A Coalgebraic Decision Procedure for WS1S

Dmitriy Traytel

January 19, 2015

Contents

1 Equivalence Framework

abbreviation $\mathfrak{d}s \equiv \textit{fold}\ (\lambda a\ L.\ \mathfrak{d}\ L\ a)$

lemma *in-language-*$\mathfrak{d}s$: *in-language* $(\mathfrak{d}s\ w\ L)\ v \longleftrightarrow$ *in-language* $L\ (w\ @\ v)$
by (*induct w arbitrary*: L) *simp-all*

lemma $\mathfrak{o}$-$\mathfrak{d}s$: $\mathfrak{o}\ (\mathfrak{d}s\ w\ L) \longleftrightarrow$ *in-language* $L\ w$

by (*induct w arbitrary*: *L*) *auto*

lemma *in-language-to-language*[*simp*]: *in-language* (*to-language L*) *w* $\longleftrightarrow$ *w* $\in$ *L*
by (*metis in-language-to-language mem-Collect-eq*)

lemma *rtrancl-fold-product*:
shows $\{((r, s), (f\ a\ r, f\ a\ s)) \mid a\ r\ s.\ a \in A\}$ ^* =
$\{((r, s), (fold\ f\ w\ r, fold\ f\ w\ s)) \mid w\ r\ s.\ w \in lists\ A\}$ (**is** *?L* = *?R*)
proof−
{ **fix** $r\ s\ r'\ s'$
have $((r, s), (r', s'))$: *?L* $\Longrightarrow$ $((r, s), (r', s')) \in$ *?R*
proof(*induction rule*: *converse-rtrancl-induct2*)
case *refl* **show** *?case* **by**(*force intro*!: *fold-simps*(*1*)[*symmetric*])
next
case *step* **thus** *?case* **by**(*force intro*!: *fold-simps*(*2*)[*symmetric*])
qed
} **moreover**
{ **fix** $r\ s\ r'\ s'$
{ **fix** *w* **assume** $w \in$ *lists A*
then have $((r, s), fold\ f\ w\ r, fold\ f\ w\ s) \in$ *?L*
proof(*induction w rule*: *rev-induct*)
case *Nil* **show** *?case* **by** *simp*
next
case *snoc* **thus** *?case* **by** (*force elim*!: *rtrancl-into-rtrancl*)
qed
}
hence $((r, s), (r', s')) \in$ *?R* $\Longrightarrow$ $((r, s), (r', s')) \in$ *?L* **by** *auto*
} **ultimately show** *?thesis* **by** (*auto 10 0*)
qed

lemma *rtrancl-fold-product1*:
shows $\{(r, s).\ \exists\, a \in A.\ s = f\ a\ r\}$ ^* = $\{(r, s).\ \exists\, a \in lists\ A.\ s = fold\ f\ a\ r\}$ (**is** *?L* = *?R*)
proof−
{ **fix** *r s*
have $(r, s) \in$ *?L* $\Longrightarrow$ $(r, s) \in$ *?R*
proof(*induction rule*: *converse-rtrancl-induct*)
case *base* **show** *?case* **by**(*force intro*!: *fold-simps*(*1*)[*symmetric*])
next
case *step* **thus** *?case* **by**(*force intro*!: *fold-simps*(*2*)[*symmetric*])
qed
} **moreover**
{ **fix** *r s*
{ **fix** *w* **assume** $w \in$ *lists A*
then have $(r, fold\ f\ w\ r) \in$ *?L*
proof(*induction w rule*: *rev-induct*)
case *Nil* **show** *?case* **by** *simp*
next
case *snoc* **thus** *?case* **by** (*force elim*!: *rtrancl-into-rtrancl*)

qed
}
hence $(r,\ s) \in\ ?R \Longrightarrow (r,\ s) \in\ ?L$ **by** *auto*
} **ultimately show** *?thesis* **by** (*auto 10 0*)
qed

lemma *lang-eq-ext-Nil-fold-Deriv*:
fixes $K\ L\ A$
assumes $\bigwedge w.\ \textit{in-language}\ K\ w \Longrightarrow w \in \textit{lists}\ A$ $\bigwedge w.\ \textit{in-language}\ L\ w \Longrightarrow w \in \textit{lists}\ A$
defines $\mathfrak{B} \equiv \{(\mathfrak{d}s\ w\ K,\ \mathfrak{d}s\ w\ L) \mid w.\ w \in \textit{lists}\ A\}$
shows $K = L \longleftrightarrow (\forall (K,\ L) \in \mathfrak{B}.\ \mathfrak{o}\ K \longleftrightarrow \mathfrak{o}\ L)$
proof
assume $\forall (K,\ L){\in}\mathfrak{B}.\ \mathfrak{o}\ K = \mathfrak{o}\ L$
then show $K = L$
unfolding $\mathfrak{B}$-*def* **using** *assms*(*1*,*2*)
proof (*coinduction arbitrary*: $K\ L$)
case (*Lang K L*)
then have *CIH*: $\bigwedge K'\ L'.\ \exists w.\ K' = \mathfrak{d}s\ w\ K \wedge L' = \mathfrak{d}s\ w\ L \wedge w \in \textit{lists}\ A \Longrightarrow \mathfrak{o}\ K' = \mathfrak{o}\ L'$ **and**
[*dest*, *simp*]: $\bigwedge w.\ \textit{in-language}\ K\ w \Longrightarrow w \in \textit{lists}\ A$ $\bigwedge w.\ \textit{in-language}\ L\ w \Longrightarrow w \in \textit{lists}\ A$
by *blast+*
show *?case* **unfolding** *ex-simps simp-thms*
proof (*safe del*: *iffI*)
show $\mathfrak{o}\ K = \mathfrak{o}\ L$ **by** (*intro CIH*[*OF exI*[**where** $x = []$]]) *simp*
next
fix $x\ w$ **assume** $\forall x{\in}\textit{set}\ w.\ x \in A$
then show $\mathfrak{o}\ (\mathfrak{d}s\ w\ (\mathfrak{d}\ K\ x)) = \mathfrak{o}\ (\mathfrak{d}s\ w\ (\mathfrak{d}\ L\ x))$
proof (*cases* $x \in A$)
assume $x \notin A$
then show *?thesis* **unfolding** *in-language-*$\mathfrak{d}$*s in-language.simps*[*symmetric*] **by** *fastforce*
qed (*intro CIH*[*OF exI*[**where** $x = x\ \#\ w$]], *auto*)
qed (*auto simp add*: *in-language.simps*[*symmetric*] *simp del*: *in-language.simps*)
qed
qed (*auto simp*: $\mathfrak{B}$-*def*)

1.1 Abstract Deterministic Automaton

locale *DA* =
fixes *alphabet* :: *'a list*
fixes *init* :: $'t \Rightarrow 's$
fixes *delta* :: $'a \Rightarrow 's \Rightarrow 's$
fixes *accept* :: $'s \Rightarrow \textit{bool}$
fixes *wellformed* :: $'s \Rightarrow \textit{bool}$
fixes *wf* :: $'t \Rightarrow \textit{bool}$
fixes *Language* :: $'s \Rightarrow$ *'a language*
fixes *Lang* :: $'t \Rightarrow$ *'a language*

assumes *Language-init*: *Language* (*init* *t*) = *Lang* *t*
assumes *wellformed-init*: *wf* *t* $\Longrightarrow$ *wellformed* (*init* *t*)
assumes *Language-alphabet*: $[\![$*wellformed* *s*; *in-language* (*Language* *s*) *w*$]\!]$ $\Longrightarrow$ *w* $\in$ *lists* (*set* *alphabet*)
assumes *wellformed-delta*: $[\![$*wellformed* *s*; *a* $\in$ *set* *alphabet*$]\!]$ $\Longrightarrow$ *wellformed* (*delta* *a* *s*)
assumes *Language-delta*: $[\![$*wellformed* *s*; *a* $\in$ *set* *alphabet*$]\!]$ $\Longrightarrow$ *Language* (*delta* *a* *s*) = $\mathfrak{d}$ (*Language* *s*) *a*
assumes *accept-Language*: *wellformed* *s* $\Longrightarrow$ *accept* *s* $\longleftrightarrow$ $\mathfrak{o}$ (*Language* *s*)
begin

lemma *wellformed-deltas*: $[\![$*wellformed* *s*; *w* $\in$ *lists* (*set* *alphabet*)$]\!]$ $\Longrightarrow$
  wellformed (*fold* *delta* *w* *s*)
  by (*induction* *w* *arbitrary*: *s*) (*auto* *simp* *add*: *Language-delta* *wellformed-delta*)

lemma *Language-deltas*: $[\![$*wellformed* *s*; *w* $\in$ *lists* (*set* *alphabet*)$]\!]$ $\Longrightarrow$
  Language (*fold* *delta* *w* *s*) = $\mathfrak{d}$*s* *w* (*Language* *s*)
  by (*induction* *w* *arbitrary*: *s*) (*auto* *simp* *add*: *Language-delta* *wellformed-delta*)

abbreviation *closure* :: $'s * 's \Rightarrow (('s * 's)$ *list* $* ('s * 's)$ *set*) *option* **where**
  closure $\equiv$ *rtrancl-while* (λ(*p*, *q*). *accept* *p* = *accept* *q*)
    (λ(*p*, *q*). *map* (λ*a*. (*delta* *a* *p*, *delta* *a* *q*)) *alphabet*)

theorem *closure-sound-complete*:
assumes *wf*: *wf* *r* *wf* *s*
and *result*: *closure* (*init* *r*, *init* *s*) = *Some* (*ws*, *R*) (**is** *closure* (*?r*, *?s*) = -)
shows *ws* = [] $\longleftrightarrow$ *Lang* *r* = *Lang* *s*
proof –
  from *wf* **have** *wellformed*: *wellformed* *?r* *wellformed* *?s* **using** *wellformed-init* **by** *blast+*
  note *Language-alphabets*[*simp*] =
    Language-alphabet[*OF* *wellformed*(*1*)] *Language-alphabet*[*OF* *wellformed*(*2*)]
  note *Language-deltass* = *Language-deltas*[*OF* *wellformed*(*1*)] *Language-deltas*[*OF* *wellformed*(*2*)]

  have *bisim*: (*Language* *?r* = *Language* *?s*) =
    ($\forall$ *a* *b*. ($\exists$ *w*. *a* = $\mathfrak{d}$*s* *w* (*Language* *?r*) $\wedge$ *b* = $\mathfrak{d}$*s* *w* (*Language* *?s*) $\wedge$ *w* $\in$ *lists* (*set* *alphabet*)) $\longrightarrow$
    $\mathfrak{o}$ *a* = $\mathfrak{o}$ *b*)
    by (*subst* *lang-eq-ext-Nil-fold-Deriv*) *auto*

  have *leq*: (*Language* *?r* = *Language* *?s*) =
  ($\forall$ (*r'*, *s'*) $\in$ {((*r*, *s*), (*delta* *a* *r*, *delta* *a* *s*)) | *a* *r* *s*. *a* $\in$ *set* *alphabet*}^* `` {(*?r*, *?s*)}.
    accept *r'* = *accept* *s'*) **using** *Language-deltass*
      accept-Language[*OF* *wellformed-deltas*[*OF* *wellformed*(*1*)]]
      accept-Language[*OF* *wellformed-deltas*[*OF* *wellformed*(*2*)]]
      unfolding *rtrancl-fold-product* *in-lists-conv-set* *bisim*
      by (*auto* *10* *0*)

have $\{(x,y).\ y \in set\ ((\lambda(p,q).\ map\ (\lambda a.\ (delta\ a\ p,\ delta\ a\ q))\ alphabet)\ x)\} =$
$\{((r,\ s),\ (delta\ a\ r,\ delta\ a\ s))\ |\ a\ r\ s.\ a \in set\ alphabet\}$ **by** *auto*
with *rtrancl-while-Some*[*OF result*]
have $(ws = []) = (Language\ ?r = Language\ ?s)$ **by** (*auto simp add*: *leq Ball-def split*: *if-splits*)
then show *?thesis* **unfolding** *Language-init* **.**
qed

1.2 The overall procedure

definition *check-eqv* :: $'t \Rightarrow 't \Rightarrow bool$ **where**
$check\text{-}eqv\ r\ s = (wf\ r \wedge wf\ s \wedge (case\ closure\ (init\ r,\ init\ s)\ of\ Some([],\ \text{-}) \Rightarrow True$
$|\ \text{-} \Rightarrow False))$

lemma *soundness*:
assumes *check-eqv r s* **shows** $Lang\ r = Lang\ s$
proof −
obtain R **where** $wf\ r\ wf\ s\ closure\ (init\ r,\ init\ s) = Some([],\ R)$
using *assms* **by** (*auto simp*: *check-eqv-def Let-def split*: *option.splits list.splits*)
from *closure-sound-complete*[*OF this*] **show** $Lang\ r = Lang\ s$ **by** *simp*
qed

Auxiliary functions:

definition *reachable* :: $'a\ list \Rightarrow 's \Rightarrow 's\ set$ **where**
$reachable\ as\ s = snd\ (the\ (rtrancl\text{-}while\ (\lambda\text{-}.\ True)\ (\lambda s.\ map\ (\lambda a.\ delta\ a\ s)\ as)\ s))$

definition *automaton* :: $'a\ list \Rightarrow 's \Rightarrow (('s * 'a) * 's)\ set$ **where**
$automaton\ as\ s =$
$snd\ (the$
$(let\ start = (([s],\ \{s\}),\ \{\});$
$test = \lambda((ws,\ Z),\ A).\ ws \neq [];$
$step = \lambda((ws,\ Z),\ A).$
$(let\ s = hd\ ws;$
$new\text{-}edges = map\ (\lambda a.\ ((s,\ a),\ delta\ a\ s))\ as;$
$new = remdups\ (filter\ (\lambda ss.\ ss \notin Z)\ (map\ snd\ new\text{-}edges))$
$in\ ((new\ @\ tl\ ws,\ set\ new \cup Z),\ set\ new\text{-}edges \cup A))$
$in\ while\text{-}option\ test\ step\ start))$

definition *match* :: $'s \Rightarrow 'a\ list \Rightarrow bool$ **where**
$match\ s\ w = accept\ (fold\ delta\ w\ s)$

lemma *match-correct*:
assumes $wellformed\ s\ w \in lists\ (set\ alphabet)$
shows $match\ s\ w \longleftrightarrow in\text{-}language\ (Language\ s)\ w$
unfolding *match-def accept-Language*[*OF wellformed-deltas*[*OF assms*]] *Language-deltas*[*OF assms*] **o-D***s* **..**

end

1.3 Abstract Deterministic Finite Automaton

locale *DFA* = *DA* +
assumes *fin*: *wellformed s* $\Longrightarrow$ *finite* {*fold delta w s* | *w*. *w* $\in$ *lists* (*set alphabet*)}
begin

lemma *finite-rtrancl-delta-Image*:
  [[*wellformed r*; *wellformed s*]] $\Longrightarrow$
  finite ({((*r*, *s*), (*delta a r*, *delta a s*))| *a r s*. *a* $\in$ *set* (*alphabet*)} ^* `` {(*r*, *s*)})
  unfolding *rtrancl-fold-product Image-singleton*
  by (*auto intro*: *finite-subset*[*OF* - *finite-cartesian-product*[*OF fin fin*]])

lemma *termination*:
  assumes *wellformed r wellformed s*
  shows $\exists$ *st*. *closure* (*r*, *s*) = *Some st* (**is** $\exists$ -. *closure* ?*i* = -)
proof (*rule rtrancl-while-finite-Some*)
  show *finite* ({(*x*, *st*). *st* $\in$ *set* ((λ(*p*,*q*). *map* (λ*a*. (*delta a p*, *delta a q*)) *alphabet*) *x*)}* `` {?*i*})
    by (*rule finite-subset*[*OF Image-mono*[*OF rtrancl-mono*] *finite-rtrancl-delta-Image*[*OF assms*]]) *auto*
qed

lemma *completeness*:
assumes *wf r wf s Lang r* = *Lang s* **shows** *check-eqv r s*
proof −
  obtain *ws R* **where** *1*: *closure* (*init r*, *init s*) = *Some* (*ws*, *R*) **using** *termination*
    wellformed-init assms **by** *fastforce*
  with *closure-sound-complete*[*OF* - - *this*] *assms*
  show *check-eqv r s* **by** (*simp add*: *check-eqv-def Language-alphabet*)
qed

lemma *finite-rtrancl-delta-Image1*:
  wellformed r $\Longrightarrow$ *finite* ({(*r*, *s*). $\exists$ *a* $\in$ *set alphabet*. *s* = *delta a r*} ^* `` {*r*})
  unfolding *rtrancl-fold-product1* **by** (*auto intro*: *finite-subset*[*OF* - *fin*])

lemma
  assumes *wellformed r set as* $\subseteq$ *set alphabet*
  shows *reachable*: *reachable as r* = {*fold delta w r* | *w*. *w* $\in$ *lists* (*set as*)}
  and *finite-reachable*: *finite* (*reachable as r*)
proof −
  obtain *wsZ* **where** $*$: *rtrancl-while* (λ-. *True*) (λ*s*. *map* (λ*a*. *delta a s*) *as*) *r* = *Some wsZ*
    using *assms* **by** (*atomize-elim*, *intro rtrancl-while-finite-Some Image-mono rtrancl-mono*
      finite-subset[*OF* - *finite-rtrancl-delta-Image1* [*of r*]]) *auto*
  then show *reachable as r* = {*fold delta w r* | *w*. *w* $\in$ *lists* (*set as*)}
    unfolding *reachable-def* **by** (*cases wsZ*)
      (*auto dest*!: *rtrancl-while-Some split*: *if-splits simp*: *rtrancl-fold-product1 image-iff*)
  then show *finite* (*reachable as r*) **using** *assms* **by** (*force intro*: *finite-subset*[*OF*

- *fin*])
qed

end

2 Preliminaries

lemma *pred-Diff-0*[*simp*]: $0 \notin A \Longrightarrow i \in (\lambda x.\ x - Suc\ 0)$ ‘ $A \longleftrightarrow Suc\ i \in A$
  by (*cases i*) (*fastforce simp*: *image-iff le-Suc-eq elim*: *contrapos-np*)+

lemma *funpow-cycle-mult*: $(f\ \hat{}\hat{}\ k)\ x = x \Longrightarrow (f\ \hat{}\hat{}\ (m * k))\ x = x$
  by (*induct m*) (*auto simp*: *funpow-add*)

lemma *funpow-cycle*: $(f\ \hat{}\hat{}\ k)\ x = x \Longrightarrow (f\ \hat{}\hat{}\ l)\ x = (f\ \hat{}\hat{}\ (l\ mod\ k))\ x$
  by (*subst mod-div-equality*[*symmetric*, *of l k*])
    (*simp only*: *add.commute funpow-add funpow-cycle-mult o-apply*)

lemma *funpow-cycle-offset*:
  fixes $f :: {}'a \Rightarrow {}'a$
  assumes $(f\ \hat{}\hat{}\ k)\ x = (f\ \hat{}\hat{}\ i)\ x\ \ i \le k\ \ i \le l$
  shows $(f\ \hat{}\hat{}\ l)\ x = (f\ \hat{}\hat{}\ ((l - i)\ mod\ (k - i) + i))\ x$
proof −
  from *assms* **have**
    $(f\ \hat{}\hat{}\ (k - i))\ ((f\ \hat{}\hat{}\ i)\ x) = ((f\ \hat{}\hat{}\ i)\ x)$
    $(f\ \hat{}\hat{}\ l)\ x = (f\ \hat{}\hat{}\ (l - i))\ ((f\ \hat{}\hat{}\ i)\ x)$
    unfolding *fun-cong*[*OF funpow-add*[*symmetric*, *unfolded o-def*]] **by** *simp-all*
  from *funpow-cycle*[*OF this*(*1*), *of* $l - i$] *this*(*2*) **show** *?thesis*
    by (*simp add*: *funpow-add*)
qed

definition *dec k m* = (*if* $m > k$ *then* $m - Suc\ 0$ *else* $m :: nat$)

