rangle$  using check-s-wf elim by auto
show  $\langle \Theta \vdash_{wf} \Phi \rangle$  using check-s-wf elim by auto
show  $\langle \Theta; \{\|\}; GNil \vdash V\text{-lit} (L\text{-bitvec } v) \Rightarrow \mathbb{I} z : B\text{-bitvec} \mid CE\text{-val} (V\text{-var } z) == CE\text{-val} (V\text{-lit}$ 
 $(L\text{-bitvec } v)) \mathbb{I} \rangle$ 
using infer-v-litI infer-l.intros  $\langle \Theta; \{\|\} \vdash_{wf} GNil \rangle$  fresh-GNil by auto
show  $\Theta; \{\|\}; GNil \vdash ([L\text{-num}$ 
 $n]^v) \Leftarrow \mathbb{I} z : B\text{-int} \mid ((leq [ [L\text{-num } 0]^v ]^{ce} [ [z]^v ]^{ce} ]^{ce}) == ([ [L\text{-true}]^v$ 
 $]^{ce})) AND [leq [ [z]^v ]^{ce} [ [L\text{-bitvec}$ 
 $v]^v ]^{ce} ]^{ce} == [ [L\text{-true}]^v ]^{ce} \mathbb{I}$  using split-n reduce-let-splitI check-v-num-leq
* wfX-wfY by metis
show  $\langle atom \ z \# AE\text{-split} [L\text{-bitvec } v]^v [L\text{-num } n]^v \rangle$  using z fresh-Pair by auto
show  $\langle atom \ z \# GNil \rangle$  using z fresh-Pair by auto
show  $\langle atom \ z \# AE\text{-split} [L\text{-bitvec } v]^v [L\text{-num } n]^v \rangle$  using z fresh-Pair by auto
show  $\langle atom \ z \# GNil \rangle$  using z fresh-Pair by auto
show  $\langle atom \ z \# AE\text{-split} [L\text{-bitvec } v]^v [L\text{-num } n]^v \rangle$  using z fresh-Pair by auto
show  $\langle atom \ z \# GNil \rangle$  using z fresh-Pair by auto
qed

show  $\langle \Theta; \Phi; \{\|\}; GNil; \Delta \vdash AE\text{-val} (V\text{-pair} (V\text{-lit} (L\text{-bitvec } v1)) (V\text{-lit} (L\text{-bitvec } v2)))) \Rightarrow \mathbb{I} z :$ 

```

```

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  $\Theta$ ;  $\Phi$ ;  $\{\|\}$ ;  $GNil$ ;  $\Delta \vdash ?e2 \Rightarrow ?t2$ )
  apply(rule infer-e-valI)
  using check-s-wf elim apply metis
  using check-s-wf elim apply metis
  apply(rule infer-v-pair2I)
  using z fresh-prodN apply metis
  using z fresh-GNil fresh-prodN apply metis
  using infer-v-litI infer-l.intros  $\langle \Theta ; \{\|\} \vdash_{wf} GNil \rangle$  b-of.simps apply blast+
  using b-of.simps apply simp+
  done
  show  $\langle \Theta ; \{\|\} ; GNil \vdash ?t2 \lesssim ?t1 \rangle$  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  $\Phi \delta c v$ )

hence elim:  $\Theta$ ;  $\Phi$ ;  $\{\|\}$ ;  $GNil$ ;  $\Delta \vdash AS\text{-assert } c [v]^s \Leftarrow \tau \wedge$ 
 $\Theta \vdash \delta \sim \Delta \wedge (\forall fd \in set \Phi. check\text{-fundef } \Theta \Phi fd)$ 
  using config-type-elim reduce-assert1I by metis
hence *: $\Theta$ ;  $\Phi$ ;  $\{\|\}$ ;  $GNil$ ;  $\Delta \vdash AS\text{-assert } c [v]^s \Leftarrow \tau$  by auto

have  $\Theta$ ;  $\Phi$ ;  $\{\|\}$ ;  $GNil$ ;  $\Delta \vdash [v]^s \Leftarrow \tau$  using check-assert-s * by metis
thus ?case using elim config-typeI by blast
next
case (reduce-assert2I  $\Phi \delta s \delta' s' c$ )