3 Abstract formulas

datatype-new (*discs-sels*) (′*a*, ′*k*) *aformula* =
  FBool bool
| *FBase* ′*a*
| *FNot* (′*a*, ′*k*) *aformula*
| *FOr* (′*a*, ′*k*) *aformula* (′*a*, ′*k*) *aformula*
| *FAnd* (′*a*, ′*k*) *aformula* (′*a*, ′*k*) *aformula*
| *FEx* ′*k* (′*a*, ′*k*) *aformula*
| *FAll* ′*k* (′*a*, ′*k*) *aformula*
datatype-compat *aformula*
derive *linorder aformula*

fun *nFOR* **where**
  $nFOR\ [] = FBool\ False$
$|\ nFOR\ [x] = x$
$|\ nFOR\ (x\ \#\ xs) = FOr\ x\ (nFOR\ xs)$

fun *nFAND* **where**
  $nFAND\ [] = FBool\ True$
$|\ nFAND\ [x] = x$
$|\ nFAND\ (x\ \#\ xs) = FAnd\ x\ (nFAND\ xs)$

definition $NFOR = nFOR\ o\ sorted\text{-}list\text{-}of\text{-}set$
definition $NFAND = nFAND\ o\ sorted\text{-}list\text{-}of\text{-}set$

fun *disjuncts* **where**
  $disjuncts\ (FOr\ \varphi\ \psi) = disjuncts\ \varphi \cup disjuncts\ \psi$
$|\ disjuncts\ \varphi = \{\varphi\}$

fun *conjuncts* **where**
  $conjuncts\ (FAnd\ \varphi\ \psi) = conjuncts\ \varphi \cup conjuncts\ \psi$
$|\ conjuncts\ \varphi = \{\varphi\}$

fun *disjuncts-list* **where**
  $disjuncts\text{-}list\ (FOr\ \varphi\ \psi) = disjuncts\text{-}list\ \varphi\ @\ disjuncts\text{-}list\ \psi$
$|\ disjuncts\text{-}list\ \varphi = [\varphi]$

fun *conjuncts-list* **where**
  $conjuncts\text{-}list\ (FAnd\ \varphi\ \psi) = conjuncts\text{-}list\ \varphi\ @\ conjuncts\text{-}list\ \psi$
$|\ conjuncts\text{-}list\ \varphi = [\varphi]$

lemma *finite-juncts*: $finite\ (disjuncts\ \varphi)\ finite\ (conjuncts\ \varphi)$
  and *nonempty-juncts*: $disjuncts\ \varphi \neq \{\}\ conjuncts\ \varphi \neq \{\}$
  by $(induct\ \varphi)\ auto$

lemma *juncts-eq-set-juncts-list*:
  $disjuncts\ \varphi = set\ (disjuncts\text{-}list\ \varphi)$
  $conjuncts\ \varphi = set\ (conjuncts\text{-}list\ \varphi)$
  by $(induct\ \varphi)\ auto$

lemma *notin-juncts*:
  $[\![\psi \in disjuncts\ \varphi;\ is\text{-}FOr\ \psi]\!] \Longrightarrow False$
  $[\![\psi \in conjuncts\ \varphi;\ is\text{-}FAnd\ \psi]\!] \Longrightarrow False$
  by $(induct\ \varphi)\ auto$

lemma *juncts-list-singleton*:
  $\neg\ is\text{-}FOr\ \varphi \Longrightarrow disjuncts\text{-}list\ \varphi = [\varphi]$
  $\neg\ is\text{-}FAnd\ \varphi \Longrightarrow conjuncts\text{-}list\ \varphi = [\varphi]$
  by $(induct\ \varphi)\ auto$

lemma *nonempty-juncts-list*: $conjuncts\text{-}list\ \varphi \neq []\ disjuncts\text{-}list\ \varphi \neq []$

using *nonempty-juncts*[*of* φ] **by** (*auto simp*: *Suc-le-eq juncts-eq-set-juncts-list*)

primrec *norm-ACI* ($\langle$-$\rangle$) **where**
$\langle FBool\ b\rangle = FBool\ b$
| $\langle FBase\ a\rangle = FBase\ a$
| $\langle FNot\ \varphi\rangle = FNot\ \langle\varphi\rangle$
| $\langle FOr\ \varphi\ \psi\rangle = NFOR\ (disjuncts\ (FOr\ \langle\varphi\rangle\ \langle\psi\rangle))$
| $\langle FAnd\ \varphi\ \psi\rangle = NFAND\ (conjuncts\ (FAnd\ \langle\varphi\rangle\ \langle\psi\rangle))$
| $\langle FEx\ k\ \varphi\rangle = FEx\ k\ \langle\varphi\rangle$
| $\langle FAll\ k\ \varphi\rangle = FAll\ k\ \langle\varphi\rangle$

fun *nf-ACI* **where**
nf-ACI ($FOr\ \psi 1\ \psi 2$) = ($\neg$ *is-FOr* $\psi 1$ $\wedge$ (*let* $\varphi s = \psi 1$ # *disjuncts-list* $\psi 2$ *in*
sorted φs $\wedge$ *distinct* φs $\wedge$ *nf-ACI* $\psi 1$ $\wedge$ *nf-ACI* $\psi 2$))
| *nf-ACI* ($FAnd\ \psi 1\ \psi 2$) = ($\neg$ *is-FAnd* $\psi 1$ $\wedge$ (*let* $\varphi s = \psi 1$ # *conjuncts-list* $\psi 2$ *in*
sorted φs $\wedge$ *distinct* φs $\wedge$ *nf-ACI* $\psi 1$ $\wedge$ *nf-ACI* $\psi 2$))
| *nf-ACI* ($FNot\ \varphi$) = *nf-ACI* φ
| *nf-ACI* ($FEx\ k\ \varphi$) = *nf-ACI* φ
| *nf-ACI* ($FAll\ k\ \varphi$) = *nf-ACI* φ
| *nf-ACI* φ = *True*

lemma *nf-ACI-D*:
nf-ACI $\varphi \Longrightarrow$ *sorted* (*disjuncts-list* φ)
nf-ACI $\varphi \Longrightarrow$ *sorted* (*conjuncts-list* φ)
nf-ACI $\varphi \Longrightarrow$ *distinct* (*disjuncts-list* φ)
nf-ACI $\varphi \Longrightarrow$ *distinct* (*conjuncts-list* φ)
nf-ACI $\varphi \Longrightarrow$ *list-all nf-ACI* (*disjuncts-list* φ)
nf-ACI $\varphi \Longrightarrow$ *list-all nf-ACI* (*conjuncts-list* φ)
by (*induct* φ) (*auto simp*: *juncts-list-singleton*)

lemma *disjuncts-list-nFOR*:
$[\![$*list-all* ($\lambda x.\ \neg$ *is-FOr* x) φs; $\varphi s \neq []]\!] \Longrightarrow$ *disjuncts-list* (*nFOR* φs) = φs
by (*induct* φs *rule*: *nFOR.induct*) (*auto simp*: *juncts-list-singleton*)

lemma *conjuncts-list-nFAND*:
$[\![$*list-all* ($\lambda x.\ \neg$ *is-FAnd* x) φs; $\varphi s \neq []]\!] \Longrightarrow$ *conjuncts-list* (*nFAND* φs) = φs
by (*induct* φs *rule*: *nFAND.induct*) (*auto simp*: *juncts-list-singleton*)

lemma *disjuncts-NFOR*:
$[\![$*finite* X; $X \neq \{\}$; $\forall x \in X.\ \neg$ *is-FOr* $x]\!] \Longrightarrow$ *disjuncts* (*NFOR* X) = X
unfolding *NFOR-def* **by** (*auto simp*: *juncts-eq-set-juncts-list list-all-iff disjuncts-list-nFOR*)

lemma *conjuncts-NFAND*:
$[\![$*finite* X; $X \neq \{\}$; $\forall x \in X.\ \neg$ *is-FAnd* $x]\!] \Longrightarrow$ *conjuncts* (*NFAND* X) = X
unfolding *NFAND-def* **by** (*auto simp*: *juncts-eq-set-juncts-list list-all-iff conjuncts-list-nFAND*)

lemma *nf-ACI-nFOR*:
$[\![$*sorted* φs; *distinct* φs; *list-all nf-ACI* φs; *list-all* ($\lambda x.\ \neg$ *is-FOr* x) $\varphi s]\!] \Longrightarrow$
nf-ACI (*nFOR* φs)

by (*induct* φs *rule*: *nFOR.induct*)
(*auto simp*: *juncts-list-singleton disjuncts-list-nFOR nf-ACI-D*)

lemma *nf-ACI-nFAND*:
$[\![$*sorted* φs; *distinct* φs; *list-all nf-ACI* φs; *list-all* (λx. $\neg$ *is-FAnd* x) $\varphi s]\!] \Longrightarrow$ *nf-ACI* (*nFAND* φs)
by (*induct* φs *rule*: *nFAND.induct*)
(*auto simp*: *juncts-list-singleton conjuncts-list-nFAND nf-ACI-D*)

lemma *nf-ACI-juncts*:
$[\![\psi \in$ *disjuncts* φ; *nf-ACI* $\varphi]\!] \Longrightarrow$ *nf-ACI* ψ
$[\![\psi \in$ *conjuncts* φ; *nf-ACI* $\varphi]\!] \Longrightarrow$ *nf-ACI* ψ
by (*induct* φ) *auto*

lemma *nf-ACI-norm-ACI*: *nf-ACI* $\langle\varphi\rangle$
by (*induct* φ)
(*force simp*: *NFOR-def NFAND-def finite-juncts list-all-iff*
intro!: *nf-ACI-nFOR nf-ACI-nFAND elim*: *nf-ACI-juncts notin-juncts*)+

lemma *nFOR-Cons*: *nFOR* (x # *xs*) = (*if xs* = [] *then* x *else FOr* x (*nFOR xs*))
by (*cases xs*) *simp-all*

lemma *nFAND-Cons*: *nFAND* (x # *xs*) = (*if xs* = [] *then* x *else FAnd* x (*nFAND xs*))
by (*cases xs*) *simp-all*

lemma *nFOR-disjuncts*: *nf-ACI* $\psi \Longrightarrow$ *nFOR* (*disjuncts-list* ψ) = ψ
by (*induct* ψ) (*auto simp*: *juncts-list-singleton nFOR-Cons*)

lemma *nFAND-conjuncts*: *nf-ACI* $\psi \Longrightarrow$ *nFAND* (*conjuncts-list* ψ) = ψ
by (*induct* ψ) (*auto simp*: *juncts-list-singleton nFAND-Cons*)

lemma *NFOR-disjuncts*: *nf-ACI* $\psi \Longrightarrow$ *NFOR* (*disjuncts* ψ) = ψ
using *nFOR-disjuncts*[*of* ψ] **unfolding** *NFOR-def o-apply juncts-eq-set-juncts-list*
by (*metis finite-set finite-sorted-distinct-unique nf-ACI-D*(*1*,*3*) *sorted-list-of-set*)

lemma *NFAND-conjuncts*: *nf-ACI* $\psi \Longrightarrow$ *NFAND* (*conjuncts* ψ) = ψ
using *nFAND-conjuncts*[*of* ψ] **unfolding** *NFAND-def o-apply juncts-eq-set-juncts-list*
by (*metis finite-set finite-sorted-distinct-unique nf-ACI-D*(*2*,*4*) *sorted-list-of-set*)

lemma *norm-ACI-if-nf-ACI*: *nf-ACI* $\varphi \Longrightarrow \langle\varphi\rangle = \varphi$
by (*induct* φ)
(*auto simp*: *juncts-list-singleton juncts-eq-set-juncts-list nonempty-juncts-list*
NFOR-def NFAND-def nFOR-Cons nFAND-Cons nFOR-disjuncts nFAND-conjuncts
sorted-Cons sorted-list-of-set-sort-remdups distinct-remdups-id sorted-sort-id insort-is-Cons)

lemma *norm-ACI-idem*: $\langle\langle\varphi\rangle\rangle = \langle\varphi\rangle$
by (*metis nf-ACI-norm-ACI norm-ACI-if-nf-ACI*)

lemma *norm-ACI-juncts*:
$nf\text{-}ACI\ \varphi \Longrightarrow norm\text{-}ACI\ `\ disjuncts\ \varphi = disjuncts\ \varphi$
$nf\text{-}ACI\ \varphi \Longrightarrow norm\text{-}ACI\ `\ conjuncts\ \varphi = conjuncts\ \varphi$
by (*drule nf-ACI-D*(*5*,*6*), *force simp*: *list-all-iff juncts-eq-set-juncts-list norm-ACI-if-nf-ACI*)+

lemma
norm-ACI-NFOR: $nf\text{-}ACI\ \varphi \Longrightarrow \varphi = NFOR\ (norm\text{-}ACI\ `\ disjuncts\ \varphi)$ **and**
norm-ACI-NFAND: $nf\text{-}ACI\ \varphi \Longrightarrow \varphi = NFAND\ (norm\text{-}ACI\ `\ conjuncts\ \varphi)$
by (*simp-all add*: *norm-ACI-juncts NFOR-disjuncts NFAND-conjuncts*)

locale *Formula-Operations* =
fixes $TYPEVARS :: 'a :: linorder \times 'i \times 'k :: linorder \times 'n \times 'x \times 'v$

and $SUC :: 'k \Rightarrow 'n \Rightarrow 'n$
and $LESS :: 'k \Rightarrow nat \Rightarrow 'n \Rightarrow bool$

and $assigns :: nat \Rightarrow 'i \Rightarrow 'k \Rightarrow 'v$ (-ˉ- [*900*, *999*, *999*] *999*)
and $nvars :: 'i \Rightarrow 'n$ ($\#_V$ - [*1000*] *900*)
and $Extend :: 'k \Rightarrow nat \Rightarrow 'i \Rightarrow 'v \Rightarrow 'i$
and $CONS :: 'x \Rightarrow 'i \Rightarrow 'i$
and $SNOC :: 'x \Rightarrow 'i \Rightarrow 'i$
and $Length :: 'i \Rightarrow nat$

and $extend :: 'k \Rightarrow bool \Rightarrow 'x \Rightarrow 'x$
and $size :: 'x \Rightarrow 'n$
and $zero :: 'n \Rightarrow 'x$

and $eval :: 'v \Rightarrow nat \Rightarrow bool$
and $downshift :: 'v \Rightarrow 'v$
and $upshift :: 'v \Rightarrow 'v$
and $add :: nat \Rightarrow 'v \Rightarrow 'v$
and $cut :: nat \Rightarrow 'v \Rightarrow 'v$
and $len :: 'v \Rightarrow nat$

and $restrict :: 'k \Rightarrow 'v \Rightarrow bool$
and $Restrict :: 'k \Rightarrow nat \Rightarrow ('a, 'k)\ aformula$

and $left\text{-}formula0 :: 'a \Rightarrow bool$
and $FV0 :: 'a \Rightarrow ('k \times nat)\ list$
and $find0 :: 'k \Rightarrow nat \Rightarrow 'a \Rightarrow bool$
and $wf0 :: 'n \Rightarrow 'a \Rightarrow bool$

and $decr0$ :: $'k \Rightarrow nat \Rightarrow 'a \Rightarrow 'a$
and $satisfies0$ :: $'i \Rightarrow 'a \Rightarrow bool$ (**infix** $\models_0$ 50)
and $nullable0$ :: $'a \Rightarrow bool$
and $lderiv0$ :: $'x \Rightarrow 'a \Rightarrow ('a, 'k)$ $aformula$
and $rderiv0$ :: $'x \Rightarrow 'a \Rightarrow ('a, 'k)$ $aformula$
begin

abbreviation $LEQ\ k\ l\ n \equiv LESS\ k\ l\ (SUC\ k\ n)$

primrec FV **where**
$FV\ (FBool\ \text{-}) = []$
| $FV\ (FBase\ a) = FV0\ a$
| $FV\ (FNot\ \varphi) = FV\ \varphi$
| $FV\ (FOr\ \varphi\ \psi) = List.union\ (FV\ \varphi)\ (FV\ \psi)$
| $FV\ (FAnd\ \varphi\ \psi) = List.union\ (FV\ \varphi)\ (FV\ \psi)$
| $FV\ (FEx\ k\ \varphi) = map\ (\lambda(k', x).\ (k', if\ k = k'\ then\ x - 1\ else\ x))\ (List.remove1\ (k, 0)\ (FV\ \varphi))$
| $FV\ (FAll\ k\ \varphi) = map\ (\lambda(k', x).\ (k', if\ k = k'\ then\ x - 1\ else\ x))\ (List.remove1\ (k, 0)\ (FV\ \varphi))$

primrec $find$ **where**
$find\ k\ l\ (FBool\ \text{-}) = False$
| $find\ k\ l\ (FBase\ a) = find0\ k\ l\ a$
| $find\ k\ l\ (FNot\ \varphi) = find\ k\ l\ \varphi$
| $find\ k\ l\ (FOr\ \varphi\ \psi) = (find\ k\ l\ \varphi \vee find\ k\ l\ \psi)$
| $find\ k\ l\ (FAnd\ \varphi\ \psi) = (find\ k\ l\ \varphi \vee find\ k\ l\ \psi)$
| $find\ k\ l\ (FEx\ k'\ \varphi) = find\ k\ (if\ k = k'\ then\ Suc\ l\ else\ l)\ \varphi$
| $find\ k\ l\ (FAll\ k'\ \varphi) = find\ k\ (if\ k = k'\ then\ Suc\ l\ else\ l)\ \varphi$

primrec wf :: $'n \Rightarrow ('a, 'k)\ aformula \Rightarrow bool$ **where**
$wf\ n\ (FBool\ \text{-}) = True$
| $wf\ n\ (FBase\ a) = wf0\ n\ a$
| $wf\ n\ (FNot\ \varphi) = wf\ n\ \varphi$
| $wf\ n\ (FOr\ \varphi\ \psi) = (wf\ n\ \varphi \wedge wf\ n\ \psi)$
| $wf\ n\ (FAnd\ \varphi\ \psi) = (wf\ n\ \varphi \wedge wf\ n\ \psi)$
| $wf\ n\ (FEx\ k\ \varphi) = wf\ (SUC\ k\ n)\ \varphi$
| $wf\ n\ (FAll\ k\ \varphi) = wf\ (SUC\ k\ n)\ \varphi$

primrec $left\text{-}formula$ :: $('a, 'k)\ aformula \Rightarrow bool$ **where**
$left\text{-}formula\ (FBool\ \text{-}) = True$
| $left\text{-}formula\ (FBase\ a) = left\text{-}formula0\ a$
| $left\text{-}formula\ (FNot\ \varphi) = left\text{-}formula\ \varphi$
| $left\text{-}formula\ (FOr\ \varphi\ \psi) = (left\text{-}formula\ \varphi \wedge left\text{-}formula\ \psi)$
| $left\text{-}formula\ (FAnd\ \varphi\ \psi) = (left\text{-}formula\ \varphi \wedge left\text{-}formula\ \psi)$
| $left\text{-}formula\ (FEx\ k\ \varphi) = left\text{-}formula\ \varphi$
| $left\text{-}formula\ (FAll\ k\ \varphi) = left\text{-}formula\ \varphi$

primrec $decr$ :: $'k \Rightarrow nat \Rightarrow ('a, 'k)\ aformula \Rightarrow ('a, 'k)\ aformula$ **where**
$decr\ k\ l\ (FBool\ b) = FBool\ b$

| *decr k l* $(FBase\ a) = FBase\ (decr0\ k\ l\ a)$
| *decr k l* $(FNot\ \varphi) = FNot\ (decr\ k\ l\ \varphi)$
| *decr k l* $(FOr\ \varphi\ \psi) = FOr\ (decr\ k\ l\ \varphi)\ (decr\ k\ l\ \psi)$
| *decr k l* $(FAnd\ \varphi\ \psi) = FAnd\ (decr\ k\ l\ \varphi)\ (decr\ k\ l\ \psi)$
| *decr k l* $(FEx\ k'\ \varphi) = FEx\ k'\ (decr\ k\ (if\ k = k'\ then\ Suc\ l\ else\ l)\ \varphi)$
| *decr k l* $(FAll\ k'\ \varphi) = FAll\ k'\ (decr\ k\ (if\ k = k'\ then\ Suc\ l\ else\ l)\ \varphi)$

primrec *satisfies-gen* :: $('k \Rightarrow 'v \Rightarrow nat \Rightarrow bool) \Rightarrow 'i \Rightarrow ('a, 'k)\ aformula \Rightarrow bool$
where
satisfies-gen $r\ \mathfrak{A}\ (FBool\ b) = b$
| *satisfies-gen* $r\ \mathfrak{A}\ (FBase\ a) = (\mathfrak{A} \models_0 a)$
| *satisfies-gen* $r\ \mathfrak{A}\ (FNot\ \varphi) = (\neg\ satisfies\text{-}gen\ r\ \mathfrak{A}\ \varphi)$
| *satisfies-gen* $r\ \mathfrak{A}\ (FOr\ \varphi_1\ \varphi_2) = (satisfies\text{-}gen\ r\ \mathfrak{A}\ \varphi_1 \vee satisfies\text{-}gen\ r\ \mathfrak{A}\ \varphi_2)$
| *satisfies-gen* $r\ \mathfrak{A}\ (FAnd\ \varphi_1\ \varphi_2) = (satisfies\text{-}gen\ r\ \mathfrak{A}\ \varphi_1 \wedge satisfies\text{-}gen\ r\ \mathfrak{A}\ \varphi_2)$
| *satisfies-gen* $r\ \mathfrak{A}\ (FEx\ k\ \varphi) = (\exists P.\ r\ k\ P\ (Length\ \mathfrak{A}) \wedge satisfies\text{-}gen\ r\ (Extend\ k\ 0\ \mathfrak{A}\ P)\ \varphi)$
| *satisfies-gen* $r\ \mathfrak{A}\ (FAll\ k\ \varphi) = (\forall P.\ r\ k\ P\ (Length\ \mathfrak{A}) \longrightarrow satisfies\text{-}gen\ r\ (Extend\ k\ 0\ \mathfrak{A}\ P)\ \varphi)$

abbreviation *satisfies* (**infix** $\models$ *50*) **where**
$\mathfrak{A} \models \varphi \equiv satisfies\text{-}gen\ (\lambda\text{- - -}.\ True)\ \mathfrak{A}\ \varphi$

abbreviation *satisfies-bounded* (**infix** $\models_b$ *50*) **where**
$\mathfrak{A} \models_b \varphi \equiv satisfies\text{-}gen\ (\lambda\text{- }P\ n.\ len\ P \leq n)\ \mathfrak{A}\ \varphi$

abbreviation *sat-vars-gen* **where**
$sat\text{-}vars\text{-}gen\ r\ V\ \mathfrak{A}\ \varphi \equiv$
$satisfies\text{-}gen\ (\lambda k\ P\ n.\ restrict\ k\ P \wedge r\ k\ P\ n)\ \mathfrak{A}\ \varphi \wedge (\forall (k, x) \in set\ V.\ restrict\ k\ (x^{\mathfrak{A}}k))$