hence elim:  $\Theta$ ;  $\Phi$ ;  $\{\|\}$ ;  $GNil$ ;  $\Delta \vdash AS\text{-assert } c s \Leftarrow \tau \wedge$ 
 $\Theta \vdash \delta \sim \Delta \wedge (\forall fd \in set \Phi. check\text{-fundef } \Theta \Phi fd)$ 
  using config-type-elim by metis
hence *: $\Theta$ ;  $\Phi$ ;  $\{\|\}$ ;  $GNil$ ;  $\Delta \vdash AS\text{-assert } c s \Leftarrow \tau$  by auto

have cv:  $\Theta$ ;  $\Phi$ ;  $\{\|\}$ ;  $GNil$ ;  $\Delta \vdash s \Leftarrow \tau \wedge \Theta ; \{\|\} ; GNil \models c$  using check-assert-s * by metis

hence  $\Theta$ ;  $\Phi$ ;  $\Delta \vdash \langle \delta, s \rangle \Leftarrow \tau$  using elim config-typeI by simp
then obtain  $\Delta'$  where  $D: \Theta$ ;  $\Phi$ ;  $\Delta' \vdash \langle \delta', s' \rangle \Leftarrow \tau \wedge \Delta \sqsubseteq \Delta'$  using reduce-assert2I by metis
hence **: $\Theta$ ;  $\Phi$ ;  $\{\|\}$ ;  $GNil$ ;  $\Delta' \vdash s' \Leftarrow \tau \wedge \Theta \vdash \delta' \sim \Delta'$  using config-type-elim by metis

obtain  $x::x$  where  $x:atom x \# (\Theta, \Phi, (\{\|\}::bv fset), GNil, \Delta', c, \tau, s')$  using obtain-fresh by metis

have *: $\Theta$ ;  $\Phi$ ;  $\{\|\}$ ;  $GNil$ ;  $\Delta' \vdash AS\text{-assert } c s' \Leftarrow \tau$  proof
  show  $atom x \# (\Theta, \Phi, \{\|\}, GNil, \Delta', c, \tau, s')$  using x by auto
  have  $\Theta ; \{\|\} ; GNil \vdash_{wf} c$  using * check-s-wf by auto
  hence wfg: $\Theta ; \{\|\} \vdash_{wf} (x, B\text{-bool}, c) \#_{\Gamma} GNil$  using wfC-wfG wfB-boolI check-s-wf * fresh-GNil
by auto
  moreover have cs:  $\Theta$ ;  $\Phi$ ;  $\{\|\}$ ;  $GNil$ ;  $\Delta' \vdash s' \Leftarrow \tau$  using ** by auto
  ultimately show  $\Theta ; \Phi ; \{\|\} ; (x, B\text{-bool}, c) \#_{\Gamma} GNil ; \Delta' \vdash s' \Leftarrow \tau$  using check-s-g-weakening(1)[OF
cs - wfg] toSet.simps by simp
  show  $\Theta ; \{\|\} ; GNil \models c$  using cv by auto
  show  $\Theta ; \{\|\} ; GNil \vdash_{wf} \Delta'$  using check-s-wf ** by auto

```