definition *sat* **where**
$sat\ \mathfrak{A}\ \varphi \equiv sat\text{-}vars\text{-}gen\ (\lambda\text{- - -}.\ True)\ (FV\ \varphi)\ \mathfrak{A}\ \varphi$

definition sat_b **where**
$sat_b\ \mathfrak{A}\ \varphi \equiv sat\text{-}vars\text{-}gen\ (\lambda\text{- }P\ n.\ len\ P \leq n)\ (FV\ \varphi)\ \mathfrak{A}\ \varphi$

fun *RESTR* **where**
$RESTR\ (FOr\ \varphi\ \psi) = FOr\ (RESTR\ \varphi)\ (RESTR\ \psi)$
| $RESTR\ (FAnd\ \varphi\ \psi) = FAnd\ (RESTR\ \varphi)\ (RESTR\ \psi)$
| $RESTR\ (FNot\ \varphi) = FNot\ (RESTR\ \varphi)$
| $RESTR\ (FEx\ k\ \varphi) = FEx\ k\ (FAnd\ (Restrict\ k\ 0)\ (RESTR\ \varphi))$
| $RESTR\ (FAll\ k\ \varphi) = FAll\ k\ (FOr\ (FNot\ (Restrict\ k\ 0))\ (RESTR\ \varphi))$
| $RESTR\ \varphi = \varphi$

abbreviation *RESTRICT-VARS* **where**
$RESTRICT\text{-}VARS\ vs\ \varphi \equiv foldr\ (\lambda(k, x)\ \varphi.\ FAnd\ (Restrict\ k\ x)\ \varphi)\ vs\ (RESTR\ \varphi)$

definition *RESTRICT* **where**

$RESTRICT\ \varphi \equiv RESTRICT\text{-}VARS\ (FV\ \varphi)\ \varphi$

primrec $nullable :: ('a, 'k)\ aformula \Rightarrow bool$ **where**
$nullable\ (FBool\ b) = b$
$|\ nullable\ (FBase\ a) = nullable0\ a$
$|\ nullable\ (FNot\ \varphi) = (\neg\ nullable\ \varphi)$
$|\ nullable\ (FOr\ \varphi\ \psi) = (nullable\ \varphi \vee nullable\ \psi)$
$|\ nullable\ (FAnd\ \varphi\ \psi) = (nullable\ \varphi \wedge nullable\ \psi)$
$|\ nullable\ (FEx\ k\ \varphi) = nullable\ \varphi$
$|\ nullable\ (FAll\ k\ \varphi) = nullable\ \varphi$

fun $nFOr :: ('a, 'k)\ aformula \Rightarrow ('a, 'k)\ aformula \Rightarrow ('a, 'k)\ aformula$ **where**
$nFOr\ (FBool\ b1)\ (FBool\ b2) = FBool\ (b1 \vee b2)$
$|\ nFOr\ (FBool\ b)\ \psi = (if\ b\ then\ FBool\ True\ else\ \psi)$
$|\ nFOr\ \varphi\ (FBool\ b) = (if\ b\ then\ FBool\ True\ else\ \varphi)$
$|\ nFOr\ (FOr\ \varphi 1\ \varphi 2)\ \psi = nFOr\ \varphi 1\ (nFOr\ \varphi 2\ \psi)$
$|\ nFOr\ \varphi\ (FOr\ \psi 1\ \psi 2) =$
$(if\ \varphi = \psi 1\ then\ FOr\ \psi 1\ \psi 2$
$else\ if\ \varphi < \psi 1\ then\ FOr\ \varphi\ (FOr\ \psi 1\ \psi 2)$
$else\ FOr\ \psi 1\ (nFOr\ \varphi\ \psi 2))$
$|\ nFOr\ \varphi\ \psi =$
$(if\ \varphi = \psi\ then\ \varphi$
$else\ if\ \varphi < \psi\ then\ FOr\ \varphi\ \psi$
$else\ FOr\ \psi\ \varphi)$

fun $nFAnd :: ('a, 'k)\ aformula \Rightarrow ('a, 'k)\ aformula \Rightarrow ('a, 'k)\ aformula$ **where**
$nFAnd\ (FBool\ b1)\ (FBool\ b2) = FBool\ (b1 \wedge b2)$
$|\ nFAnd\ (FBool\ b)\ \psi = (if\ b\ then\ \psi\ else\ FBool\ False)$
$|\ nFAnd\ \varphi\ (FBool\ b) = (if\ b\ then\ \varphi\ else\ FBool\ False)$
$|\ nFAnd\ (FAnd\ \varphi 1\ \varphi 2)\ \psi = nFAnd\ \varphi 1\ (nFAnd\ \varphi 2\ \psi)$
$|\ nFAnd\ \varphi\ (FAnd\ \psi 1\ \psi 2) =$
$(if\ \varphi = \psi 1\ then\ FAnd\ \psi 1\ \psi 2$
$else\ if\ \varphi < \psi 1\ then\ FAnd\ \varphi\ (FAnd\ \psi 1\ \psi 2)$
$else\ FAnd\ \psi 1\ (nFAnd\ \varphi\ \psi 2))$
$|\ nFAnd\ \varphi\ \psi =$
$(if\ \varphi = \psi\ then\ \varphi$
$else\ if\ \varphi < \psi\ then\ FAnd\ \varphi\ \psi$
$else\ FAnd\ \psi\ \varphi)$

fun $nFNot :: ('a, 'k)\ aformula \Rightarrow ('a, 'k)\ aformula$ **where**
$nFNot\ (FNot\ \varphi) = \varphi$
$|\ nFNot\ (FOr\ \varphi\ \psi) = nFAnd\ (nFNot\ \varphi)\ (nFNot\ \psi)$
$|\ nFNot\ (FAnd\ \varphi\ \psi) = nFOr\ (nFNot\ \varphi)\ (nFNot\ \psi)$
$|\ nFNot\ (FEx\ b\ \varphi) = FAll\ b\ (nFNot\ \varphi)$
$|\ nFNot\ (FAll\ b\ \varphi) = FEx\ b\ (nFNot\ \varphi)$
$|\ nFNot\ (FBool\ b) = FBool\ (\neg\ b)$
$|\ nFNot\ \varphi = FNot\ \varphi$

fun $nFEx :: 'k \Rightarrow ('a, 'k)\ aformula \Rightarrow ('a, 'k)\ aformula$ **where**

$nFEx\ k\ (FOr\ \varphi\ \psi) = nFOr\ (nFEx\ k\ \varphi)\ (nFEx\ k\ \psi)$
$|\ nFEx\ k\ \varphi = (if\ find\ k\ 0\ \varphi\ then\ FEx\ k\ \varphi\ else\ decr\ k\ 0\ \varphi)$

fun *nFAll* **where**
$nFAll\ k\ (FAnd\ \varphi\ \psi) = nFAnd\ (nFAll\ k\ \varphi)\ (nFAll\ k\ \psi)$
$|\ nFAll\ k\ \varphi = (if\ find\ k\ 0\ \varphi\ then\ FAll\ k\ \varphi\ else\ decr\ k\ 0\ \varphi)$

fun *norm* **where**
$norm\ (FOr\ \varphi\ \psi) = nFOr\ (norm\ \varphi)\ (norm\ \psi)$
$|\ norm\ (FAnd\ \varphi\ \psi) = nFAnd\ (norm\ \varphi)\ (norm\ \psi)$
$|\ norm\ (FNot\ \varphi) = nFNot\ (norm\ \varphi)$
$|\ norm\ (FEx\ k\ \varphi) = nFEx\ k\ (norm\ \varphi)$
$|\ norm\ (FAll\ k\ \varphi) = nFAll\ k\ (norm\ \varphi)$
$|\ norm\ \varphi = \varphi$

context
fixes $deriv0 :: 'x \Rightarrow 'a \Rightarrow ('a,\ 'k)\ aformula$
begin

primrec $deriv :: 'x \Rightarrow ('a,\ 'k)\ aformula \Rightarrow ('a,\ 'k)\ aformula$ **where**
$deriv\ x\ (FBool\ b) = FBool\ b$
$|\ deriv\ x\ (FBase\ a) = deriv0\ x\ a$
$|\ deriv\ x\ (FNot\ \varphi) = FNot\ (deriv\ x\ \varphi)$
$|\ deriv\ x\ (FOr\ \varphi\ \psi) = FOr\ (deriv\ x\ \varphi)\ (deriv\ x\ \psi)$
$|\ deriv\ x\ (FAnd\ \varphi\ \psi) = FAnd\ (deriv\ x\ \varphi)\ (deriv\ x\ \psi)$
$|\ deriv\ x\ (FEx\ k\ \varphi) = FEx\ k\ (FOr\ (deriv\ (extend\ k\ True\ x)\ \varphi)\ (deriv\ (extend\ k\ False\ x)\ \varphi))$
$|\ deriv\ x\ (FAll\ k\ \varphi) = FAll\ k\ (FAnd\ (deriv\ (extend\ k\ True\ x)\ \varphi)\ (deriv\ (extend\ k\ False\ x)\ \varphi))$

end

abbreviation $lderiv \equiv deriv\ lderiv0$
abbreviation $rderiv \equiv deriv\ rderiv0$

lemma *fold-deriv-FBool*: $fold\ (deriv\ d0)\ xs\ (FBool\ b) = FBool\ b$
by $(induct\ xs)\ auto$

lemma *fold-deriv-FNot*:
$fold\ (deriv\ d0)\ xs\ (FNot\ \varphi) = FNot\ (fold\ (deriv\ d0)\ xs\ \varphi)$
by $(induct\ xs\ arbitrary{:}\ \varphi)\ auto$

lemma *fold-deriv-FOr*:
$fold\ (deriv\ d0)\ xs\ (FOr\ \varphi\ \psi) = FOr\ (fold\ (deriv\ d0)\ xs\ \varphi)\ (fold\ (deriv\ d0)\ xs\ \psi)$
by $(induct\ xs\ arbitrary{:}\ \varphi\ \psi)\ auto$

lemma *fold-deriv-FAnd*:
fold (*deriv d0*) *xs* (*FAnd* φ ψ) = *FAnd* (*fold* (*deriv d0*) *xs* φ) (*fold* (*deriv d0*) *xs* ψ)
by (*induct xs arbitrary*: φ ψ) *auto*

lemma *fold-deriv-FEx*:
$\{\langle$*fold* (*deriv d0*) *xs* (*FEx k* φ)$\rangle$ | *xs*. *True*$\}$ $\subseteq$
$\{$*FEx k* ψ | ψ. *nf-ACI* ψ $\wedge$ *disjuncts* ψ $\subseteq$ ($\bigcup$*xs*. *disjuncts* $\langle$*fold* (*deriv d0*) *xs* $\varphi\rangle$)$\}$
proof –
{ fix *xs*
have $\exists\psi$. $\langle$*fold* (*deriv d0*) *xs* (*FEx k* φ)$\rangle$ = *FEx k* ψ $\wedge$
nf-ACI ψ $\wedge$ *disjuncts* ψ $\subseteq$ ($\bigcup$*xs*. *disjuncts* $\langle$*fold* (*deriv d0*) *xs* $\varphi\rangle$)
proof (*induct xs arbitrary*: φ)
case (*Cons x xs*)
let *?*φ = *FOr* (*deriv d0* (*extend k True x*) φ) (*deriv d0* (*extend k False x*) φ)
from *Cons*[*of ?*φ] **obtain** ψ **where** $\langle$*fold* (*deriv d0*) *xs* (*FEx k ?*φ)$\rangle$ = *FEx k* ψ
nf-ACI ψ **and** $*$: *disjuncts* ψ $\subseteq$ ($\bigcup$*xs*. *disjuncts* $\langle$*fold* (*deriv d0*) *xs ?*$\varphi\rangle$) **by** *blast+*
then show *?case*
proof (*intro exI conjI*)
have ($\bigcup$*xs*. *disjuncts* $\langle$*fold* (*deriv d0*) *xs ?*$\varphi\rangle$) $\subseteq$
($\bigcup$*xs*. *disjuncts* $\langle$*fold* (*Formula-Operations.deriv extend d0*) *xs* $\varphi\rangle$)
by (*force simp*: *fold-deriv-FOr finite-juncts nonempty-juncts nf-ACI-juncts nf-ACI-norm-ACI*
dest: *notin-juncts set-mp*[*OF equalityD1*[*OF disjuncts-NFOR*], *rotated* -1]
intro: *exI*[*of - extend k b x # xs* **for** *b xs*])
with $*$ **show** *disjuncts* ψ $\subseteq$ ... **by** *blast*
qed *simp-all*
qed (*auto simp*: *nf-ACI-norm-ACI intro*!: *exI*[*of -* []])
}
then show *?thesis* **by** *blast*
qed

lemma *fold-deriv-FAll*:
$\{\langle$*fold* (*deriv d0*) *xs* (*FAll k* φ)$\rangle$ | *xs*. *True*$\}$ $\subseteq$
$\{$*FAll k* ψ | ψ. *nf-ACI* ψ $\wedge$ *conjuncts* ψ $\subseteq$ ($\bigcup$*xs*. *conjuncts* $\langle$*fold* (*deriv d0*) *xs* $\varphi\rangle$)$\}$
proof –
{ fix *xs*
have $\exists\psi$. $\langle$*fold* (*deriv d0*) *xs* (*FAll k* φ)$\rangle$ = *FAll k* ψ $\wedge$
nf-ACI ψ $\wedge$ *conjuncts* ψ $\subseteq$ ($\bigcup$*xs*. *conjuncts* $\langle$*fold* (*deriv d0*) *xs* $\varphi\rangle$)
proof (*induct xs arbitrary*: φ)
case (*Cons x xs*)
let *?*φ = *FAnd* (*deriv d0* (*extend k True x*) φ) (*deriv d0* (*extend k False x*) φ)
from *Cons*[*of ?*φ] **obtain** ψ **where** $\langle$*fold* (*deriv d0*) *xs* (*FAll k ?*φ)$\rangle$ = *FAll k* ψ

nf-ACI ψ **and** $*$: *conjuncts* $\psi \subseteq (\bigcup xs.$ *conjuncts* ⟨*fold* (*deriv d0*) *xs ?*φ⟩)
by *blast*+
then show *?case*
proof (*intro exI conjI*)
have $(\bigcup xs.$ *conjuncts* ⟨*fold* (*deriv d0*) *xs ?*φ⟩) $\subseteq$
$(\bigcup xs.$ *conjuncts* ⟨*fold* (*Formula-Operations.deriv extend d0*) *xs* φ⟩)
by (*force simp*: *fold-deriv-FAnd finite-juncts nonempty-juncts nf-ACI-juncts nf-ACI-norm-ACI*
dest: *notin-juncts set-mp*[*OF equalityD1*[*OF conjuncts-NFAND*], *rotated* −*1*]
intro: *exI*[*of - extend k b x # xs* **for** *b xs*])
with $*$ **show** *conjuncts* $\psi \subseteq \ldots$ **by** *blast*
qed *simp-all*
qed (*auto simp*: *nf-ACI-norm-ACI intro*!: *exI*[*of -* []])
}
then show *?thesis* **by** *blast*
qed

lemma *finite-norm-ACI-juncts*:
fixes *f* :: ('*a*, '*k*) *aformula* ⇒ ('*a*, '*k*) *aformula*
shows *finite B* ⟹ *finite* {*f* φ | φ. *nf-ACI* φ ∧ *disjuncts* $\varphi \subseteq B$}
finite B ⟹ *finite* {*f* φ | φ. *nf-ACI* φ ∧ *conjuncts* $\varphi \subseteq B$}
by (*elim finite-surj*[*OF iffD2*[*OF finite-Pow-iff*], *of - - f o NFOR o image norm-ACI*]
finite-surj[*OF iffD2*[*OF finite-Pow-iff*], *of - - f o NFAND o image norm-ACI*],
force simp: *Pow-def image-Collect intro*: *arg-cong*[*OF norm-ACI-NFOR*] *arg-cong*[*OF norm-ACI-NFAND*])+

end

locale *Formula* = *Formula-Operations*
where *TYPEVARS* = *TYPEVARS*
for *TYPEVARS* :: '*a*(*l*) :: *linorder* (×'*ar* :: *linorder*)× '*i* × '*k* :: *linorder* × '*n* × '*x* × '*v* +

assumes *SUC-SUC*: *SUC k* (*SUC k' idx*) = *SUC k'* (*SUC k idx*)
and *LEQ-0*: *LEQ k 0 idx*
and *LESS-SUC*: *LEQ k* (*Suc l*) *idx* = *LESS k l idx*
$k \neq k'$ ⟹ *LESS k l* (*SUC k' idx*) = *LESS k l idx*

and *nvars-Extend*: $\#_V$ (*Extend k i* $\mathfrak{A}$ *P*) = *SUC k* ($\#_V$ $\mathfrak{A}$)
and *Length-Extend*: *Length* (*Extend k i* $\mathfrak{A}$ *P*) = *max* (*Length* $\mathfrak{A}$) (*len P*)
and *Length-0-inj*: ⟦*Length* $\mathfrak{A}$ = *0*; *Length* $\mathfrak{B}$ = *0*; $\#_V$ $\mathfrak{A}$ = $\#_V$ $\mathfrak{B}$⟧ ⟹ $\mathfrak{A} = \mathfrak{B}$
and *ex-Length-0*: $\exists\mathfrak{A}$. *Length* $\mathfrak{A}$ = *0* ∧ $\#_V$ $\mathfrak{A}$ = *idx*
and *Extend-commute-safe*: ⟦$j \leq i$; *LEQ k i* ($\#_V$ $\mathfrak{A}$)⟧ ⟹
Extend k j (*Extend k i* $\mathfrak{A}$ *P*) *Q* = *Extend k* (*Suc i*) (*Extend k j* $\mathfrak{A}$ *Q*) *P*
and *Extend-commute-unsafe*: $k \neq k'$ ⟹
Extend k j (*Extend k' i* $\mathfrak{A}$ *P*) *Q* = *Extend k' i* (*Extend k j* $\mathfrak{A}$ *Q*) *P*

and *assigns-Extend*: $LEQ\ ord\ i\ (\#_V\ \mathfrak{A}) \Longrightarrow$
$m^{Extend\ ord\ i\ \mathfrak{A}\ P}{}_{ord'} = (if\ ord = ord'\ then\ (if\ m = i\ then\ P\ else\ dec\ i\ m^{\mathfrak{A}}{}_{ord})\ else\ m^{\mathfrak{A}}{}_{ord'})$
and *assigns-SNOC-zero*: $LESS\ ord\ m\ (\#_V\ \mathfrak{A}) \Longrightarrow m^{SNOC\ (zero\ (\#_V\ \mathfrak{A}))\ \mathfrak{A}}{}_{ord} = m^{\mathfrak{A}}{}_{ord}$
and *Length-CONS*: $Length\ (CONS\ x\ \mathfrak{A}) = Length\ \mathfrak{A} + 1$
and *Length-SNOC*: $Length\ (SNOC\ x\ \mathfrak{A}) = Suc\ (Length\ \mathfrak{A})$
and *nvars-CONS*: $\#_V\ (CONS\ x\ \mathfrak{A}) = \#_V\ \mathfrak{A}$
and *nvars-SNOC*: $\#_V\ (SNOC\ x\ \mathfrak{A}) = \#_V\ \mathfrak{A}$
and *Extend-CONS*: $\#_V\ \mathfrak{A} = size\ x \Longrightarrow Extend\ k\ 0\ (CONS\ x\ \mathfrak{A})\ P =$
$CONS\ (extend\ k\ (if\ eval\ P\ 0\ then\ True\ else\ False)\ x)\ (Extend\ k\ 0\ \mathfrak{A}\ (downshift\ P))$
and *Extend-SNOC-cut*: $\#_V\ \mathfrak{A} = size\ x \Longrightarrow len\ P \leq Length\ (SNOC\ x\ \mathfrak{A}) \Longrightarrow$
$Extend\ ord\ 0\ (SNOC\ x\ \mathfrak{A})\ P =$
$SNOC\ (extend\ ord\ (if\ eval\ P\ (Length\ \mathfrak{A})\ then\ True\ else\ False)\ x)\ (Extend\ ord\ 0\ \mathfrak{A}\ (cut\ (Length\ \mathfrak{A})\ P))$

and *size-zero*: $size\ (zero\ idx) = idx$
and *size-extend*: $size\ (extend\ k\ b\ x) = SUC\ k\ (size\ x)$

and *downshift-upshift*: $downshift\ (upshift\ P) = P$
and *downshift-add-zero*: $downshift\ (add\ 0\ P) = downshift\ P$
and *eval-add*: $eval\ (add\ n\ P)\ n$
and *eval-upshift*: $\neg\ eval\ (upshift\ P)\ 0$
and *eval-ge-len*: $p \geq len\ P \Longrightarrow \neg\ eval\ P\ p$
and *len-cut-le*: $len\ (cut\ n\ P) \leq n$
and *len-cut*: $len\ P \leq n \Longrightarrow cut\ n\ P = P$
and *cut-add*: $cut\ n\ (add\ m\ P) = (if\ m \geq n\ then\ cut\ n\ P\ else\ add\ m\ (cut\ n\ P))$
and *len-add*: $len\ (add\ m\ P) = max\ (Suc\ m)\ (len\ P)$
and *len-upshift*: $len\ (upshift\ P) = (case\ len\ P\ of\ 0 \Rightarrow 0 \mid n \Rightarrow Suc\ n)$
and *len-downshift*: $len\ (downshift\ P) = (case\ len\ P\ of\ 0 \Rightarrow 0 \mid Suc\ n \Rightarrow n)$