```

qed

thus ?case using elim config-typeI D ** by metis
qed

lemma preservation-many:
  assumes  $\Phi \vdash \langle \delta, s \rangle \longrightarrow^* \langle \delta', s' \rangle$ 
  shows  $\Theta; \Phi; \Delta \vdash \langle \delta, s \rangle \Leftarrow \tau \implies \exists \Delta'. \Theta; \Phi; \Delta' \vdash \langle \delta', s' \rangle \Leftarrow \tau \wedge \Delta \sqsubseteq \Delta'$ 
  using assms proof(induct arbitrary:  $\Delta$  rule: reduce-stmt-many.induct)
  case (reduce-stmt-many-oneI  $\Phi \delta s \delta' s'$ )
  then show ?case using preservation by simp
next
  case (reduce-stmt-many-manyI  $\Phi \delta s \delta' s' \delta'' s''$ )
  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  $\Theta; \Phi; \{\|\}; \Gamma; \Delta \vdash LET\ x = (AE\text{-}op\ opp\ v1\ v2)\ IN\ s \Leftarrow \tau$  and  $supp\ (LET\ x = (AE\text{-}op\ opp\ v1\ v2)\ IN\ s) \subseteq atomfst'setD\ \Delta$ 
  shows  $\exists z\ b\ c. \Theta; \Phi; \{\|\}; \Gamma; \Delta \vdash (AE\text{-}op\ opp\ v1\ v2) \Rightarrow \llbracket z:b|c \rrbracket$ 
proof -
  have  $xx: \Theta; \Phi; \{\|\}; \Gamma; \Delta \vdash LET\ x = (AE\text{-}op\ opp\ v1\ v2)\ IN\ s \Leftarrow \tau$  using assms by simp
  then show ?thesis using check-s-elim(2)[OF xx] by meson
qed

```

```

lemma infer-pair:
  assumes  $\Theta; B; \Gamma \vdash v \Rightarrow \llbracket z : B\text{-}pair\ b1\ b2 \mid c \rrbracket$  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.suppl* 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-elimsl*(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-elimsl*(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 $\exists \delta' s'. \Phi \vdash \langle \delta, LET\ x = e\ IN\ s \rangle \longrightarrow \langle \delta', s' \rangle$
 proof –
 obtain $z\ b\ c$ where $*: \Theta; \Phi; \{\|\}; \Gamma; \Delta \vdash e \Rightarrow \{\!\!| z : b \mid c |\!\!\}$ using *check-s-elimsl*(2)[*OF assms*(1)]
 by *metis*
 have $**:$ $supp\ e \subseteq atom'fst'setD\ \Delta$ using *assms s-branch-s-branch-list.suppl* by *auto*
 from $**$ *assms* show *?thesis* **proof**(*nominal-induct* $\{\!\!| z : b \mid c |\!\!\}$ *rule: infer-e.strong-induct*)
 case (*infer-e-valI* $\Theta\ \mathcal{B}\ \Gamma\ \Delta\ \Phi\ v$)
 then show *?case* using *reduce-stmt-elimsl reduce-let-valI* by *metis*
 next
 case (*infer-e-plusI* $\Theta\ \mathcal{B}\ \Gamma\ \Delta\ \Phi\ v1\ z1\ c1\ v2\ z2\ c2\ z3$)
 hence *vf:* $supp\ v1 = \{\} \wedge supp\ v2 = \{\}$ by *force*
 then obtain $n1$ and $n2$ where $*: v1 = V\text{-}lit\ (L\text{-}num\ n1) \wedge v2 = (V\text{-}lit\ (L\text{-}num\ n2))$ using
infer-int infer-e-plusI by *metis*
 then show *?case* using *reduce-let-plusI ** by *metis*
 next
 case (*infer-e-leqI* $\Theta\ \mathcal{B}\ \Gamma\ \Delta\ \Phi\ v1\ z1\ c1\ v2\ z2\ c2\ z3$)
 hence *vf:* $supp\ v1 = \{\} \wedge supp\ v2 = \{\}$ by *force*
 then obtain $n1$ and $n2$ where $*: v1 = V\text{-}lit\ (L\text{-}num\ n1) \wedge v2 = (V\text{-}lit\ (L\text{-}num\ n2))$ using
infer-int infer-e-leqI by *metis*
 then show *?case* using *reduce-let-leqI ** by *metis*
 next
 case (*infer-e-appI* $\Theta\ \mathcal{B}\ \Gamma\ \Delta\ \Phi\ f\ x\ b\ c\ \tau'\ s'\ v$)
 then show *?case* using *reduce-let-appI* by *metis*
 next
 case (*infer-e-appPI* $\Theta\ \mathcal{B}\ \Gamma\ \Delta\ \Phi\ b'\ f\ bv\ x\ b\ c\ \tau'\ s'\ v$)
 then show *?case* using *reduce-let-appPI* by *metis*
 next
 case (*infer-e-fstI* $\Theta\ \mathcal{B}\ \Gamma\ \Delta\ \Phi\ v\ z'\ b2\ c\ z$)
 hence $supp\ v = \{\}$ by *force*
 then obtain $v1$ and $v2$ where $v = V\text{-}pair\ v1\ v2$ using *infer-e-fstI infer-pair* by *metis*
 then show *?case* using *reduce-let-fstI ** by *metis*
 next
 case (*infer-e-sndI* $\Theta\ \mathcal{B}\ \Gamma\ \Delta\ \Phi\ v\ z'\ b1\ c\ z$)
 hence $supp\ v = \{\}$ by *force*
 then obtain $v1$ and $v2$ where $v = V\text{-}pair\ v1\ v2$ using *infer-e-sndI infer-pair* by *metis*

```

    then show ?case using reduce-let-sndI * by metis
next
  case (infer-e-lenI  $\Theta \mathcal{B} \Gamma \Delta \Phi v z' c za$ )
  hence  $\text{supp } v = \{\}$  by force
  then obtain bvec where  $v = V\text{-lit } (L\text{-bitvec } bvec)$  using infer-e-lenI infer-bitvec by metis
  then show ?case using reduce-let-lenI * by metis
next
  case (infer-e-mvarI  $\Theta \mathcal{B} \Gamma \Phi \Delta u$ )
  hence  $(u, \llbracket z : b \mid c \rrbracket) \in \text{setD } \Delta$  using infer-e-elim(10) by meson
  then obtain  $v$  where  $(u, v) \in \text{set } \delta$  using infer-e-mvarI delta-sim-delta-lookup by meson
  then show ?case using reduce-let-mvar by metis
next
  case (infer-e-concatI  $\Theta \mathcal{B} \Gamma \Delta \Phi v1 z1 c1 v2 z2 c2 z3$ )
  hence  $\text{vf: } \text{supp } v1 = \{\} \wedge \text{supp } v2 = \{\}$  by force
  then obtain  $n1$  and  $n2$  where  $*: v1 = V\text{-lit } (L\text{-bitvec } n1) \wedge v2 = (V\text{-lit } (L\text{-bitvec } n2))$  using
infer-bitvec infer-e-concatI by metis
  then show ?case using reduce-let-concatI * by metis
next
  case (infer-e-splitI  $\Theta \mathcal{B} \Gamma \Delta \Phi v1 z1 c1 v2 z2 z3$ )
  hence  $\text{vf: } \text{supp } v1 = \{\} \wedge \text{supp } v2 = \{\}$  by force
  then obtain  $n1$  and  $n2$  where  $*: v1 = V\text{-lit } (L\text{-bitvec } n1) \wedge v2 = (V\text{-lit } (L\text{-num } n2))$  using
infer-bitvec infer-e-splitI check-int by metis

  have  $0 \leq n2 \wedge n2 \leq \text{int } (\text{length } n1)$  using check-v-range[OF - *] infer-e-splitI by simp
  then obtain  $bv1$  and  $bv2$  where  $\text{split } n2 n1 (bv1, bv2)$  using obtain-split by metis
  then show ?case using reduce-let-splitI * by metis
qed
qed

```

lemma *check-css-lookup-branch-exist:*

```

  fixes  $s::s$  and  $cs::\text{branch-s}$  and  $css::\text{branch-list}$  and  $v::v$ 
  shows
     $\Theta; \Phi; B; G; \Delta \vdash s \Leftarrow \tau \implies \text{True}$  and
     $\text{check-branch-s } \Theta \Phi \{\llbracket \rrbracket\} GNil \Delta tid dc \text{const } v cs \tau \implies \text{True}$  and
     $\Theta; \Phi; \mathcal{B}; \Gamma; \Delta; tid; dclist; v \vdash css \Leftarrow \tau \implies (dc, t) \in \text{set } dclist \implies$ 
 $\exists x' s'. \text{Some } (AS\text{-branch } dc x' s') = \text{lookup-branch } dc css$ 
  proof(nominal-induct  $\tau$  and  $\tau$  and  $\tau$  rule: check-s-check-branch-s-check-branch-list.strong-induct)
    case (check-branch-list-consI  $\Theta \Phi \mathcal{B} \Gamma \Delta tid \text{cons const } v cs \tau dclist css$ )
    then show ?case using lookup-branch.simps check-branch-list-finalI by force
  next
    case (check-branch-list-finalI  $\Theta \Phi \mathcal{B} \Gamma \Delta tid \text{cons const } v cs \tau$ )
    then show ?case using lookup-branch.simps check-branch-list-finalI by force
  qed(auto+)