and *wf0-decr0*: $[\![wf0\ (SUC\ k\ idx)\ a;\ LESS\ k\ l\ (SUC\ k\ idx);\ \neg\ find0\ k\ l\ a]\!] \Longrightarrow wf0\ idx\ (decr0\ k\ l\ a)$
and *left-formula0-decr0*: $left\text{-}formula0\ \varphi \Longrightarrow left\text{-}formula0\ (decr0\ k\ l\ \varphi)$
and *Extend-satisfies0*: $[\![\neg\ find0\ k\ i\ a;\ LESS\ k\ i\ (SUC\ k\ (\#_V\ \mathfrak{A}))]\!] \Longrightarrow$
$Extend\ k\ i\ \mathfrak{A}\ P \models_0 a \longleftrightarrow \mathfrak{A} \models_0 decr0\ k\ i\ a$
and *nullable0-satisfies0*: $Length\ \mathfrak{A} = 0 \Longrightarrow nullable0\ a \longleftrightarrow \mathfrak{A} \models_0 a$
and *satisfies0-eqI*: $wf0\ (\#_V\ \mathfrak{B})\ a \Longrightarrow \#_V\ \mathfrak{A} = \#_V\ \mathfrak{B} \Longrightarrow left\text{-}formula0\ a \Longrightarrow$
$(\bigwedge m\ k.\ LESS\ k\ m\ (\#_V\ \mathfrak{B}) \Longrightarrow m^{\mathfrak{A}}k = m^{\mathfrak{B}}k) \Longrightarrow \mathfrak{A} \models_0 a \longleftrightarrow \mathfrak{B} \models_0 a$
and *wf-lderiv0*: $wf0\ idx\ a \Longrightarrow wf\ idx\ (lderiv0\ x\ a)$
and *left-formula-lderiv0*: $left\text{-}formula0\ a \Longrightarrow left\text{-}formula\ (lderiv0\ x\ a)$
and *wf-rderiv0*: $wf0\ idx\ a \Longrightarrow wf\ idx\ (rderiv0\ x\ a)$
and *satisfies-lderiv0*: $[\![wf0\ (\#_V\ \mathfrak{A})\ a;\ \#_V\ \mathfrak{A} = size\ x]\!] \Longrightarrow \mathfrak{A} \models lderiv0\ x\ a \longleftrightarrow CONS\ x\ \mathfrak{A} \models_0 a$

and *satisfies-bounded-lderiv0*: $[\![wf0\ (\#_V\ \mathfrak{A})\ a;\ \#_V\ \mathfrak{A} = size\ x]\!] \Longrightarrow \mathfrak{A} \models_b lderiv0\ x\ a \longleftrightarrow CONS\ x\ \mathfrak{A} \models_0 a$
and *satisfies-bounded-rderiv0*: $[\![wf0\ (\#_V\ \mathfrak{A})\ a;\ \#_V\ \mathfrak{A} = size\ x]\!] \Longrightarrow \mathfrak{A} \models_b rderiv0\ x\ a \longleftrightarrow SNOC\ x\ \mathfrak{A} \models_0 a$
and *find0-FV0*: $find0\ k\ l\ a \longleftrightarrow (k,\ l) \in set\ (FV0\ a)$
and *distinct-FV0*: $distinct\ (FV0\ a)$
and *wf0-FV0-LESS*: $[\![wf0\ idx\ a;\ (k,\ v) \in set\ (FV0\ a)]\!] \Longrightarrow LESS\ k\ v\ idx$
and *restrict-Restrict*: $i^{\mathfrak{A}}k = P \Longrightarrow restrict\ k\ P \longleftrightarrow \mathfrak{A} \models Restrict\ k\ i$
and *restrict-Restrict$_b$*: $i^{\mathfrak{A}}k = P \Longrightarrow restrict\ k\ P \longleftrightarrow \mathfrak{A} \models_b Restrict\ k\ i$
and *wf-Restrict*: $LESS\ k\ i\ idx \Longrightarrow wf\ idx\ (Restrict\ k\ i)$
and *left-formula-Restrict*: $left\text{-}formula\ (Restrict\ k\ i)$
and *finite-lderiv0*: $finite\ \{fold\ lderiv\ xs\ (FBase\ a) \mid xs.\ True\}$
and *finite-rderiv0*: $finite\ \{fold\ rderiv\ xs\ (FBase\ a) \mid xs.\ True\}$

locale *Word-Formula* = *Formula*
where $TYPEVARS = TYPEVARS$
for $TYPEVARS :: 'a :: linorder \times 'i \times 'k :: linorder \times 'n \times 'x \times 'v +$
fixes $enc :: 'i \Rightarrow 'x\ list$
and $alphabet :: 'n \Rightarrow 'x\ list$
and $ZERO :: 'n$

assumes *enc-inj*: $\#_V\ \mathfrak{A} = \#_V\ \mathfrak{B} \Longrightarrow enc\ \mathfrak{A} = enc\ \mathfrak{B} \longleftrightarrow \mathfrak{A} = \mathfrak{B}$
and *length-enc*: $length\ (enc\ \mathfrak{A}) = Length\ \mathfrak{A}$
and *enc-CONS*: $\#_V\ \mathfrak{A} = size\ x \Longrightarrow enc\ (CONS\ x\ \mathfrak{A}) = x\ \#\ enc\ \mathfrak{A}$
and *in-set-encD*: $x \in set\ (enc\ \mathfrak{A}) \Longrightarrow size\ x = \#_V\ \mathfrak{A}$
and *alphabet-size*: $x \in set\ (alphabet\ idx) \longleftrightarrow size\ x = idx$

context *Formula*
begin

lemma *satisfies-eqI*:
$[\![wf\ (\#_V\ \mathfrak{A})\ \varphi;\ \#_V\ \mathfrak{A} = \#_V\ \mathfrak{B};\ \bigwedge m\ k.\ LESS\ k\ m\ (\#_V\ \mathfrak{A}) \Longrightarrow m^{\mathfrak{A}}k = m^{\mathfrak{B}}k;\ left\text{-}formula\ \varphi]\!] \Longrightarrow$
$\mathfrak{A} \models \varphi \longleftrightarrow \mathfrak{B} \models \varphi$
proof (*induct* φ *arbitrary*: $\mathfrak{A}\ \mathfrak{B}$)
case $(FEx\ k\ \varphi)$
from *FEx.prems* **have** $\bigwedge P.\ (Extend\ k\ 0\ \mathfrak{A}\ P \models \varphi) \longleftrightarrow (Extend\ k\ 0\ \mathfrak{B}\ P \models \varphi)$
by (*intro FEx.hyps*) (*auto simp*: *nvars-Extend assigns-Extend dec-def gr0-conv-Suc LEQ-0 LESS-SUC*)
then show *?case* **by** *simp*
next
case $(FAll\ k\ \varphi)$
from *FAll.prems* **have** $\bigwedge P.\ (Extend\ k\ 0\ \mathfrak{A}\ P \models \varphi) \longleftrightarrow (Extend\ k\ 0\ \mathfrak{B}\ P \models \varphi)$
by (*intro FAll.hyps*) (*auto simp*: *nvars-Extend assigns-Extend dec-def gr0-conv-Suc LEQ-0 LESS-SUC*)
then show *?case* **by** *simp*
next
case $(FNot\ \varphi)$
from *FNot.prems* **have** $(\mathfrak{A} \models \varphi) \longleftrightarrow (\mathfrak{B} \models \varphi)$ **by** (*intro FNot.hyps*) *simp-all*

then show *?case* **by** *simp*
qed (*auto dest*: *satisfies0-eqI*)

lemma *wf-decr*:
$[\![$*wf* (*SUC k idx*) φ; *LEQ k l idx*; $\neg$ *find k l* $\varphi]\!] \Longrightarrow$ *wf idx* (*decr k l* φ)
by (*induct* φ *arbitrary*: *idx l*) (*auto simp*: *wf0-decr0 LESS-SUC SUC-SUC*)

lemma *left-formula-decr*:
left-formula $\varphi \Longrightarrow$ *left-formula* (*decr k l* φ)
by (*induct* φ *arbitrary*: *l*) (*auto simp*: *left-formula0-decr0*)

lemma *Extend-satisfies-decr*:
$[\![\neg$ *find k i* φ; *LEQ k i* ($\#_V\ \mathfrak{A}$)$]\!] \Longrightarrow$ *Extend k i* $\mathfrak{A}$ *P* $\models \varphi \longleftrightarrow \mathfrak{A} \models$ *decr k i* φ
by (*induct* φ *arbitrary*: *i* $\mathfrak{A}$)
(*auto simp*: *Extend-commute-unsafe*[*of - k 0 - - P*] *Extend-commute-safe*
Extend-satisfies0 nvars-Extend LESS-SUC SUC-SUC split: *bool.splits*)

lemma *LEQ-SUC*: $k \neq k' \Longrightarrow$ *LEQ k i* (*SUC k' idx*) = *LEQ k i idx*
by (*metis LESS-SUC*(*2*) *SUC-SUC*)

lemma *Extend-satisfies-bounded-decr*:
$[\![\neg$ *find k i* φ; *LEQ k i* ($\#_V\ \mathfrak{A}$); *len P* $\leq$ *Length* $\mathfrak{A}]\!] \Longrightarrow$
Extend k i $\mathfrak{A}$ *P* $\models_b \varphi \longleftrightarrow \mathfrak{A} \models_b$ *decr k i* φ
proof (*induct* φ *arbitrary*: *i* $\mathfrak{A}$ *P*)
case (*FEx k'* φ)
show *?case*
proof (*cases* $k = k'$)
case *True*
with *FEx*(*2*,*3*,*4*) **show** *?thesis*
using *FEx*(*1*)[*of Suc i Extend k' 0* $\mathfrak{A}$ *Q P* **for** *Q j*]
by (*auto simp*: *Extend-commute-safe LESS-SUC Length-Extend nvars-Extend*
max-def)
next
case *False*
with *FEx*(*2*,*3*,*4*) **show** *?thesis*
using *FEx*(*1*)[*of i Extend k' j* $\mathfrak{A}$ *Q P* **for** *Q j*]
by (*auto simp*: *Extend-commute-unsafe LEQ-SUC Length-Extend nvars-Extend*
max-def)
qed
next
case (*FAll k'* φ) **show** *?case*
proof (*cases* $k = k'$)
case *True*
with *FAll*(*2*,*3*,*4*) **show** *?thesis*
using *FAll*(*1*)[*of Suc i Extend k' 0* $\mathfrak{A}$ *Q P* **for** *Q j*]
by (*auto simp*: *Extend-commute-safe LESS-SUC Length-Extend nvars-Extend*
max-def)
next
case *False*

with *FAll*(*2*,*3*,*4*) **show** *?thesis*
using *FAll*(*1*)[*of i Extend* k' *j* $\mathfrak{A}$ *Q P* **for** *Q j*]
by (*auto simp*: *Extend-commute-unsafe LEQ-SUC Length-Extend nvars-Extend max-def*)
qed
qed (*auto simp*: *Extend-satisfies0 split*: *bool.splits*)

4 Normalization

lemma *wf-nFOr*:
wf idx (*FOr* φ ψ) $\Longrightarrow$ *wf idx* (*nFOr* φ ψ)
by (*induct* φ ψ *rule*: *nFOr.induct*) (*simp-all add*: *Let-def*)

lemma *wf-nFAnd*:
wf idx (*FAnd* φ ψ) $\Longrightarrow$ *wf idx* (*nFAnd* φ ψ)
by (*induct* φ ψ *rule*: *nFAnd.induct*) (*simp-all add*: *Let-def*)

lemma *wf-nFNot*:
wf idx (*FNot* φ) $\Longrightarrow$ *wf idx* (*nFNot* φ)
by (*induct* φ *arbitrary*: *idx rule*: *nFNot.induct*) (*auto simp*: *wf-nFOr wf-nFAnd*)

lemma *wf-nFEx*:
wf idx (*FEx b* φ) $\Longrightarrow$ *wf idx* (*nFEx b* φ)
by (*induct* φ *arbitrary*: *idx rule*: *nFEx.induct*)
(*auto simp*: *SUC-SUC LEQ-0 LESS-SUC*(*1*) *gr0-conv-Suc wf-nFOr intro*: *wf0-decr0 wf-decr*)

lemma *wf-nFAll*:
wf idx (*FAll b* φ) $\Longrightarrow$ *wf idx* (*nFAll b* φ)
by (*induct* φ *arbitrary*: *idx rule*: *nFAll.induct*)
(*auto simp*: *SUC-SUC LEQ-0 LESS-SUC*(*1*) *gr0-conv-Suc wf-nFAnd intro*: *wf0-decr0 wf-decr*)

lemma *wf-norm*: *wf idx* φ $\Longrightarrow$ *wf idx* (*norm* φ)
by (*induct* φ *arbitrary*: *idx*) (*simp-all add*: *wf-nFOr wf-nFAnd wf-nFNot wf-nFEx wf-nFAll*)

lemma *left-formula-nFOr*:
left-formula (*FOr* φ ψ) $\Longrightarrow$ *left-formula* (*nFOr* φ ψ)
by (*induct* φ ψ *rule*: *nFOr.induct*) (*simp-all add*: *Let-def*)

lemma *left-formula-nFAnd*:
left-formula (*FAnd* φ ψ) $\Longrightarrow$ *left-formula* (*nFAnd* φ ψ)
by (*induct* φ ψ *rule*: *nFAnd.induct*) (*simp-all add*: *Let-def*)

lemma *left-formula-nFNot*:
left-formula (*FNot* φ) $\Longrightarrow$ *left-formula* (*nFNot* φ)
by (*induct* φ *rule*: *nFNot.induct*) (*auto simp*: *left-formula-nFOr left-formula-nFAnd*)

lemma *left-formula-nFEx*:
  left-formula $(FEx\ b\ \varphi) \Longrightarrow$ *left-formula* $(nFEx\ b\ \varphi)$
  by (*induct* φ *rule*: *nFEx.induct*)
    (*auto simp*: *left-formula-nFOr left-formula0-decr0 left-formula-decr*)

lemma *left-formula-nFAll*:
  left-formula $(FAll\ b\ \varphi) \Longrightarrow$ *left-formula* $(nFAll\ b\ \varphi)$
  by (*induct* φ *rule*: *nFAll.induct*)
    (*auto simp*: *left-formula-nFAnd left-formula0-decr0 left-formula-decr*)

lemma *left-formula-norm*: *left-formula* $\varphi \Longrightarrow$ *left-formula* $(norm\ \varphi)$
  by (*induct* φ) (*simp-all add*: *left-formula-nFOr left-formula-nFAnd left-formula-nFNot*
    left-formula-nFEx left-formula-nFAll)

lemma *satisfies-nFOr*:
  $\mathfrak{A} \models nFOr\ \varphi\ \psi \longleftrightarrow \mathfrak{A} \models FOr\ \varphi\ \psi$
  by (*induct* $\varphi\ \psi$ *arbitrary*: $\mathfrak{A}$ *rule*: *nFOr.induct*) *auto*

lemma *satisfies-nFAnd*:
  $\mathfrak{A} \models nFAnd\ \varphi\ \psi \longleftrightarrow \mathfrak{A} \models FAnd\ \varphi\ \psi$
  by (*induct* $\varphi\ \psi$ *arbitrary*: $\mathfrak{A}$ *rule*: *nFAnd.induct*) *auto*

lemma *satisfies-nFNot*:
  $\mathfrak{A} \models nFNot\ \varphi \longleftrightarrow \mathfrak{A} \models FNot\ \varphi$
  by (*induct* φ *arbitrary*: $\mathfrak{A}$)
   (*auto simp*: *satisfies-nFOr satisfies-nFAnd*)

lemma *satisfies-nFEx*: $\mathfrak{A} \models nFEx\ b\ \varphi \longleftrightarrow \mathfrak{A} \models FEx\ b\ \varphi$
  by (*induct* φ *rule*: *nFEx.induct*)
    (*auto simp add*: *satisfies-nFOr Extend-satisfies-decr*
      LEQ-0 LESS-SUC(1) nvars-Extend Extend-satisfies0 Extend-commute-safe
Extend-commute-unsafe)

lemma *satisfies-nFAll*: $\mathfrak{A} \models nFAll\ b\ \varphi \longleftrightarrow \mathfrak{A} \models FAll\ b\ \varphi$
  by (*induct* φ *rule*: *nFAll.induct*)
    (*auto simp add*: *satisfies-nFAnd Extend-satisfies-decr*
      Extend-satisfies0 LEQ-0 LESS-SUC(1) nvars-Extend Extend-commute-safe
Extend-commute-unsafe)

lemma *satisfies-norm*: $\mathfrak{A} \models norm\ \varphi \longleftrightarrow \mathfrak{A} \models \varphi$
  using *satisfies-nFOr satisfies-nFAnd satisfies-nFNot satisfies-nFEx satisfies-nFAll*
  by (*induct* φ *arbitrary*: $\mathfrak{A}$) *simp-all*

lemma *satisfies-bounded-nFOr*:
  $\mathfrak{A} \models_b nFOr\ \varphi\ \psi \longleftrightarrow \mathfrak{A} \models_b FOr\ \varphi\ \psi$
  by (*induct* $\varphi\ \psi$ *arbitrary*: $\mathfrak{A}$ *rule*: *nFOr.induct*) *auto*

lemma *satisfies-bounded-nFAnd*:
  $\mathfrak{A} \models_b nFAnd\ \varphi\ \psi \longleftrightarrow \mathfrak{A} \models_b FAnd\ \varphi\ \psi$

by (*induct* φ ψ *arbitrary*: $\mathfrak{A}$ *rule*: *nFAnd.induct*) *auto*

lemma *satisfies-bounded-nFNot*:
$\mathfrak{A} \models_b$ *nFNot* $\varphi \longleftrightarrow \mathfrak{A} \models_b$ *FNot* φ
by (*induct* φ *arbitrary*: $\mathfrak{A}$)
(*auto simp*: *satisfies-bounded-nFOr satisfies-bounded-nFAnd*)

lemma *len-cut-0*: *len* (*cut 0 P*) = *0*
by (*metis le-0-eq len-cut-le*)

lemma *satisfies-bounded-nFEx*: $\mathfrak{A} \models_b$ *nFEx b* $\varphi \longleftrightarrow \mathfrak{A} \models_b$ *FEx b* φ
by (*induct* φ *rule*: *nFEx.induct*)
(*auto 4 4 simp add*: *satisfies-bounded-nFOr Extend-satisfies-bounded-decr*
LEQ-0 LESS-SUC(*1*) *nvars-Extend Length-Extend len-cut-0*
Extend-satisfies0 Extend-commute-safe Extend-commute-unsafe cong: *ex-cong*
split: *bool.splits*
intro: *exI*[**where** $P = \lambda x.\ P\ x \wedge Q\ x$ **for** *P Q*, *OF conjI*[*rotated*]] *exI*[*of* -
cut 0 P **for** *P*])

lemma *satisfies-bounded-nFAll*: $\mathfrak{A} \models_b$ *nFAll b* $\varphi \longleftrightarrow \mathfrak{A} \models_b$ *FAll b* φ
by (*induct* φ *rule*: *nFAll.induct*)
(*auto 4 4 simp add*: *satisfies-bounded-nFAnd Extend-satisfies-bounded-decr*
LEQ-0 LESS-SUC(*1*) *nvars-Extend Length-Extend len-cut-0*
Extend-satisfies0 Extend-commute-safe Extend-commute-unsafe cong: *split*:
bool.splits
intro: *exI*[**where** $P = \lambda x.\ P\ x \wedge Q\ x$ **for** *P Q*, *OF conjI*[*rotated*]] *dest*: *spec*[*of*
- *cut 0 P* **for** *P*])

lemma *satisfies-bounded-norm*: $\mathfrak{A} \models_b$ *norm* $\varphi \longleftrightarrow \mathfrak{A} \models_b \varphi$
by (*induct* φ *arbitrary*: $\mathfrak{A}$)
(*simp-all add*: *satisfies-bounded-nFOr satisfies-bounded-nFAnd*
satisfies-bounded-nFNot satisfies-bounded-nFEx satisfies-bounded-nFAll)