```

lemma *progress-aux:*

```

  shows  $\Theta; \Phi; \mathcal{B}; \Gamma; \Delta \vdash s \Leftarrow \tau \implies \mathcal{B} = \{\llbracket \rrbracket\} \implies \text{sble } \Theta \Gamma \implies \text{supp } s \subseteq \text{atom 'fst 'setD } \Delta \implies$ 
 $\Theta \vdash \delta \sim \Delta \implies$ 
 $(\exists v. s = [v]^s) \vee (\exists \delta' s'. \Phi \vdash \langle \delta, s \rangle \longrightarrow \langle \delta', s' \rangle)$  and
 $\Theta; \Phi; \{\llbracket \rrbracket\}; \Gamma; \Delta; tid; dc; \text{const}; v2 \vdash cs \Leftarrow \tau \implies \text{supp } cs = \{\} \implies \text{True}$ 

```

$\Theta; \Phi; \{\|\}; \Gamma; \Delta; tid; dclist; v2 \vdash css \Leftarrow \tau \implies supp\ css = \{\} \implies True$
proof(*induct rule: check-s-check-branch-s-check-branch-list.inducts*)
case (*check-valI* $\Delta \Theta \Gamma v \tau' \tau$)
 then show ?*case* **by** *auto*
next
 case (*check-letI* $x \Theta \Phi \mathcal{B} \Gamma \Delta e \tau z s b c$)
 hence $\Theta; \Phi; \{\|\}; \Gamma; \Delta \vdash AS\text{-}let\ x\ e\ s \Leftarrow \tau$ **using** *Typing.check-letI* **by** *meson*
 then show ?*case* **using** *progress-let check-letI* **by** *metis*
next
 case (*check-branch-s-branchI* $\Theta \Phi \mathcal{B} \Gamma \Delta \tau\ const\ x\ \Phi\ tid\ cons\ v\ s$)
 then show ?*case* **by** *auto*
next
 case (*check-branch-list-consI* $\Theta \Phi \mathcal{B} \Gamma \Delta\ tid\ dclist\ v\ cs\ \tau\ css$)
 then show ?*case* **by** *auto*
next
 case (*check-branch-list-finalI* $\Theta \Phi \mathcal{B} \Gamma \Delta\ tid\ dclist\ v\ cs\ \tau$)
 then show ?*case* **by** *auto*
next
 case (*check-ifI* $z \Theta \Phi \mathcal{B} \Gamma \Delta\ v\ s1\ s2\ \tau$)
 have $supp\ v = \{\}$ **using** *check-ifI s-branch-s-branch-list.suppl* **by** *auto*
 hence $v = V\text{-}lit\ L\text{-}true \vee v = V\text{-}lit\ L\text{-}false$ **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* $x \Theta \Phi \mathcal{B} G \Delta\ t\ s1\ \tau\ s2$)
 then consider $(\exists v. s1 = AS\text{-}val\ v) \mid (\exists \delta' a. \Phi \vdash \langle \delta, s1 \rangle \longrightarrow \langle \delta', a \rangle)$ **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* $u \Theta \Phi \mathcal{B} \Gamma \Delta\ \tau' v \tau s$)

 obtain $uu::u$ **where** $uf: atom\ uu \# (u, \delta, s)$ **using** *obtain-fresh* **by** *blast*
 obtain sa **where** $(uu \leftrightarrow u) \cdot s = sa$ **by** *presburger*
 moreover have $atom\ uu \# s$ **using** *uf fresh-prod3* **by** *auto*
 ultimately have $AS\text{-}var\ uu\ \tau' v\ sa = AS\text{-}var\ u\ \tau' v\ s$ **using** *s-branch-s-branch-list.eq-iff(7) Abs1-eq-iff(3)*[*of uu sa u s*] **by** *auto*

 moreover have $atom\ uu \# \delta$ **using** *uf fresh-prod3* **by** *auto*
 ultimately have $\Phi \vdash \langle \delta, AS\text{-}var\ u\ \tau' v\ s \rangle \longrightarrow \langle (uu, v) \# \delta, sa \rangle$
 using *reduce-varI uf* **by** *metis*
 then show ?*case* **by** *auto*
next
 case (*check-assignI* $\Delta\ u\ \tau\ P\ G\ v\ z\ \tau'$)
 then show ?*case* **using** *reduce-assignI* **by** *blast*
next
 case (*check-whileI* $\Theta \Phi \mathcal{B} \Gamma \Delta\ s1\ z\ s2\ \tau'$)
 obtain $x::x$ **where** $atom\ x \# (s1, s2)$ **using** *obtain-fresh* **by** *metis*
 moreover obtain $z::x$ **where** $atom\ z \# x$ **using** *obtain-fresh* **by** *metis*

ultimately show *?case* using *reduce-whileI* by *fast*

next

case (*check-seqI* $P \Phi \mathcal{B} G \Delta s1 z s2 \tau$)

thus *?case* **proof**(*cases* $\exists v. s1 = AS\text{-}val v$)

case *True*

then obtain v where $v: s1 = AS\text{-}val v$ by *blast*

hence $supp v = \{\}$ using *check-seqI* by *auto*

have $\exists z1 c1. P; \mathcal{B}; G \vdash v \Rightarrow (\llbracket z1 : B\text{-}unit \mid c1 \rrbracket)$ **proof** –

obtain t where $t: P; \mathcal{B}; G \vdash v \Rightarrow t \wedge P; \mathcal{B}; G \vdash t \lesssim (\llbracket z : B\text{-}unit \mid TRUE \rrbracket)$

using v *check-seqI*(1) *check-s-elim*s(1) by *blast*

obtain $z1$ and $b1$ and $c1$ where $teq: t = (\llbracket z1 : b1 \mid c1 \rrbracket)$ using *obtain-fresh-z* by *meson*

hence $b1 = B\text{-}unit$ using *subtype-eq-base* t by *meson*

thus *?thesis* using t *teq* by *fast*

qed

then obtain $z1$ and $c1$ where $P; \mathcal{B}; G \vdash v \Rightarrow (\llbracket z1 : B\text{-}unit \mid c1 \rrbracket)$ by *auto*

hence $v = V\text{-}lit L\text{-}unit$ using *infer-v-unit-form* $\langle supp v = \{\} \rangle$ by *simp*

hence $s1 = AS\text{-}val (V\text{-}lit L\text{-}unit)$ 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.sup*p(9))

qed

next

case (*check-caseI* $\Theta \Phi \mathcal{B} \Gamma \Delta tid dclist v cs \tau z$)

hence $supp v = \{\}$ by *auto*

then obtain v' and dc and $t::\tau$ where $v: v = V\text{-}cons tid dc v' \wedge (dc, t) \in set dclist$

using *check-v-tid-form* *check-caseI* by *metis*

obtain z and b and c where $teq: t = (\llbracket z : b \mid c \rrbracket)$ using *obtain-fresh-z* by *meson*

moreover then obtain $x' s'$ where $Some (AS\text{-}branch dc x' s') = lookup\text{-}branch dc cs$ 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* $x \Theta \Phi \mathcal{B} \Gamma \Delta c \tau s$)

hence $sps: supp s \subseteq atom \text{ 'fst ' setD } \Delta$ by *auto*

have $atom x \nVdash c$ using *check-assertI* by *auto*

have $atom x \nVdash \Gamma$ using *check-assertI* *check-s-wf wfG-elim*s by *metis*