5 Derivatives of Formulas

lemma *wf-lderiv*:
wf idx $\varphi \Longrightarrow$ *wf idx* (*lderiv x* φ)
by (*induct* φ *arbitrary*: *x idx*) (*auto simp*: *wf-lderiv0*)

lemma *left-formula-lderiv*:
left-formula $\varphi \Longrightarrow$ *left-formula* (*lderiv x* φ)
by (*induct* φ *arbitrary*: *x*) (*auto simp*: *left-formula-lderiv0*)

lemma *wf-rderiv*:
wf idx $\varphi \Longrightarrow$ *wf idx* (*rderiv x* φ)
by (*induct* φ *arbitrary*: *x idx*) (*auto simp*: *wf-rderiv0*)

theorem *satisfies-lderiv*: $[\![$*wf* ($\#_V\ \mathfrak{A}$) φ; $\#_V\ \mathfrak{A}$ = *size x*$]\!] \Longrightarrow \mathfrak{A} \models$ *lderiv x* $\varphi \longleftrightarrow$
CONS x $\mathfrak{A} \models \varphi$

proof (*induct* φ *arbitrary*: x $\mathfrak{A}$)
  case (*FEx k* φ)
  from *FEx.prems FEx.hyps*[*of Extend k 0* $\mathfrak{A}$ *P extend k b x* **for** *P b*] **show** *?case*
    by (*auto simp*: *nvars-Extend size-extend Extend-CONS*
      downshift-upshift eval-add eval-upshift downshift-add-zero
      intro: *exI*[*of - add 0* (*upshift P*) **for** *P*] *exI*[*of - upshift P* **for** *P*])
next
  case (*FAll k* φ)
  from *FAll.prems FAll.hyps*[*of Extend k 0* $\mathfrak{A}$ *P extend k b x* **for** *P b*] **show** *?case*
    by (*auto simp*: *nvars-Extend size-extend Extend-CONS*
      downshift-upshift eval-add eval-upshift downshift-add-zero
      dest: *spec*[*of - add 0* (*upshift P*) **for** *P*] *spec*[*of - upshift P* **for** *P*])
qed (*simp-all add*: *satisfies-lderiv0 split*: *bool.splits*)

theorem *satisfies-bounded-lderiv*: $[\![wf\ (\#_V\ \mathfrak{A})\ \varphi;\ \#_V\ \mathfrak{A} = size\ x]\!] \Longrightarrow \mathfrak{A} \models_b lderiv\ x\ \varphi \longleftrightarrow CONS\ x\ \mathfrak{A} \models_b \varphi$
proof (*induct* φ *arbitrary*: x $\mathfrak{A}$)
  case (*FEx k* φ)
  note [*simp*] = *nvars-Extend size-extend Extend-CONS Length-CONS*
    downshift-upshift eval-add eval-upshift downshift-add-zero len-add len-upshift len-downshift
  from *FEx.prems FEx.hyps*[*of Extend k 0* $\mathfrak{A}$ *P extend k b x* **for** *P b*] **show** *?case*
    by *auto* (*force intro*: *exI*[*of - add 0* (*upshift P*) **for** *P*] *exI*[*of - upshift P* **for** *P*] *split*: *nat.splits*)+
next
  case (*FAll k* φ)
  note [*simp*] = *nvars-Extend size-extend Extend-CONS Length-CONS*
    downshift-upshift eval-add eval-upshift downshift-add-zero len-add len-upshift len-downshift
  from *FAll.prems FAll.hyps*[*of Extend k 0* $\mathfrak{A}$ *P extend k b x* **for** *P b*] **show** *?case*
    by *auto* (*force dest*: *spec*[*of - add 0* (*upshift P*) **for** *P*] *spec*[*of - upshift P* **for** *P*] *split*: *nat.splits*)+
qed (*simp-all add*: *satisfies-bounded-lderiv0 split*: *bool.splits*)

theorem *satisfies-bounded-rderiv*:
  $[\![wf\ (\#_V\ \mathfrak{A})\ \varphi;\ \#_V\ \mathfrak{A} = size\ x]\!] \Longrightarrow \mathfrak{A} \models_b rderiv\ x\ \varphi \longleftrightarrow SNOC\ x\ \mathfrak{A} \models_b \varphi$
proof (*induct* φ *arbitrary*: x $\mathfrak{A}$)
  case (*FEx k* φ)
  from *FEx.prems FEx.hyps*[*of Extend k 0* $\mathfrak{A}$ *P extend k b x* **for** *P b*] **show** *?case*
    by (*auto simp*: *nvars-Extend size-extend Extend-SNOC-cut len-cut-le eval-ge-len*

      eval-add cut-add Length-SNOC len-add len-cut le-Suc-eq max-def
      intro: *exI*[*of - cut* (*Length* $\mathfrak{A}$) *P* **for** *P*] *exI*[*of - add* (*Length* $\mathfrak{A}$) *P* **for** *P*] *split*: *if-splits*)
next
  case (*FAll k* φ)
  from *FAll.prems FAll.hyps*[*of Extend k 0* $\mathfrak{A}$ *P extend k b x* **for** *P b*] **show** *?case*
    by (*auto simp*: *nvars-Extend size-extend Extend-SNOC-cut len-cut-le eval-ge-len*

eval-add cut-add Length-SNOC len-add len-cut le-Suc-eq max-def
dest: *spec*[*of* - *cut* (*Length* $\mathfrak{A}$) *P* **for** *P*] *spec*[*of* - *add* (*Length* $\mathfrak{A}$) *P* **for** *P*]
split: *if-splits*)
qed (*simp-all add*: *satisfies-bounded-rderiv0 split*: *bool.splits*)

lemma *wf-norm-rderivs*: *wf idx* $\varphi \Longrightarrow$ *wf idx* (((*norm* $\circ$ *rderiv* (*zero idx*)) ^^ *k*) φ)
by (*induct k*) (*auto simp*: *wf-norm wf-rderiv*)

6 Finiteness of Derivatives Modulo ACI

lemma *finite-fold-deriv*:
assumes *d0* = *lderiv0* $\vee$ *d0* = *rderiv0*
shows *finite* {⟨*fold* (*deriv d0*) *xs* φ⟩ | *xs*. *True*}
proof (*induct* φ)
case (*FBase a*) **then show** *?case* **using** *assms*
by (*auto intro*:
finite-subset[*OF* - *finite-imageI*[*OF finite-lderiv0*]]
finite-subset[*OF* - *finite-imageI*[*OF finite-rderiv0*]])
next
case (*FNot* φ)
then show *?case*
by (*auto simp*: *fold-deriv-FNot intro*: *finite-surj*[*OF FNot*])
next
case (*FOr* φ ψ)
then show *?case*
by (*auto simp*: *fold-deriv-FOr intro*!: *finite-surj*[*OF finite-cartesian-product*[*OF FOr*]])
next
case (*FAnd* φ ψ)
then show *?case*
by (*auto simp*: *fold-deriv-FAnd intro*!: *finite-surj*[*OF finite-cartesian-product*[*OF FAnd*]])
next
case (*FEx k* φ)
then have *finite* ($\bigcup$ (*disjuncts* ‘ {⟨*fold* (*deriv d0*) *xs* φ⟩ | *xs* . *True*}))
by (*auto simp*: *finite-juncts*)
then have *finite* ($\bigcup$*xs*. *disjuncts* ⟨*fold* (*deriv d0*) *xs* φ⟩)
by (*rule finite-subset*[*rotated*]) *auto*
then have *finite* {*FEx k* ψ | ψ. *nf-ACI* ψ $\wedge$ *disjuncts* $\psi \subseteq$ ($\bigcup$*xs*. *disjuncts* ⟨*fold* (*deriv d0*) *xs* φ⟩)}
by (*rule finite-norm-ACI-juncts*)
then show *?case*
by (*rule finite-subset*[*OF fold-deriv-FEx*])
next
case (*FAll k* φ)
then have *finite* ($\bigcup$ (*conjuncts* ‘ {⟨*fold* (*deriv d0*) *xs* φ⟩ | *xs* . *True*}))
by (*auto simp*: *finite-juncts*)
then have *finite* ($\bigcup$*xs*. *conjuncts* ⟨*fold* (*deriv d0*) *xs* φ⟩)

by (*rule finite-subset*[*rotated*]) *auto*
then have *finite* $\{FAll\ k\ \psi \mid \psi.\ nf\text{-}ACI\ \psi \wedge conjuncts\ \psi \subseteq (\bigcup xs.\ conjuncts\ \langle fold\ (deriv\ d0)\ xs\ \varphi\rangle)\}$
by (*rule finite-norm-ACI-juncts*)
then show *?case*
by (*rule finite-subset*[*OF fold-deriv-FAll*])
qed (*simp add*: *fold-deriv-FBool*)

theorem
finite-fold-lderiv: *finite* $\{\langle fold\ lderiv\ xs\ \langle\varphi\rangle\rangle \mid xs.\ True\}$ **and**
finite-fold-rderiv: *finite* $\{\langle fold\ rderiv\ xs\ \langle\varphi\rangle\rangle \mid xs.\ True\}$
by (*blast intro*: *nf-ACI-norm-ACI finite-fold-deriv*)+

lemma *wf-nFOR*: $wf\ idx\ (nFOR\ \varphi s) \longleftrightarrow (\forall \varphi \in set\ \varphi s.\ wf\ idx\ \varphi)$
by (*induct rule*: *nFOR.induct*) *auto*

lemma *wf-nFAND*: $wf\ idx\ (nFAND\ \varphi s) \longleftrightarrow (\forall \varphi \in set\ \varphi s.\ wf\ idx\ \varphi)$
by (*induct rule*: *nFAND.induct*) *auto*

lemma *wf-NFOR*: $finite\ \Phi \Longrightarrow wf\ idx\ (NFOR\ \Phi) \longleftrightarrow (\forall \varphi \in \Phi.\ wf\ idx\ \varphi)$
unfolding *NFOR-def o-apply* **by** (*auto simp*: *wf-nFOR*)

lemma *wf-NFAND*: $finite\ \Phi \Longrightarrow wf\ idx\ (NFAND\ \Phi) \longleftrightarrow (\forall \varphi \in \Phi.\ wf\ idx\ \varphi)$
unfolding *NFAND-def o-apply* **by** (*auto simp*: *wf-nFAND*)

lemma *satisfies-bounded-nFOR*: $\mathfrak{A} \models_b nFOR\ \varphi s \longleftrightarrow (\exists \varphi \in set\ \varphi s.\ \mathfrak{A} \models_b \varphi)$
by (*induct rule*: *nFOR.induct*) (*auto simp*: *satisfies-bounded-nFOr*)

lemma *satisfies-bounded-nFAND*: $\mathfrak{A} \models_b nFAND\ \varphi s \longleftrightarrow (\forall \varphi \in set\ \varphi s.\ \mathfrak{A} \models_b \varphi)$
by (*induct rule*: *nFAND.induct*) (*auto simp*: *satisfies-bounded-nFAnd*)

lemma *satisfies-bounded-NFOR*: $finite\ \Phi \Longrightarrow \mathfrak{A} \models_b NFOR\ \Phi \longleftrightarrow (\exists \varphi \in \Phi.\ \mathfrak{A} \models_b \varphi)$
unfolding *NFOR-def o-apply* **by** (*auto simp*: *satisfies-bounded-nFOR*)

lemma *satisfies-bounded-NFAND*: $finite\ \Phi \Longrightarrow \mathfrak{A} \models_b NFAND\ \Phi \longleftrightarrow (\forall \varphi \in \Phi.\ \mathfrak{A} \models_b \varphi)$
unfolding *NFAND-def o-apply* **by** (*auto simp*: *satisfies-bounded-nFAND*)

lemma *wf-juncts*:
$wf\ idx\ \varphi \longleftrightarrow (\forall \psi \in disjuncts\ \varphi.\ wf\ idx\ \psi)$
$wf\ idx\ \varphi \longleftrightarrow (\forall \psi \in conjuncts\ \varphi.\ wf\ idx\ \psi)$
by (*induct* φ) *auto*

lemma *wf-norm-ACI*: $wf\ idx\ \langle\varphi\rangle = wf\ idx\ \varphi$
by (*induct* φ *arbitrary*: *idx*)
(*auto simp*: *finite-juncts wf-NFOR wf-NFAND ball-Un wf-juncts*[*symmetric*])

lemma *satisfies-bounded-disjuncts*:

$\mathfrak{A} \models_b \varphi \longleftrightarrow (\exists \psi \in disjuncts\ \varphi.\ \mathfrak{A} \models_b \psi)$
by (*induct* φ *arbitrary*: $\mathfrak{A}$) *auto*

lemma *satisfies-bounded-conjuncts*:
$\mathfrak{A} \models_b \varphi \longleftrightarrow (\forall \psi \in conjuncts\ \varphi.\ \mathfrak{A} \models_b \psi)$
by (*induct* φ *arbitrary*: $\mathfrak{A}$) *auto*

lemma *satisfies-bounded-norm-ACI*: $\mathfrak{A} \models_b \langle\varphi\rangle \longleftrightarrow \mathfrak{A} \models_b \varphi$
by (*rule sym*, *induct* φ *arbitrary*: $\mathfrak{A}$)
(*auto simp*: *satisfies-bounded-NFOR satisfies-bounded-NFAND finite-juncts*
intro: *iffD2*[*OF satisfies-bounded-disjuncts*] *iffD2*[*OF satisfies-bounded-conjuncts*]
dest: *iffD1*[*OF satisfies-bounded-disjuncts*] *iffD1*[*OF satisfies-bounded-conjuncts*])

lemma *nvars-SNOCs*: $\#_V\ ((SNOC\ x\hat{}\hat{}k)\ \mathfrak{A}) = \#_V\ \mathfrak{A}$
by (*induct k*) (*auto simp*: *nvars-SNOC*)

lemma *wf-fold-rderiv*: $wf\ idx\ \varphi \Longrightarrow wf\ idx\ (fold\ rderiv\ (replicate\ k\ x)\ \varphi)$
by (*induct k arbitrary*: φ) (*auto simp*: *wf-rderiv*)

lemma *satisfies-bounded-fold-rderiv*:
$[\![wf\ idx\ \varphi;\ \#_V\ \mathfrak{A} = idx;\ size\ x = idx]\!] \Longrightarrow$
$\mathfrak{A} \models_b fold\ rderiv\ (replicate\ k\ x)\ \varphi \longleftrightarrow (SNOC\ x\hat{}\hat{}k)\ \mathfrak{A} \models_b \varphi$
by (*induct k arbitrary*: $\mathfrak{A}\ \varphi$) (*auto simp*: *satisfies-bounded-rderiv wf-rderiv nvars-SNOCs*)

7 Emptiness Check

context
fixes b :: *bool*
and *idx* :: $'n$
and ψ :: $('a, 'k)$ *aformula*
begin

abbreviation *fut-test* $\equiv \lambda(\varphi, \Phi).\ \varphi \notin set\ \Phi$
abbreviation *fut-step* $\equiv \lambda(\varphi, \Phi).\ (norm\ (rderiv\ (zero\ idx)\ \varphi),\ \varphi\ \#\ \Phi)$
definition *fut-derivs* $k\ \varphi \equiv ((norm\ o\ rderiv\ (zero\ idx))\hat{}\hat{}k)\ \varphi$

lemma *fut-derivs-Suc*[*simp*]: $norm\ (rderiv\ (zero\ idx)\ (fut\text{-}derivs\ k\ \varphi)) = fut\text{-}derivs$ $(Suc\ k)\ \varphi$
unfolding *fut-derivs-def* **by** *auto*

definition *fut-invariant* =
$(\lambda(\varphi, \Phi).\ wf\ idx\ \varphi \land (\forall \varphi \in set\ \Phi.\ wf\ idx\ \varphi) \land$
$(\exists k.\ \varphi = fut\text{-}derivs\ k\ \psi \land \Phi = map\ (\lambda i.\ fut\text{-}derivs\ i\ \psi)\ (rev\ [0\ ..< k])))$
definition *fut-spec* $\varphi\Phi \equiv (\forall \varphi \in set\ (snd\ \varphi\Phi).\ wf\ idx\ \varphi) \land$
$(\forall \mathfrak{A}.\ \#_V\ \mathfrak{A} = idx \longrightarrow$
$(if\ b\ then\ (\exists k.\ (SNOC\ (zero\ idx)\ \hat{}\hat{}\ k)\ \mathfrak{A} \models_b \psi) \longleftrightarrow (\exists \varphi \in set\ (snd\ \varphi\Phi).\ \mathfrak{A}$ $\models_b \varphi)$
$else\ (\forall k.\ (SNOC\ (zero\ idx)\ \hat{}\hat{}\ k)\ \mathfrak{A} \models_b \psi) \longleftrightarrow (\forall \varphi \in set\ (snd\ \varphi\Phi).\ \mathfrak{A} \models_b$ $\varphi)))$

definition *fut-default* =
  (ψ, *sorted-list-of-set* $\{\langle$*fold rderiv* (*replicate k* (*zero idx*)) $\langle\psi\rangle\rangle$ | *k*. *True*$\}$)

lemma *finite-fold-rderiv-zeros*: *finite* $\{\langle$*fold rderiv* (*replicate k* (*zero idx*)) $\langle\psi\rangle\rangle$ | *k*. *True*$\}$
  by (*rule finite-subset*[*OF - finite-fold-rderiv*[*of* ψ]]) *blast*

definition *fut* :: ('*a*, '*k*) *aformula* **where**
  fut = (*if b then nFOR else nFAND*) (*snd* (*while-default fut-default fut-test fut-step* (ψ, [])))

context
  assumes *wf*: *wf idx* ψ
begin

lemma *wf-fut-derivs*:
  wf idx (*fut-derivs k* ψ)
  by (*induct k*) (*auto simp*: *wf-norm wf-rderiv wf fut-derivs-def*)

lemma *satisfies-bounded-fut-derivs*:
  $\#_V\ \mathfrak{A} = idx \Longrightarrow \mathfrak{A} \models_b$ *fut-derivs k* $\psi \longleftrightarrow$ (*SNOC* (*zero idx*) ^^*k*) $\mathfrak{A} \models_b \psi$
  by (*induct k arbitrary*: $\mathfrak{A}$) (*auto simp*: *fut-derivs-def satisfies-bounded-rderiv satisfies-bounded-norm*
    wf-norm-rderivs size-zero nvars-SNOC funpow-swap1[*of SNOC x* **for** *x*] *wf*)

lemma *fut-init*: *fut-invariant* (ψ, [])
  unfolding *fut-invariant-def* **by** (*auto simp*: *fut-derivs-def wf*)

lemma *fut-spec-default*: *fut-spec fut-default*
  using *satisfies-bounded-fold-rderiv*[*OF iffD2*[*OF wf-norm-ACI wf*] *sym size-zero*]

  unfolding *fut-spec-def fut-default-def snd-conv*
    conjunct1[*OF sorted-list-of-set*[*OF finite-fold-rderiv-zeros*]]
  by (*auto simp*: *satisfies-bounded-norm-ACI wf-fold-rderiv wf wf-norm-ACI simp del*: *fold-replicate*)

lemma *fut-invariant*: *fut-invariant* $\varphi\Phi \Longrightarrow$ *fut-test* $\varphi\Phi \Longrightarrow$ *fut-invariant* (*fut-step* $\varphi\Phi$)
  by (*cases* $\varphi\Phi$) (*auto simp*: *fut-invariant-def wf-norm wf-rderiv split*: *if-splits*)

lemma *fut-terminate*: *fut-invariant* $\varphi\Phi \Longrightarrow \neg$ *fut-test* $\varphi\Phi \Longrightarrow$ *fut-spec* $\varphi\Phi$
proof (*induct* $\varphi\Phi$, *unfold prod.case not-not*)
  fix φ Φ **assume** *fut-invariant* (φ, Φ) $\varphi \in$ *set* Φ
  then obtain *i k* **where** $i < k$ **and** φ-*def*: φ = *fut-derivs i* ψ
    and Φ-*def*: Φ = *map* (λi. *fut-derivs i* ψ) (*rev* [*0*..<*k*])
    and $*$: *fut-derivs k* ψ = *fut-derivs i* ψ **unfolding** *fut-invariant-def* **by** *auto*
  have *set* Φ = $\{$*fut-derivs k* ψ | *k* . *True*$\}$
  unfolding Φ-*def set-map set-rev set-upt* **proof** *safe*

fix j
show *fut-derivs* $j\ \psi \in (\lambda i.\ \textit{fut-derivs}\ i\ \psi)$ ‘ $\{0..<k\}$
proof (*cases* $j < k$)
case *False*
with $*$ ‹$i < k$› **have** *fut-derivs* $j\ \psi = \textit{fut-derivs}\ ((j - i)\ \textit{mod}\ (k - i) + i)\ \psi$
unfolding *fut-derivs-def* **by** (*auto intro*: *funpow-cycle-offset*)
then show *?thesis* **using** ‹$i < k$› ‹$\neg\ j < k$›
by (*metis image-eqI atLeastLessThan-iff le0 less-diff-conv mod-less-divisor zero-less-diff*)
qed *simp*
qed (*blast intro*: $*$)
then show *fut-spec* (φ, Φ)
unfolding *fut-spec-def* **using** *satisfies-bounded-fut-derivs* **by** (*auto simp*: *wf-fut-derivs*)
qed

lemma *fut-spec-while-default*:
fut-spec (*while-default fut-default fut-test fut-step* $(\psi, [])$)
using *fut-invariant fut-terminate fut-init fut-spec-default* **by** (*rule while-default-rule*)

lemma *wf-fut*: *wf idx fut*
using *fut-spec-while-default* **unfolding** *fut-def fut-spec-def* **by** (*auto simp*: *wf-nFOR wf-nFAND*)

lemma *satisfies-bounded-fut*:
assumes $\#_V\ \mathfrak{A} = \textit{idx}$
shows $\mathfrak{A} \models_b \textit{fut} \longleftrightarrow$
(*if* b *then* $(\exists\, k.\ (\textit{SNOC}\ (\textit{zero idx})\ \hat{}\hat{}\ k)\ \mathfrak{A} \models_b \psi)$ *else* $(\forall\, k.\ (\textit{SNOC}\ (\textit{zero idx})\ \hat{}\hat{}\ k)\ \mathfrak{A} \models_b \psi))$
using *fut-spec-while-default assms* **unfolding** *fut-def fut-spec-def*
by (*auto simp*: *satisfies-bounded-nFOR satisfies-bounded-nFAND*)

end

end

fun *finalize* :: $'n \Rightarrow ('a, 'k)$ *aformula* $\Rightarrow ('a, 'k)$ *aformula* **where**
finalize idx $(\textit{FEx}\ k\ \varphi) = \textit{fut True idx}\ (\textit{nFEx}\ k\ (\textit{finalize}\ (\textit{SUC}\ k\ \textit{idx})\ \varphi))$
| *finalize idx* $(\textit{FAll}\ k\ \varphi) = \textit{fut False idx}\ (\textit{nFAll}\ k\ (\textit{finalize}\ (\textit{SUC}\ k\ \textit{idx})\ \varphi))$
| *finalize idx* $(\textit{FOr}\ \varphi\ \psi) = \textit{FOr}\ (\textit{finalize idx}\ \varphi)\ (\textit{finalize idx}\ \psi)$
| *finalize idx* $(\textit{FAnd}\ \varphi\ \psi) = \textit{FAnd}\ (\textit{finalize idx}\ \varphi)\ (\textit{finalize idx}\ \psi)$
| *finalize idx* $(\textit{FNot}\ \varphi) = \textit{FNot}\ (\textit{finalize idx}\ \varphi)$
| *finalize idx* $\varphi = \varphi$