have $sble \Theta ((x, B\text{-}bool, c) \#_{\Gamma} \Gamma)$ **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* $i \langle atom x \nVdash c \rangle$ by *auto*

moreover have $i \models \Gamma$ using *is-satis-g-weakening-x* $i' i$ *check-assertI* $\langle atom x \nVdash \Gamma \rangle$ by *metis*

ultimately show *?thesis* using *is-satis-g.simps* i by *auto*

qed

moreover have $\Theta; ((x, B\text{-}bool, c) \#_{\Gamma} \Gamma) \vdash i$ **proof**(*rule* *wfI-cons*)

show $\langle i' \models \Gamma \rangle$ using i' by *auto*

```

  show  $\langle \Theta ; \Gamma \vdash i' \rangle$  using  $i'$  by auto
  show  $\langle i = i'(x \mapsto SBool\ True) \rangle$  using  $i$  by auto
  show  $\langle \Theta \vdash SBool\ True: B\text{-}bool \rangle$  using  $wfRCV\text{-}BBoolI$  by auto
  show  $\langle atom\ x \# \Gamma \rangle$  using  $check\text{-}assertI\ check\text{-}s\text{-}wf\ wfG\text{-}elims$  by auto
qed
ultimately show  $?thesis$  using  $sble\text{-}def$  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  $check\text{-}assertI\ sps$  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  $reduce\text{-}assert1I$  by metis
next
  case 2
  then show  $?thesis$  using  $reduce\text{-}assert2I$  by metis
qed
thus  $?case$  by auto
qed

```

lemma *progress*:

```

  assumes  $\Theta; \Phi; \Delta \vdash \langle \delta, s \rangle \Leftarrow \tau$ 
  shows  $(\exists v. s = [v]^s) \vee (\exists \delta' s'. \Phi \vdash \langle \delta, s \rangle \longrightarrow \langle \delta', s' \rangle)$ 
proof –
  have  $\Theta; \Phi; \{\mid\}; GNil; \Delta \vdash s \Leftarrow \tau$  and  $\Theta \vdash \delta \sim \Delta$ 
  using  $config\text{-}type\text{-}elims[OF\ assms(1)]$  by auto+
  moreover hence  $supp\ s \subseteq atom\ 'fst\ 'setD\ \Delta$  using  $check\text{-}s\text{-}wf\ wfS\text{-}supp$  by fastforce
  moreover have  $sble\ \Theta\ GNil$  using  $sble\text{-}def\ wfI\text{-}def\ is\text{-}satis\text{-}g.\text{simps}$  by simp
  ultimately show  $?thesis$  using  $progress\text{-}aux$  by blast
qed

```

16.4 Safety

lemma *safety-stmt*:

```

  assumes  $\Phi \vdash \langle \delta, s \rangle \longrightarrow^* \langle \delta', s' \rangle$  and  $\Theta; \Phi; \Delta \vdash \langle \delta, s \rangle \Leftarrow \tau$ 
  shows  $(\exists v. s' = [v]^s) \vee (\exists \delta'' s''. \Phi \vdash \langle \delta', s' \rangle \longrightarrow \langle \delta'', s'' \rangle)$ 
  using  $preservation\text{-}many\ progress\ assms$  by meson

```

lemma *safety*:

```

  assumes  $\vdash \langle PROG\ \Theta\ \Phi\ \mathcal{G}\ s \rangle \Leftarrow \tau$  and  $\Phi \vdash \langle \delta\text{-}of\ \mathcal{G}, s \rangle \longrightarrow^* \langle \delta', s' \rangle$ 
  shows  $(\exists v. s' = [v]^s) \vee (\exists \delta'' s''. \Phi \vdash \langle \delta', s' \rangle \longrightarrow \langle \delta'', s'' \rangle)$ 
  using  $assms\ config\text{-}type\text{-}prog\text{-}elims\ safety\text{-}stmt$  by metis

```

unused-thms *Eisbach-Tools Nominal2 AList Nominal-Utils RCLogic-*

end