definition *final* :: $'n \Rightarrow ('a, 'k)$ *aformula* $\Rightarrow$ *bool* **where**
final idx = *nullable o finalize idx*

lemma *wf-finalize*: *wf idx* $\varphi \Longrightarrow$ *wf idx* (*finalize idx* φ)
by (*induct* φ *arbitrary*: *idx*) (*auto simp*: *wf-fut wf-nFEx wf-nFAll*)

lemma *Length-SNOCs*: *Length* $((SNOC\ x\ \hat{}\hat{}\ k)\ \mathfrak{A}) = Length\ \mathfrak{A} + k$
by (*induct k arbitrary*: $\mathfrak{A}$) (*auto simp*: *Length-SNOC*)

lemma *assigns-SNOCs-zero*:
$[\![LESS\ ord\ m\ (\#_V\ \mathfrak{A});\ \#_V\ \mathfrak{A} = idx]\!] \Longrightarrow m^{(SNOC\ (zero\ idx)\ \hat{}\hat{}\ k)\ \mathfrak{A}}{}_{ord} = m^{\mathfrak{A}}{}_{ord}$
by (*induct k arbitrary*: $\mathfrak{A}$) (*auto simp*: *assigns-SNOC-zero nvars-SNOC funpow-swap1*)

lemma *Extend-SNOCs-zero-satisfies*: $[\![wf\ (SUC\ ord\ idx)\ \varphi;\ \#_V\ \mathfrak{A} = idx;\ left\text{-}formula$
$\varphi]\!] \Longrightarrow$
$Extend\ ord\ 0\ ((SNOC\ (zero\ (\#_V\ \mathfrak{A}))\ \hat{}\hat{}\ k)\ \mathfrak{A})\ P \models \varphi \longleftrightarrow Extend\ ord\ 0\ \mathfrak{A}\ P \models$
φ
by (*rule satisfies-eqI*)
(*auto simp*: *nvars-Extend nvars-SNOCs assigns-Extend assigns-SNOCs-zero LEQ-0*
LESS-SUC
dec-def gr0-conv-Suc)

lemma *finalize-satisfies*: $[\![wf\ idx\ \varphi;\ \#_V\ \mathfrak{A} = idx;\ left\text{-}formula\ \varphi]\!] \Longrightarrow \mathfrak{A} \models_b finalize$
$idx\ \varphi \longleftrightarrow \mathfrak{A} \models \varphi$
by (*induct* φ *arbitrary*: *idx* $\mathfrak{A}$)
(*force simp add*: *wf-nFEx wf-nFAll wf-finalize Length-SNOCs nvars-Extend*
nvars-SNOCs
satisfies-bounded-fut satisfies-bounded-nFEx satisfies-bounded-nFAll Extend-SNOCs-zero-satisfies
intro: *le-add2*)+

lemma *Extend-empty-satisfies0*:
$[\![Length\ \mathfrak{A} = 0;\ len\ P = 0]\!] \Longrightarrow Extend\ k\ i\ \mathfrak{A}\ P \models_0 a \longleftrightarrow \mathfrak{A} \models_0 a$
by (*intro box-equals*[*OF - nullable0-satisfies0 nullable0-satisfies0*])
(*auto simp*: *nvars-Extend Length-Extend*)

lemma *Extend-empty-satisfies-bounded*:
$[\![Length\ \mathfrak{A} = 0;\ len\ P = 0]\!] \Longrightarrow Extend\ k\ 0\ \mathfrak{A}\ P \models_b \varphi \longleftrightarrow \mathfrak{A} \models_b \varphi$
by (*induct* φ *arbitrary*: $k\ \mathfrak{A}\ P$)
(*auto simp*: *Extend-empty-satisfies0 Length-Extend split*: *bool.splits*)

lemma *nullable-satisfies-bounded*: $Length\ \mathfrak{A} = 0 \Longrightarrow nullable\ \varphi \longleftrightarrow \mathfrak{A} \models_b \varphi$
by (*induct* φ) (*auto simp*: *nullable0-satisfies0 Extend-empty-satisfies-bounded*
len-cut-0
intro: *exI*[*of - cut 0 P* **for** *P*])

lemma *final-satisfies*:
$[\![wf\ idx\ \varphi \wedge left\text{-}formula\ \varphi;\ Length\ \mathfrak{A} = 0;\ \#_V\ \mathfrak{A} = idx]\!] \Longrightarrow final\ idx\ \varphi = (\mathfrak{A}$
$\models \varphi)$
by (*simp only*: *final-def o-apply nullable-satisfies-bounded finalize-satisfies*)

8 Restrictions

lemma *satisfies-gen-restrict-RESTR*:
satisfies-gen $(\lambda k\ P\ \text{-}.\ restrict\ k\ P)\ \mathfrak{A}\ \varphi \longleftrightarrow \mathfrak{A} \models RESTR\ \varphi$
by (*induct* φ *arbitrary*: $\mathfrak{A}$) (*auto simp*: *restrict-Restrict*[*symmetric*] *assigns-Extend*

LEQ-0)

lemma *satisfies-gen-restrict*$_b$*-RESTR*:
satisfies-gen (λk *P n. restrict k P* $\wedge$ *len P* $\leq$ *n*) $\mathfrak{A}$ φ $\longleftrightarrow$ $\mathfrak{A}$ $\models_b$ *RESTR* φ
by (*induct* φ *arbitrary*: $\mathfrak{A}$) (*auto simp*: *restrict-Restrict*$_b$[*symmetric*] *assigns-Extend LEQ-0*)

lemma *distinct-FV*: *distinct* (*FV* φ)
by (*induct* φ) (*auto simp*: *distinct-FV0 distinct-map inj-on-def split*: *if-splits*)

lemma *sat-vars-RESTRICT-VARS*: *sat-vars-gen* (λ- - -. *True*) *vs* $\mathfrak{A}$ φ $\longleftrightarrow$ $\mathfrak{A}$ $\models$ *RESTRICT-VARS vs* φ
by (*induct vs arbitrary*: φ) (*force simp*: *satisfies-gen-restrict-RESTR restrict-Restrict*)+

lemma *sat-vars*$_b$*-RESTRICT-VARS*: *sat-vars-gen* (λ- *P n. len P* $\leq$ *n*) *vs* $\mathfrak{A}$ φ $\longleftrightarrow$ $\mathfrak{A}$ $\models_b$ *RESTRICT-VARS vs* φ
by (*induct vs arbitrary*: φ) (*auto simp*: *satisfies-gen-restrict*$_b$*-RESTR restrict-Restrict*$_b$)+

lemma *sat-RESTRICT*: *sat* $\mathfrak{A}$ φ $\longleftrightarrow$ $\mathfrak{A}$ $\models$ *RESTRICT* φ
unfolding *sat-def RESTRICT-def* **by** (*rule sat-vars-RESTRICT-VARS*)

lemma *sat*$_b$*-RESTRICT*: *sat*$_b$ $\mathfrak{A}$ φ $\longleftrightarrow$ $\mathfrak{A}$ $\models_b$ *RESTRICT* φ
unfolding *sat*$_b$*-def RESTRICT-def* **by** (*rule sat-vars*$_b$*-RESTRICT-VARS*)

end

context *Word-Formula*
begin

lemma *enc-Nil*: *Length* $\mathfrak{A}$ = *0* $\Longrightarrow$ *enc* $\mathfrak{A}$ = []
by (*metis length-0-conv length-enc*)

definition *decode idx* = *the-inv-into* {$\mathfrak{B}$. $\#_V$ $\mathfrak{B}$ = *idx*} *enc*

lemma *inj-on-enc*: *inj-on enc* {$\mathfrak{B}$. $\#_V$ $\mathfrak{B}$ = *idx*}
unfolding *inj-on-def* **by** (*auto simp*: *enc-inj*)

lemma *surj-enc*: $\forall$ *x*$\in$*set xs. size x* = *idx* $\Longrightarrow$ *xs* $\in$ *enc* ‘ {$\mathfrak{B}$. $\#_V$ $\mathfrak{B}$ = *idx*}
proof (*induct xs*)
case *Nil* **then show** *?case* **using** *ex-Length-0*[*of idx*]
by (*auto dest*: *enc-Nil*[*symmetric*])
next
case (*Cons x xs*) **then show** *?case*
by (*auto simp*: *image-iff enc-CONS nvars-CONS intro*: *exI*[*of* - *CONS x* $\mathfrak{A}$ **for** $\mathfrak{A}$])
qed

lemma *enc-decode*: $\forall$ *x* $\in$ *set xs. size x* = *idx* $\Longrightarrow$ *enc* (*decode idx xs*) = *xs*
unfolding *decode-def* **by** (*rule f-the-inv-into-f*[*OF inj-on-enc surj-enc*])

definition $TL\ \mathfrak{A} = decode\ (\#_V\ \mathfrak{A})\ (tl\ (enc\ \mathfrak{A}))$

lemma *in-set-tlD*: $x \in set\ (tl\ xs) \Longrightarrow x \in set\ xs$
  by (*cases xs*) *auto*

lemma *enc-TL*: $enc\ (TL\ \mathfrak{A}) = tl\ (enc\ \mathfrak{A})$
  unfolding *TL-def* **by** (*subst enc-decode*) (*auto dest*!: *in-set-encD in-set-tlD*)

lemma *nvars-TL*: $\#_V\ (TL\ \mathfrak{A}) = \#_V\ \mathfrak{A}$
  unfolding *TL-def decode-def*
  using *the-inv-into-into*[*OF inj-on-enc surj-enc subset-refl*, *of tl* (*enc* $\mathfrak{A}$)]
  by (*auto dest*!: *in-set-encD in-set-tlD*)

definition $lang\ idx\ \varphi = \{enc\ \mathfrak{A} \mid \mathfrak{A}.\ \mathfrak{A} \models \varphi \wedge \#_V\ \mathfrak{A} = idx\}$
definition $lang_b\ idx\ \varphi = \{enc\ \mathfrak{A} \mid \mathfrak{A}.\ \mathfrak{A} \models_b \varphi \wedge \#_V\ \mathfrak{A} = idx\}$

lemma *lang-eq-iff*: $lang\ idx\ \varphi = lang\ idx\ \psi \longleftrightarrow (\forall \mathfrak{A}.\ \#_V\ \mathfrak{A} = idx \longrightarrow \mathfrak{A} \models \varphi \longleftrightarrow \mathfrak{A} \models \psi)$
  unfolding *lang-def set-eq-iff* **by** *auto* (*metis enc-inj*)+

lemma $lang_b$*-eq-iff*: $lang_b\ idx\ \varphi = lang_b\ idx\ \psi \longleftrightarrow (\forall \mathfrak{A}.\ \#_V\ \mathfrak{A} = idx \longrightarrow \mathfrak{A} \models_b \varphi \longleftrightarrow \mathfrak{A} \models_b \psi)$
  unfolding $lang_b$*-def set-eq-iff* **by** *auto* (*metis enc-inj*)+

lemma *final-iff-Nil*: $wf\ idx\ \varphi \wedge left\text{-}formula\ \varphi \Longrightarrow final\ idx\ \varphi \longleftrightarrow ([] \in lang\ idx\ \varphi)$
  using *ex-Length-0*[*of idx*] *Length-0-inj*
   enc-Nil[*symmetric*] *enc-inj final-satisfies*[*of idx* φ]
  unfolding *lang-def* **by** *clarsimp metis*

lemma *nullable-iff-Nil*: $wf\ idx\ \varphi \wedge left\text{-}formula\ \varphi \Longrightarrow nullable\ \varphi \longleftrightarrow ([] \in lang_b\ idx\ \varphi)$
  using *ex-Length-0*[*of idx*] *Length-0-inj*
   enc-Nil[*symmetric*] *enc-inj nullable-satisfies-bounded*
  unfolding $lang_b$*-def* **by** *clarsimp metis*

lemma *lQuot-enc*:
  assumes $\bigwedge\mathfrak{A}.\ P\ \mathfrak{A} \Longrightarrow \#_V\ \mathfrak{A} = size\ a$
  shows $\{w.\ a\ \#\ w \in \{enc\ \mathfrak{A} \mid \mathfrak{A}.\ P\ \mathfrak{A}\}\} = \{enc\ \mathfrak{A} \mid \mathfrak{A}.\ P\ (CONS\ a\ \mathfrak{A})\}$
proof *safe*
  fix w **and** $\mathfrak{A} :: 'i$ **assume** $a\ \#\ w = enc\ \mathfrak{A}\ P\ \mathfrak{A}$
  with *assms* **have** $CONS\ a\ (TL\ \mathfrak{A}) = \mathfrak{A}\ enc\ \mathfrak{A} = a\ \#\ w$
    by (*auto simp*: *enc-inj*[*symmetric*] *enc-CONS enc-TL nvars-CONS nvars-TL dest*: *sym*)
   with ‹$P\ \mathfrak{A}$› **show** $\exists \mathfrak{A}.\ w = enc\ \mathfrak{A} \wedge P\ (CONS\ a\ \mathfrak{A})$ **by** (*auto simp*: *enc-TL intro*!: *exI*[*of - TL* $\mathfrak{A}$])
next
  fix $\mathfrak{A} :: 'i$ **assume** $P\ (CONS\ a\ \mathfrak{A})$

with *assms*[*of CONS a* $\mathfrak{A}$] **show** $\exists\mathfrak{A}'.\ a \mathbin{\#} enc\ \mathfrak{A} = enc\ \mathfrak{A}' \wedge P\ \mathfrak{A}'$
by (*auto simp*: *enc-CONS nvars-CONS intro*: *exI*[*of - CONS a* $\mathfrak{A}$])
qed

lemma *lang-lderiv*:
$[\![wf\ idx\ \varphi;\ idx = size\ x]\!] \Longrightarrow lang\ idx\ (lderiv\ x\ \varphi) = \{w.\ x \mathbin{\#} w \in lang\ idx\ \varphi\}$
unfolding *lang-def* **by** (*subst lQuot-enc*) (*auto simp*: *satisfies-lderiv nvars-CONS*)

lemma $lang_b$*-lderiv*:
$[\![wf\ idx\ \varphi;\ idx = size\ x]\!] \Longrightarrow lang_b\ idx\ (lderiv\ x\ \varphi) = \{w.\ x \mathbin{\#} w \in lang_b\ idx\ \varphi\}$
unfolding $lang_b$*-def* **by** (*subst lQuot-enc*) (*auto simp*: *satisfies-bounded-lderiv nvars-CONS*)

lemma *lang-norm*: $lang\ idx\ (norm\ \varphi) = lang\ idx\ \varphi$
unfolding *lang-eq-iff* **by** (*simp add*: *satisfies-norm*)

lemma $lang_b$*-norm*: $lang_b\ idx\ (norm\ \varphi) = lang_b\ idx\ \varphi$
unfolding $lang_b$*-eq-iff* **by** (*simp add*: *satisfies-bounded-norm*)

lemma *lang-size*: $[\![w \in lang\ idx\ \varphi;\ x \in set\ w]\!] \Longrightarrow size\ x = idx$
unfolding *lang-def* **by** (*auto elim*: *in-set-encD*)

lemma $lang_b$*-size*: $[\![w \in lang_b\ idx\ \varphi;\ x \in set\ w]\!] \Longrightarrow size\ x = idx$
unfolding $lang_b$*-def* **by** (*auto elim*: *in-set-encD*)

definition $language\ idx\ \varphi = \{enc\ \mathfrak{A} \mid \mathfrak{A}.\ sat\ \mathfrak{A}\ \varphi \wedge \#_V\ \mathfrak{A} = idx\}$
definition $language_b\ idx\ \varphi = \{enc\ \mathfrak{A} \mid \mathfrak{A}.\ sat_b\ \mathfrak{A}\ \varphi \wedge \#_V\ \mathfrak{A} = idx\}$

lemma *language-lang-RESTRICT*: $language\ idx\ \varphi = lang\ idx\ (RESTRICT\ \varphi)$
unfolding *language-def lang-def* **by** (*auto simp*: *sat-RESTRICT*)

lemma $language_b$*-*$lang_b$*-RESTRICT*: $language_b\ idx\ \varphi = lang_b\ idx\ (RESTRICT\ \varphi)$
unfolding $language_b$*-def* $lang_b$*-def* **by** (*auto simp*: sat_b*-RESTRICT*)

lemma *wf-RESTR*: $wf\ idx\ \varphi \Longrightarrow wf\ idx\ (RESTR\ \varphi)$
by (*induct* φ *arbitrary*: *idx*) (*auto simp*: *wf-Restrict LESS-SUC LEQ-0*)

lemma *wf-RESTRICT-VARS*: $[\![wf\ idx\ \varphi;\ list\text{-}all\ (\lambda(k,\ v).\ LESS\ k\ v\ idx)\ vs]\!] \Longrightarrow$
$wf\ idx\ (RESTRICT\text{-}VARS\ vs\ \varphi)$
by (*induct vs*) (*auto simp*: *wf-RESTR wf-Restrict*)

lemma *wf-FV-LESS*: $[\![wf\ idx\ \varphi;\ (k,\ v) \in set\ (FV\ \varphi)]\!] \Longrightarrow LESS\ k\ v\ idx$
by (*induct* φ *arbitrary*: *idx v*)
(*force simp*: *distinct-FV wf0-FV0-LESS LESS-SUC diff-Suc split*: *if-splits nat.splits*)+

lemma *wf-RESTRICT*: $wf\ idx\ \varphi \Longrightarrow wf\ idx\ (RESTRICT\ \varphi)$
unfolding *RESTRICT-def* **by** (*rule wf-RESTRICT-VARS*) (*auto simp*: *list-all-iff wf-FV-LESS*)

lemma *left-formula-RESTR*: *left-formula* $\varphi \Longrightarrow$ *left-formula* (*RESTR* φ)
by (*induct* φ) (*auto simp*: *left-formula-Restrict*)

lemma *left-formula-RESTRICT-VARS*: *left-formula* $\varphi \Longrightarrow$ *left-formula* (*RESTRICT-VARS vs* φ)
by (*induct vs*) (*auto simp*: *left-formula-RESTR left-formula-Restrict*)

lemma *left-formula-RESTRICT*: *left-formula* $\varphi \Longrightarrow$ *left-formula* (*RESTRICT* φ)
unfolding *RESTRICT-def* **by** (*rule left-formula-RESTRICT-VARS*)

end

sublocale *Word-Formula* <
bounded!: *DA alphabet idx* $\lambda\varphi$. *norm* (*RESTRICT* φ) $\lambda a\ \varphi$. *norm* (*lderiv a* φ) *nullable*
$\lambda\varphi$. *wf idx* $\varphi \wedge$ *left-formula* φ $\lambda\varphi$. *wf idx* $\varphi \wedge$ *left-formula* φ
$\lambda\varphi$. *to-language* ($lang_b$ *idx* φ) $\lambda\varphi$. *to-language* ($language_b$ *idx* φ) **for** *idx*
by *unfold-locales*
(*auto simp*: *nullable-iff-Nil* $lang_b$*-norm* $lang_b$*-lderiv wf-norm wf-lderiv*
left-formula-norm left-formula-lderiv alphabet-size $lang_b$*-size*
$language_b$*-*$lang_b$*-RESTRICT wf-RESTRICT left-formula-RESTRICT*)

sublocale *Word-Formula* <
DA alphabet idx $\lambda\varphi$. *norm* (*RESTRICT* φ) $\lambda a\ \varphi$. *norm* (*lderiv a* φ) *final idx*
$\lambda\varphi$. *wf idx* $\varphi \wedge$ *left-formula* φ $\lambda\varphi$. *wf idx* $\varphi \wedge$ *left-formula* φ
$\lambda\varphi$. *to-language* (*lang idx* φ) $\lambda\varphi$. *to-language* (*language idx* φ) **for** *idx*
by *unfold-locales*
(*auto simp*: *final-iff-Nil lang-norm lang-lderiv wf-norm wf-lderiv*
left-formula-norm left-formula-lderiv alphabet-size lang-size
language-lang-RESTRICT wf-RESTRICT left-formula-RESTRICT)

lemma (**in** *Word-Formula*) *check-eqv-soundness*:
$[\![\#_V\ \mathfrak{A} = idx;\ check\text{-}eqv\ idx\ \varphi\ \psi]\!] \Longrightarrow sat\ \mathfrak{A}\ \varphi \longleftrightarrow sat\ \mathfrak{A}\ \psi$
by (*drule soundness*, *drule injD*[*OF bij-is-inj*[*OF to-language-bij*]])
(*force simp*: *language-def set-eq-iff enc-inj*)

lemma (**in** *Word-Formula*) *bounded-check-eqv-soundness*:
$[\![\#_V\ \mathfrak{A} = idx;\ bounded.check\text{-}eqv\ idx\ \varphi\ \psi]\!] \Longrightarrow sat_b\ \mathfrak{A}\ \varphi \longleftrightarrow sat_b\ \mathfrak{A}\ \psi$
by (*drule bounded.soundness*, *drule injD*[*OF bij-is-inj*[*OF to-language-bij*]])
(*force simp*: $language_b$*-def set-eq-iff enc-inj*)

end

9 Concrete Atomic WS1S Formulas

definition *eval* $P\ x = (x\ |{\in}|\ P)$
definition *downshift* $P = (\lambda x.\ x - Suc\ 0)\ |`|\ (P\ |{-}|\ \{|0|\})$
definition *upshift* $P = Suc\ |`|\ P$

definition *lift bs i P* = (*if bs* ! *i then finsert 0* (*upshift P*) *else upshift P*)
definition *snoc n bs i P* = (*if bs* ! *i then finsert n P else P*)
definition *cut n P* = *ffilter* (λ*i*. *i* < *n*) *P*
definition *len P* = (*if P* = {||} *then 0 else Suc* (*fMax P*))

datatype-new *order* = *FO* | *SO*
datatype-compat *order*
derive *linorder order*

typedef *idx* = *UNIV* :: (*nat* × *nat*) *set* **by** (*rule UNIV-witness*)

setup-lifting *type-definition-idx*

lift-definition *ZERO* :: *idx* **is** (*0*, *0*) **.**
lift-definition *SUC* :: *order* ⇒ *idx* ⇒ *idx* **is**
  λ*ord* (*m*, *n*). *case ord of FO* ⇒ (*Suc m*, *n*) | *SO* ⇒ (*m*, *Suc n*) **.**
lift-definition *LESS* :: *order* ⇒ *nat* ⇒ *idx* ⇒ *bool* **is**
  λ*ord l* (*m*, *n*). *case ord of FO* ⇒ *l* < *m* | *SO* ⇒ *l* < *n* **.**
abbreviation *LEQ ord l idx* ≡ *LESS ord l* (*SUC ord idx*)

definition *MSB Is* ≡
  if ∀ *P* ∈ *set Is*. *P* = {||} *then 0 else Suc* (*Max* (⋃ *P* ∈ *set Is*. *fset P*))

lemma *MSB-Nil*[*simp*]: *MSB* [] = *0*
  unfolding *MSB-def* **by** *simp*

lemma *MSB-Cons*[*simp*]: *MSB* (*I* # *Is*) = *max* (*if I* = {||} *then 0 else Suc* (*fMax I*)) (*MSB Is*)
  unfolding *MSB-def* **including** *fset.lifting*
  by *transfer* (*auto simp*: *Max-Un list-all-iff Sup-bot-conv*(*2*)[*symmetric*] *simp del*: *Sup-bot-conv*(*2*))

lemma *MSB-append*[*simp*]: *MSB* (*I1* @ *I2*) = *max* (*MSB I1*) (*MSB I2*)
  by (*induct I1*) *auto*

lemma *MSB-insert-nth*[*simp*]:
  MSB (*insert-nth n P Is*) = *max* (*if P* = {||} *then 0 else Suc* (*fMax P*)) (*MSB Is*)
  by (*subst* (*2*) *append-take-drop-id*[*of n Is*, *symmetric*])
    (*simp only*: *insert-nth-take-drop MSB-append MSB-Cons MSB-Nil*)

lemma *MSB-greater*:
  ⟦*i* < *length Is*; *p* |∈| *Is* ! *i*⟧ ⟹ *p* < *MSB Is*
  unfolding *MSB-def* **by** (*fastforce simp*: *Bex-def in-set-conv-nth less-Suc-eq-le intro*: *Max-ge*)

lemma *MSB-mono*: *set I1* ⊆ *set I2* ⟹ *MSB I1* ≤ *MSB I2*
  unfolding *MSB-def* **including** *fset.lifting*
  by *transfer* (*auto simp*: *list-all-iff intro*!: *Max-ge*)

lemma *MSB-map-index'-CONS*[*simp*]:
MSB (*map-index' i* (*lift bs*) *Is*) =
(*if MSB Is* = *0* ∧ (∀ *i* ∈ {*i* ..< *i* + *length Is*}. ¬ *bs* ! *i*) *then 0 else Suc* (*MSB Is*))
by (*induct Is arbitrary*: *i*)
(*auto split*: *if-splits simp*: *mono-fMax-commute*[**where** *f* = *Suc*, *symmetric*] *mono-def*
lift-def upshift-def,
metis atLeastLessThan-iff le-antisym not-less-eq-eq)

lemma *MSB-map-index'-SNOC*[*simp*]:
MSB Is ≤ *n* ⟹ *MSB* (*map-index' i* (*snoc n bs*) *Is*) =
(*if* (∀ *i* ∈ {*i* ..< *i* + *length Is*}. ¬ *bs* ! *i*) *then MSB Is else Suc n*)
by (*induct Is arbitrary*: *i*)
(*auto split*: *if-splits simp*: *mono-fMax-commute*[**where** *f* = *Suc*, *symmetric*] *mono-def*
snoc-def, (*metis atLeastLessThan-iff le-antisym not-less-eq-eq*)+)

lemma *MSB-replicate*[*simp*]: *MSB* (*replicate n P*) = (*if P* = {||} ∨ *n* = *0 then 0 else Suc* (*fMax P*))
by (*induct n*) *auto*

typedef *interp* =
{(*n* :: *nat*, *I1* :: *nat fset list*, *I2* :: *nat fset list*) | *n I1 I2*. *MSB* (*I1* @ *I2*) ≤ *n*}
by *auto*

setup-lifting *type-definition-interp*

lift-definition *assigns* :: *nat* ⇒ *interp* ⇒ *order* ⇒ *nat fset* (-ˉ- [*900*, *999*, *999*] *999*)
is λ*n* (-, *I1*, *I2*) *ord*. *case ord of FO* ⇒ *if n* < *length I1 then I1* ! *n else* {||}
| *SO* ⇒ *if n* < *length I2 then I2* ! *n else* {||} **.**
lift-definition *nvars* :: *interp* ⇒ *idx* ($\#_V$ - [*1000*] *900*)
is λ(-, *I1*, *I2*). (*length I1*, *length I2*) **.**
lift-definition *Length* :: *interp* ⇒ *nat*
is λ(*n*, -, -). *n* **.**
lift-definition *Extend* :: *order* ⇒ *nat* ⇒ *interp* ⇒ *nat fset* ⇒ *interp*
is λ*ord i* (*n*, *I1*, *I2*) *P*. *case ord of*
FO ⇒ (*max n* (*if P* = {||} *then 0 else Suc* (*fMax P*)), *insert-nth i P I1*, *I2*)
| *SO* ⇒ (*max n* (*if P* = {||} *then 0 else Suc* (*fMax P*)), *I1*, *insert-nth i P I2*)
using *MSB-mono* **by** (*auto simp del*: *insert-nth-take-drop split*: *order.splits*)

lift-definition *CONS* :: (*bool list* × *bool list*) ⇒ *interp* ⇒ *interp*
is λ(*bs1*, *bs2*) (*n*, *I1*, *I2*).
(*Suc n*, *map-index* (*lift bs1*) *I1*, *map-index* (*lift bs2*) *I2*)
by *auto*

lift-definition *SNOC* :: (*bool list* × *bool list*) ⇒ *interp* ⇒ *interp*

is $\lambda(bs1, bs2)\ (n, I1, I2).$
$(Suc\ n,\ map\text{-}index\ (snoc\ n\ bs1)\ I1,\ map\text{-}index\ (snoc\ n\ bs2)\ I2)$
by $(auto\ simp{:}\ Let\text{-}def)$

abbreviation $enc\text{-}atom\text{-}bool\ I\ n \equiv map\ (\lambda P.\ n\ |{\in}|\ P)\ I$

abbreviation $enc\text{-}atom\ I1\ I2\ n \equiv (enc\text{-}atom\text{-}bool\ I1\ n,\ enc\text{-}atom\text{-}bool\ I2\ n)$

type-synonym $atom = bool\ list \times bool\ list$

lift-definition $enc :: interp \Rightarrow atom\ list$
is $\lambda(n, I1, I2).\ let\ m = MSB\ (I1\ @\ I2)\ in$
$map\ (enc\text{-}atom\ I1\ I2)\ [0\ ..{<}\ m]\ @$
$replicate\ (n - m)\ (replicate\ (length\ I1)\ False,\ replicate\ (length\ I2)\ False)$ **.**

lift-definition $zero :: idx \Rightarrow atom$
is $\lambda(m, n).\ (replicate\ m\ False,\ replicate\ n\ False)$ **.**

definition $extend\ ord\ b \equiv$
$\lambda(bs1, bs2).\ case\ ord\ of\ FO \Rightarrow (b\ \#\ bs1,\ bs2)\ |\ SO \Rightarrow (bs1,\ b\ \#\ bs2)$
lift-definition $size\text{-}atom :: bool\ list \times bool\ list \Rightarrow idx$
is $\lambda(bs1, bs2).\ (length\ bs1,\ length\ bs2)$ **.**

type-synonym $fo = nat$
type-synonym $so = nat$

datatype-new $ws1s =$
$Q\ fo\ |$
$Less\ fo\ fo\ |\ LessF\ fo\ fo\ |\ LessT\ fo\ fo\ |$
$In\ fo\ so\ |\ InT\ fo\ so$
datatype-compat $ws1s$
derive $linorder\ option$
derive $linorder\ ws1s$
type-synonym $formula = (ws1s,\ order)\ aformula$

primrec $wf0$ **where**
$wf0\ idx\ (Q\ m) = LESS\ FO\ m\ idx$
$|\ wf0\ idx\ (Less\ m1\ m2) = (LESS\ FO\ m1\ idx \wedge LESS\ FO\ m2\ idx)$
$|\ wf0\ idx\ (LessF\ m1\ m2) = (LESS\ FO\ m1\ idx \wedge LESS\ FO\ m2\ idx)$
$|\ wf0\ idx\ (LessT\ m1\ m2) = (LESS\ FO\ m1\ idx \wedge LESS\ FO\ m2\ idx)$
$|\ wf0\ idx\ (In\ m\ M) = (LESS\ FO\ m\ idx \wedge LESS\ SO\ M\ idx)$
$|\ wf0\ idx\ (InT\ m\ M) = (LESS\ FO\ m\ idx \wedge LESS\ SO\ M\ idx)$

fun $left\text{-}formula0$ **where**
$left\text{-}formula0\ x = True$

fun $FV0$ **where**
$FV0\ (Q\ m) = [(FO, m)]$
$|\ FV0\ (Less\ m1\ m2) = List.insert\ (FO, m1)\ [(FO, m2)]$

| *FV0* (*LessF m1 m2*) = *List.insert* (*FO*, *m1*) [(*FO*, *m2*)]
| *FV0* (*LessT m1 m2*) = *List.insert* (*FO*, *m1*) [(*FO*, *m2*)]
| *FV0* (*In m M*) = [(*FO*, *m*), (*SO*, *M*)]
| *FV0* (*InT m M*) = [(*FO*, *m*), (*SO*, *M*)]

fun *find0* **where**
find0 FO i (*Q m*) = (*i* = *m*)
| *find0 FO i* (*Less m1 m2*) = (*i* = *m1* ∨ *i* = *m2*)
| *find0 FO i* (*LessF m1 m2*) = (*i* = *m1* ∨ *i* = *m2*)
| *find0 FO i* (*LessT m1 m2*) = (*i* = *m1* ∨ *i* = *m2*)
| *find0 FO i* (*In m -*) = (*i* = *m*)
| *find0 SO i* (*In - M*) = (*i* = *M*)
| *find0 FO i* (*InT m -*) = (*i* = *m*)
| *find0 SO i* (*InT - M*) = (*i* = *M*)
| *find0 - - -* = *False*

primrec *decr0* **where**
decr0 ord k (*Q m*) = *Q* (*case-order* (*dec k*) *id ord m*)
| *decr0 ord k* (*Less m1 m2*) = *Less* (*case-order* (*dec k*) *id ord m1*) (*case-order* (*dec k*) *id ord m2*)
| *decr0 ord k* (*LessF m1 m2*) = *LessF* (*case-order* (*dec k*) *id ord m1*) (*case-order* (*dec k*) *id ord m2*)
| *decr0 ord k* (*LessT m1 m2*) = *LessT* (*case-order* (*dec k*) *id ord m1*) (*case-order* (*dec k*) *id ord m2*)
| *decr0 ord k* (*In m M*) = *In* (*case-order* (*dec k*) *id ord m*) (*case-order id* (*dec k*) *ord M*)
| *decr0 ord k* (*InT m M*) = *InT* (*case-order* (*dec k*) *id ord m*) (*case-order id* (*dec k*) *ord M*)

primrec *satisfies0* **where**
satisfies0 $\mathfrak{A}$ (*Q m*) = ($m^{\mathfrak{A}}FO \neq \{||\}$)
| *satisfies0* $\mathfrak{A}$ (*Less m1 m2*) =
(*let* $P1 = m1^{\mathfrak{A}}FO$; $P2 = m2^{\mathfrak{A}}FO$ *in if* $P1 = \{||\} \vee P2 = \{||\}$ *then False else fMin P1* < *fMin P2*)
| *satisfies0* $\mathfrak{A}$ (*LessF m1 m2*) =
(*let* $P1 = m1^{\mathfrak{A}}FO$; $P2 = m2^{\mathfrak{A}}FO$ *in*
if $P1 = \{||\}$ *then False else if* $P2 = \{||\}$ *then True else fMin P1* < *fMin P2*)
| *satisfies0* $\mathfrak{A}$ (*LessT m1 m2*) =
(*let* $P1 = m1^{\mathfrak{A}}FO$; $P2 = m2^{\mathfrak{A}}FO$ *in*
if $P2 = \{||\}$ *then True else if* $P1 = \{||\}$ *then False else fMin P1* < *fMin P2*)
| *satisfies0* $\mathfrak{A}$ (*In m M*) =
(*let* $P = m^{\mathfrak{A}}FO$ *in if* $P = \{||\}$ *then False else fMin P* $|\in|$ $M^{\mathfrak{A}}SO$)
| *satisfies0* $\mathfrak{A}$ (*InT m M*) =
(*let* $P = m^{\mathfrak{A}}FO$ *in if* $P = \{||\}$ *then True else fMin P* $|\in|$ $M^{\mathfrak{A}}SO$)

fun *lderiv0* **where**
lderiv0 (*bs1*, *bs2*) (*Q m*) = (*if bs1* ! *m then FBool True else FBase* (*Q m*))
| *lderiv0* (*bs1*, *bs2*) (*Less m1 m2*) = (*case* (*bs1* ! *m1*, *bs1* ! *m2*) *of*
(*False*, *False*) ⇒ *FBase* (*Less m1 m2*)

| (*True*, *False*) ⇒ *FBase* (*Q m2*)
| - ⇒ *FBool False*)
| *lderiv0* (*bs1*, *bs2*) (*LessF m1 m2*) = (*case* (*bs1* ! *m1*, *bs1* ! *m2*) *of*
(*False*, *False*) ⇒ *FBase* (*LessF m1 m2*)
| (*True*, *False*) ⇒ *FBool True*
| - ⇒ *FBool False*)
| *lderiv0* (*bs1*, *bs2*) (*LessT m1 m2*) = (*case* (*bs1* ! *m1*, *bs1* ! *m2*) *of*
(*False*, *False*) ⇒ *FBase* (*LessT m1 m2*)
| (*True*, *False*) ⇒ *FBool True*
| - ⇒ *FBool False*)
| *lderiv0* (*bs1*, *bs2*) (*In m M*) = (*case* (*bs1* ! *m*, *bs2* ! *M*) *of*
(*False*, -) ⇒ *FBase* (*In m M*)
| (*True*, *True*) ⇒ *FBool True*
| - ⇒ *FBool False*)
| *lderiv0* (*bs1*, *bs2*) (*InT m M*) = (*case* (*bs1* ! *m*, *bs2* ! *M*) *of*
(*False*, -) ⇒ *FBase* (*InT m M*)
| (*True*, *True*) ⇒ *FBool True*
| - ⇒ *FBool False*)

fun *rderiv0* **where**
rderiv0 (*bs1*, *bs2*) (*Q m*) = (*if bs1* ! *m then FBool True else FBase* (*Q m*))
| *rderiv0* (*bs1*, *bs2*) (*Less m1 m2*) = (*case bs1* ! *m2 of*
False ⇒ *FBase* (*Less m1 m2*)
| *True* ⇒ *FBase* (*LessF m1 m2*))
| *rderiv0* (*bs1*, *bs2*) (*LessF m1 m2*) = (*case* (*bs1* ! *m1*, *bs1* ! *m2*) *of*
(*True*, *False*) ⇒ *FBase* (*LessT m1 m2*)
| - ⇒ *FBase* (*LessF m1 m2*))
| *rderiv0* (*bs1*, *bs2*) (*LessT m1 m2*) = (*case bs1* ! *m2 of*
False ⇒ *FBase* (*LessT m1 m2*)
| *True* ⇒ *FBase* (*LessF m1 m2*))
| *rderiv0* (*bs1*, *bs2*) (*In m M*) = (*case* (*bs1* ! *m*, *bs2* ! *M*) *of*
(*True*, *True*) ⇒ *FBase* (*InT m M*)
| - ⇒ *FBase* (*In m M*))
| *rderiv0* (*bs1*, *bs2*) (*InT m M*) = (*case* (*bs1* ! *m*, *bs2* ! *M*) *of*
(*True*, *False*) ⇒ *FBase* (*In m M*)
| - ⇒ *FBase* (*InT m M*))

fun *nullable0* **where**
nullable0 (*Q m*) = *False*
| *nullable0* (*Less m1 m2*) = *False*
| *nullable0* (*LessF m1 m2*) = *False*
| *nullable0* (*LessT m1 m2*) = *True*
| *nullable0* (*In m M*) = *False*
| *nullable0* (*InT m M*) = *True*

lift-definition σ :: *idx* ⇒ *atom list*
is (λ(*n*, *N*). *map* (λ*bs*. (*take n bs*, *drop n bs*)) (*List.n-lists* (*n* + *N*) [*True*, *False*])) **.**

10 Interpretation

lemma *fMin-fimage-Suc*[*simp*]: x |∈| $A \Longrightarrow$ *fMin* (*Suc* |'| A) = *Suc* (*fMin* A)
by (*rule fMin-eqI*) (*auto intro*: *fMin-in*)

lemma *fMin-eq-0*[*simp*]: *0* |∈| $A \Longrightarrow$ *fMin* A = (*0* :: *nat*)
by (*rule fMin-eqI*) *auto*

lemma *insert-nth-Cons*[*simp*]:
insert-nth i x (*y* # *xs*) = (*case i of 0* ⇒ *x* # *y* # *xs* | *Suc i* ⇒ *y* # *insert-nth i x xs*)
by (*cases i*) *simp-all*

lemma *insert-nth-commute*[*simp*]:
assumes $j \leq i$ $i \leq$ *length xs*
shows *insert-nth j y* (*insert-nth i x xs*) = *insert-nth* (*Suc i*) *x* (*insert-nth j y xs*)
using *assms* **by** (*induct xs arbitrary*: *i j*) (*auto simp del*: *insert-nth-take-drop split*: *nat.splits*)

lemma *SUC-SUC*[*simp*]: *SUC ord* (*SUC ord′ idx*) = *SUC ord′* (*SUC ord idx*)
by *transfer* (*auto split*: *order.splits*)

lemma *LESS-SUC*[*simp*]:
LESS ord 0 (*SUC ord idx*)
LESS ord (*Suc l*) (*SUC ord idx*) = *LESS ord l idx*
ord ≠ *ord′* $\Longrightarrow$ *LESS ord l* (*SUC ord′ idx*) = *LESS ord l idx*
LESS ord l idx $\Longrightarrow$ *LESS ord l* (*SUC ord′ idx*)
by (*transfer*, *force split*: *order.splits*)+

lemma *nvars-Extend*[*simp*]:
$\#_V$ (*Extend ord i* $\mathfrak{A}$ *P*) = *SUC ord* ($\#_V$ $\mathfrak{A}$)
by (*transfer*, *force split*: *order.splits*)

lemma *Length-Extend*[*simp*]:
Length (*Extend k i* $\mathfrak{A}$ *P*) = *max* (*Length* $\mathfrak{A}$) (*if P* = {||} *then 0 else Suc* (*fMax P*))
unfolding *max-def* **by** (*split if-splits*, *transfer*) (*force split*: *order.splits*)

lemma *assigns-Extend*[*simp*]:
LEQ ord i ($\#_V$ $\mathfrak{A}$) $\Longrightarrow m^{Extend\ ord\ i\ \mathfrak{A}\ P}{}_{ord} = (if\ m = i\ then\ P\ else\ dec\ i\ m^{\mathfrak{A}}{}_{ord})$
ord ≠ *ord′* $\Longrightarrow m^{Extend\ ord\ i\ \mathfrak{A}\ P}{}_{ord'} = m^{\mathfrak{A}}{}_{ord'}$
by (*transfer*, *force simp*: *dec-def min-def nth-append split*: *order.splits*)+

lemma *Extend-commute-safe*[*simp*]:
⟦$j \leq i$; *LEQ ord i* ($\#_V$ $\mathfrak{A}$)⟧ $\Longrightarrow$
Extend ord j (*Extend ord i* $\mathfrak{A}$ *P1*) *P2* = *Extend ord* (*Suc i*) (*Extend ord j* $\mathfrak{A}$ *P2*) *P1*
by (*transfer*,
force simp del: *insert-nth-take-drop simp*: *replicate-add*[*symmetric*] *split*: *or-*

der.splits)

lemma *Extend-commute-unsafe*:
ord $\neq$ *ord'* $\Longrightarrow$ *Extend ord j* (*Extend ord' i* $\mathfrak{A}$ *P1*) *P2* = *Extend ord' i* (*Extend ord j* $\mathfrak{A}$ *P2*) *P1*
by (*transfer*, *force simp*: *replicate-add*[*symmetric*] *split*: *order.splits*)

lemma *Length-CONS*[*simp*]:
Length (*CONS x* $\mathfrak{A}$) = *Suc* (*Length* $\mathfrak{A}$)
by (*transfer*, *force split*: *order.splits*)

lemma *Length-SNOC*[*simp*]:
Length (*SNOC x* $\mathfrak{A}$) = *Suc* (*Length* $\mathfrak{A}$)
by (*transfer*, *force simp*: *Let-def split*: *order.splits*)

lemma *nvars-CONS*[*simp*]:
$\#_V$ (*CONS x* $\mathfrak{A}$) = $\#_V$ $\mathfrak{A}$
by (*transfer*, *force*)

lemma *nvars-SNOC*[*simp*]:
$\#_V$ (*SNOC x* $\mathfrak{A}$) = $\#_V$ $\mathfrak{A}$
by (*transfer*, *force simp*: *Let-def*)

lemma *assigns-CONS*[*simp*]:
assumes $\#_V$ $\mathfrak{A}$ = *size-atom bs1-bs2*
shows *LESS ord x* ($\#_V$ $\mathfrak{A}$) $\Longrightarrow$ $x^{CONS\ bs1\text{-}bs2\ \mathfrak{A}}{}_{ord}$ =
(*if split case-order bs1-bs2 ord* ! *x then finsert 0* (*upshift* ($x^{\mathfrak{A}}$*ord*)) *else upshift* ($x^{\mathfrak{A}}$*ord*))
by (*insert assms*, *transfer*) (*auto simp*: *lift-def split*: *order.splits*)

lemma *assigns-SNOC*[*simp*]:
assumes $\#_V$ $\mathfrak{A}$ = *size-atom bs1-bs2*
shows *LESS ord x* ($\#_V$ $\mathfrak{A}$) $\Longrightarrow$ $x^{SNOC\ bs1\text{-}bs2\ \mathfrak{A}}{}_{ord}$ =
(*if split case-order bs1-bs2 ord* ! *x then finsert* (*Length* $\mathfrak{A}$) ($x^{\mathfrak{A}}$*ord*) *else* $x^{\mathfrak{A}}$*ord*)
by (*insert assms*, *transfer*) (*force simp*: *snoc-def Let-def split*: *order.splits*)

lemma *map-index'-eq-conv*[*simp*]:
map-index' i f xs = *map-index' j g xs* = ($\forall$ *k* < *length xs*. *f* (*i* + *k*) (*xs* ! *k*) = *g* (*j* + *k*) (*xs* ! *k*))
proof (*induct xs arbitrary*: *i j*)
case *Cons* **from** *Cons*(*1*)[*of Suc i Suc j*] **show** *?case* **by** (*auto simp*: *nth-Cons split*: *nat.splits*)
qed *simp*

lemma *fMax-Diff-0*[*simp*]: *Suc m* |$\in$| *P* $\Longrightarrow$ *fMax* (*P* |−| {|*0*|}) = *fMax P*
by (*rule fMax-eqI*) (*auto intro*: *fMax-in dest*: *fMax-ge*)

lemma *Suc-fMax-pred-fimage*[*simp*]:
assumes *Suc p* |$\in$| *P 0* |$\notin$| *P*

shows *Suc* (*fMax* (($\lambda x.$ *x* − *Suc 0*) |‘| *P*)) = *fMax P*
using *assms* **by** (*subst mono-fMax-commute*[*of Suc*, *unfolded mono-def*, *simplified*]) (*auto simp*: *o-def*)

lemma *Extend-CONS*[*simp*]: $\#_V$ $\mathfrak{A}$ = *size-atom x* $\Longrightarrow$ *Extend ord 0* (*CONS x* $\mathfrak{A}$) *P* =
CONS (*extend ord* (*eval P 0*) *x*) (*Extend ord 0* $\mathfrak{A}$ (*downshift P*))
by *transfer* (*auto simp*: *extend-def o-def gr0-conv-Suc*
mono-fMax-commute[*of Suc*, *symmetric*, *unfolded mono-def*, *simplified*]
lift-def upshift-def downshift-def eval-def
dest!: *fsubset-fsingletonD split*: *order.splits*)

lemma *size-atom-extend*[*simp*]:
size-atom (*extend ord b x*) = *SUC ord* (*size-atom x*)
unfolding *extend-def* **by** *transfer* (*simp split*: *prod.splits order.splits*)

lemma *size-atom-zero*[*simp*]:
size-atom (*zero idx*) = *idx*
unfolding *extend-def* **by** *transfer* (*simp split*: *prod.splits order.splits*)

lemma *interp-eqI*:
⟦*Length* $\mathfrak{A}$ = *Length* $\mathfrak{B}$; $\#_V$ $\mathfrak{A}$ = $\#_V$ $\mathfrak{B}$; $\bigwedge$*m k*. *LESS k m* ($\#_V$ $\mathfrak{A}$) $\Longrightarrow$ $m^{\mathfrak{A}}k$ = $m^{\mathfrak{B}}k$⟧ $\Longrightarrow$ $\mathfrak{A}$ = $\mathfrak{B}$
by *transfer* (*force split*: *order.splits intro*!: *nth-equalityI*)

lemma *Extend-SNOC-cut*[*unfolded eval-def cut-def Length-SNOC*, *simp*]:
⟦*len P* ≤ *Length* (*SNOC x* $\mathfrak{A}$); $\#_V$ $\mathfrak{A}$ = *size-atom x*⟧ $\Longrightarrow$
Extend ord 0 (*SNOC x* $\mathfrak{A}$) *P* =
SNOC (*extend ord* (*if eval P* (*Length* $\mathfrak{A}$) *then True else False*) *x*) (*Extend ord 0* $\mathfrak{A}$ (*cut* (*Length* $\mathfrak{A}$) *P*))
by *transfer* (*fastforce simp*: *extend-def len-def cut-def ffilter-eq-fempty-iff snoc-def eval-def*
split: *if-splits order.splits dest*: *fMax-ge fMax-boundedD intro*: *fMax-in*)

lemma *nth-replicate-simp*: *replicate m x* ! *i* = (*if i* < *m then x else* [] ! (*i* − *m*))
by (*induct m arbitrary*: *i*) (*auto simp*: *nth-Cons'*)

lemma *MSB-eq-SucD*: *MSB Is* = *Suc x* $\Longrightarrow$ ∃ *P*∈*set Is*. *x* |∈| *P*
using *Max-in*[*of* $\bigcup$*x*∈*set Is*. *fset x*]
unfolding *MSB-def* **by** (*force simp*: *fmember-def split*: *if-splits*)

lemma *last-enc-atom-MSB*:
fixes *I1 I2*
defines *xs* ≡ *map* (*enc-atom I1 I2*) [*0..*<*MSB* (*I1* @ *I2*)]
assumes *m* = *length I1 n* = *length I2 xs* ≠ []
shows *last xs* ≠ (*replicate m False*, *replicate n False*)
proof (*rule ccontr*, *unfold not-not*)
assume *last xs* = (*replicate m False*, *replicate n False*)
moreover from ‹*xs* ≠ []› **obtain** *i* **where** *i*: *MSB* (*I1* @ *I2*) = *Suc i*

unfolding *xs-def* **by** (*auto simp*: *gr0-conv-Suc*)
ultimately have *enc-atom I1 I2 i* = (*replicate m False*, *replicate n False*)
unfolding *xs-def* **by** *auto*
with *i* **show** *False*
by (*auto simp*: *list-eq-iff-nth-eq max-def in-set-conv-nth dest*!: *MSB-eq-SucD split*: *if-splits*)
qed

lemma *append-replicate-inj*:
assumes $xs \neq [] \Longrightarrow last\ xs \neq x$ **and** $ys \neq [] \Longrightarrow last\ ys \neq x$
shows $xs\ @\ replicate\ m\ x = ys\ @\ replicate\ n\ x \longleftrightarrow (xs = ys \wedge m = n)$
proof *safe*
from *assms* **have** *assms'*: $xs \neq [] \Longrightarrow rev\ xs\ !\ 0 \neq x\ ys \neq [] \Longrightarrow rev\ ys\ !\ 0 \neq x$
by (*auto simp*: *hd-rev hd-conv-nth*[*symmetric*])
assume $*$: $xs\ @\ replicate\ m\ x = ys\ @\ replicate\ n\ x$
then have $rev\ (xs\ @\ replicate\ m\ x) = rev\ (ys\ @\ replicate\ n\ x)$ **..**
then have $replicate\ m\ x\ @\ rev\ xs = replicate\ n\ x\ @\ rev\ ys$ **by** *simp*
then have $take\ (max\ m\ n)\ (replicate\ m\ x\ @\ rev\ xs) = take\ (max\ m\ n)\ (replicate\ n\ x\ @\ rev\ ys)$ **by** *simp*
then have $replicate\ m\ x\ @\ take\ (max\ m\ n - m)\ (rev\ xs) =$
$replicate\ n\ x\ @\ take\ (max\ m\ n - n)\ (rev\ ys)$ **by** (*auto simp*: *min-def max-def split*: *if-splits*)
then have $(replicate\ m\ x\ @\ take\ (max\ m\ n - m)\ (rev\ xs))\ !\ min\ m\ n =$
$(replicate\ n\ x\ @\ take\ (max\ m\ n - n)\ (rev\ ys))\ !\ min\ m\ n$ **by** *simp*
with *arg-cong*[*OF* $*$, *of length*, *simplified*] *assms'* **show** $m = n$
by (*cases* $xs = []$ $ys = []$ *rule*: *bool.exhaust*[*case-product bool.exhaust*])
(*auto simp*: *min-def nth-append split*: *if-splits*)
with $*$ **show** $xs = ys$ **by** *auto*
qed

lemma *enc-inj*[*simp*]: $\#_V\ \mathfrak{A} = \#_V\ \mathfrak{B} \Longrightarrow enc\ \mathfrak{A} = enc\ \mathfrak{B} \Longrightarrow \mathfrak{A} = \mathfrak{B}$
using *MSB-greater*
by *transfer* (*elim exE conjE*, *hypsubst*, *unfold prod.case Pair-eq Let-def*, *elim conjE*, *simp only*:,
subst (*asm*) *append-replicate-inj*,
erule (*2*) *last-enc-atom-MSB*[*OF sym sym*], *erule last-enc-atom-MSB*[*OF refl refl*],
auto simp: *list-eq-iff-nth-eq*, (*metis less-max-iff-disj*)+)

lemma *length-enc*[*simp*]: $length\ (enc\ \mathfrak{A}) = Length\ \mathfrak{A}$
by *transfer* (*auto simp*: *Let-def*)

lemma *in-set-encD*[*simp*]:
$x \in set\ (enc\ \mathfrak{A}) \Longrightarrow \#_V\ \mathfrak{A} = size\text{-}atom\ x$
by *transfer* (*auto simp*: *Let-def split*: *if-splits*)

lemma *fin-lift*[*simp*]: $m\ |{\in}|\ lift\ bs\ i\ (I\ !\ i) \longleftrightarrow (case\ m\ of\ 0 \Rightarrow bs\ !\ i\ |\ Suc\ m \Rightarrow m\ |{\in}|\ I\ !\ i)$
unfolding *lift-def upshift-def* **by** (*auto split*: *nat.splits*)

lemma *enc-CONS*[*simp*]:
  assumes $\#_V\ \mathfrak{A} = \textit{size-atom}\ x$
  shows $\textit{enc}\ (\textit{CONS}\ x\ \mathfrak{A}) = x\ \#\ \textit{enc}\ \mathfrak{A}$
  using *assms* **by** *transfer* (*fastforce simp add*: *Let-def nth-append nth-Cons simp del*: *upt-conv-Nil diff-is-0-eq′ atLeastLessThan-empty split*: *prod.splits nat.splits intro*!: *nth-equalityI*)

lemma *ex-Length-0*[*simp*]:
  $\exists\mathfrak{A}.\ \textit{Length}\ \mathfrak{A} = 0 \wedge \#_V\ \mathfrak{A} = \textit{idx}$
  by *transfer* (*auto intro*!: *exI*[*of - replicate m* {||} **for** *m*])

lemma *is-empty-inj*[*simp*]: $[\![\textit{Length}\ \mathfrak{A} = 0;\ \textit{Length}\ \mathfrak{B} = 0;\ \#_V\ \mathfrak{A} = \#_V\ \mathfrak{B}]\!] \Longrightarrow \mathfrak{A} = \mathfrak{B}$
  by *transfer* (*simp add*: *list-eq-iff-nth-eq split*: *prod.splits*, *metis MSB-greater fMax-in less-nat-zero-code*)

lemma *set-σ-length-atom*[*simp*]: $(x \in \textit{set}\ (\sigma\ \textit{idx})) \longleftrightarrow \textit{idx} = \textit{size-atom}\ x$
  by *transfer* (*auto simp*: *set-n-lists enum-UNIV image-iff intro*!: *exI*[*of - I1 @ I2* **for** *I1 I2*])

lemma *fMin-less-Length*[*simp*]: $x\ |\!\in\!|\ m1^{\mathfrak{A}}k \Longrightarrow \textit{fMin}\ (m1^{\mathfrak{A}}k) < \textit{Length}\ \mathfrak{A}$
  by *transfer*
    (*force elim*: *order.strict-trans2*[*OF MSB-greater*, *rotated* −*1*] *intro*: *fMin-in split*: *order.splits*)

lemma *min-Length-fMin*[*simp*]: $x\ |\!\in\!|\ m1^{\mathfrak{A}}k \Longrightarrow \min\ (\textit{Length}\ \mathfrak{A})\ (\textit{fMin}\ (m1^{\mathfrak{A}}k)) = \textit{fMin}\ (m1^{\mathfrak{A}}k)$
  using *fMin-less-Length*[*of x m1* 𝔄 *k*] **unfolding** *fMin-def* **by** *auto*

lemma *assigns-less-Length*[*simp*]: $x\ |\!\in\!|\ m1^{\mathfrak{A}}k \Longrightarrow x < \textit{Length}\ \mathfrak{A}$
  by *transfer* (*force dest*: *MSB-greater split*: *order.splits if-splits*)

lemma *Length-notin-assigns*[*simp*]: $\textit{Length}\ \mathfrak{A}\ |\!\notin\!|\ m^{\mathfrak{A}}k$
  by (*metis assigns-less-Length less-not-refl*)

lemma *nth-zero*[*simp*]: $\textit{LESS ord m}\ (\#_V\ \mathfrak{A}) \Longrightarrow \neg\ \textit{split case-order}\ (\textit{zero}\ (\#_V\ \mathfrak{A}))\ \textit{ord}\ !\ m$
  by *transfer* (*auto split*: *order.splits*)

lemma *in-fimage-Suc*[*simp*]: $x\ |\!\in\!|\ \textit{Suc}\ |`|\ A \longleftrightarrow (\exists y.\ y\ |\!\in\!|\ A \wedge x = \textit{Suc}\ y)$
  by *blast*

lemma *fimage-Suc-inj*[*simp*]: $\textit{Suc}\ |`|\ A = \textit{Suc}\ |`|\ B \longleftrightarrow A = B$
  by *blast*

lemma *MSB-eq0-D*: $\textit{MSB}\ I = 0 \Longrightarrow x < \textit{length}\ I \Longrightarrow I\ !\ x = \{||\}$
  unfolding *MSB-def* **by** (*auto split*: *if-splits*)

lemma *Suc-in-fimage-Suc*: $Suc\ x\ |\in|\ Suc\ |`|\ X \longleftrightarrow x\ |\in|\ X$
  by *auto*

lemma *Suc-in-fimage-Suc-o-Suc*[*simp*]: $Suc\ x\ |\in|\ (Suc \circ Suc)\ |`|\ X \longleftrightarrow x\ |\in|\ Suc$
$|`|\ X$
  by *auto*

lemma *finsert-same-eq-iff* [*simp*]: $finsert\ k\ X = finsert\ k\ Y \longleftrightarrow X\ |-|\ \{|k|\} = Y$
$|-|\ \{|k|\}$
  by *auto*

lemma *fimage-Suc-o-Suc-eq-fimage-Suc-iff* [*simp*]:
  $((Suc \circ Suc)\ |`|\ X = Suc\ |`|\ Y) \longleftrightarrow (Suc\ |`|\ X = Y)$
  by (*metis fimage-Suc-inj fset.map-comp*)

lemma *fMax-image-Suc*[*simp*]: $x\ |\in|\ P \Longrightarrow fMax\ (Suc\ |`|\ P) = Suc\ (fMax\ P)$
  by (*rule fMax-eqI*) (*metis Suc-le-mono fMax-ge fimageE*, *metis fimageI fempty-iff fMax-in*)

lemma *len-downshift-helper*:
  $x\ |\in|\ P \Longrightarrow Suc\ (fMax\ ((\lambda x.\ x - Suc\ 0)\ |`|\ (P\ |-|\ \{|0|\}))) \neq fMax\ P \Longrightarrow xa$
$|\in|\ P \Longrightarrow xa = 0$
proof −
  assume *a1*: $xa\ |\in|\ P$
  assume *a2*: $Suc\ (fMax\ ((\lambda x.\ x - Suc\ 0)\ |`|\ (P\ |-|\ \{|0|\}))) \neq fMax\ P$
  have $xa\ |\in|\ \{|0|\} \longrightarrow xa = 0$ **by** *fastforce*
  moreover
  { assume $xa\ |\notin|\ \{|0|\}$
    hence $0\ |\notin|\ P\ |-|\ \{|0|\} \wedge xa\ |\notin|\ \{|0|\}$ **by** *blast*
      then obtain $esk1_1 :: nat \Rightarrow nat$ **where** $xa = 0$ **using** *a1 a2* **by** (*metis Suc-fMax-pred-fimage fMax-Diff-0 fminus-iff not0-implies-Suc*) **}**
  ultimately show $xa = 0$ **by** *blast*
qed

declare [[*goals-limit* = *50*]]

definition *restrict ord P* = (*case ord of FO* ⇒ $P \neq \{||\}$ | *SO* ⇒ *True*)
definition *Restrict ord i* = (*case ord of FO* ⇒ *FBase* (*Q i*) | *SO* ⇒ *FBool True*)

permanent-interpretation *WS1S*: *Word-Formula SUC LESS assigns nvars Extend CONS SNOC Length*
  extend size-atom zero eval downshift upshift finsert cut len restrict Restrict
  left-formula0 FV0 find0 wf0 decr0 satisfies0 nullable0 lderiv0 rderiv0 undefined enc σ *ZERO*
  defining *norm* = *Formula-Operations.norm find0 decr0*
  and *nFOr* = *Formula-Operations.nFOr* :: *formula* ⇒ -
  and *nFAnd* = *Formula-Operations.nFAnd* :: *formula* ⇒ -
  and *nFNot* = *Formula-Operations.nFNot* :: *formula* ⇒ -

and *nFEx* = *Formula-Operations.nFEx find0 decr0*
and *nFAll* = *Formula-Operations.nFAll find0 decr0*
and *decr* = *Formula-Operations.decr decr0* :: - ⇒ - ⇒ *formula* ⇒ -
and *find* = *Formula-Operations.find find0* :: - ⇒ - ⇒ *formula* ⇒ -
and *FV* = *Formula-Operations.FV FV0*
and *RESTR* = *Formula-Operations.RESTR Restrict*
and *RESTRICT* = *Formula-Operations.RESTRICT Restrict FV0*
and *deriv* = *λd0* (*a* :: *atom*) (*φ* :: *formula*). *Formula-Operations.deriv extend d0 a φ*
and *nullable* = *λφ* :: *formula*. *Formula-Operations.nullable nullable0 φ*
and *fut-default* = *Formula.fut-default extend zero rderiv0*
and *fut* = *Formula.fut extend zero find0 decr0 rderiv0*
and *finalize* = *Formula.finalize SUC extend zero find0 decr0 rderiv0*
and *final* = *Formula.final SUC extend zero find0 decr0*
nullable0 rderiv0 :: *idx* ⇒ *formula* ⇒ -
and *ws1s-wf* = *Formula-Operations.wf SUC* (*wf0* :: *idx* ⇒ *ws1s* ⇒ -)
and *ws1s-left-formula* = *Formula-Operations.left-formula left-formula0* :: *formula* ⇒ -
and *check-eqv* = *λidx*. *DA.check-eqv* (*σ idx*)
(*λφ*. *norm* (*RESTRICT φ*) :: (*ws1s*, *order*) *aformula*)
(*λa φ*. *norm* (*deriv* (*lderiv0* :: - ⇒ - ⇒ *formula*) (*a* :: *atom*) *φ*))
(*final idx*) (*λφ* :: *formula*. *ws1s-wf idx φ* ∧ *ws1s-left-formula φ*)
and *bounded-check-eqv* = *λidx*. *DA.check-eqv* (*σ idx*)
(*λφ*. *norm* (*RESTRICT φ*) :: (*ws1s*, *order*) *aformula*)
(*λa φ*. *norm* (*deriv* (*lderiv0* :: - ⇒ - ⇒ *formula*) (*a* :: *atom*) *φ*))
nullable (*λφ* :: *formula*. *ws1s-wf idx φ* ∧ *ws1s-left-formula φ*)
and *automaton* = *DA.automaton*
(*λa φ*. *norm* (*deriv lderiv0* (*a* :: *atom*) *φ* :: *formula*))
proof
fix *k idx* **and** *a* :: *ws1s* **and** *l* **assume** *wf0* (*SUC k idx*) *a LESS k l* (*SUC k idx*) ¬ *find0 k l a*
then show *wf0 idx* (*decr0 k l a*)
by (*induct a*) (*unfold wf0.simps decr0.simps find0.simps*,
(*transfer*, *force simp*: *dec-def split*: *if-splits order.splits*)+)
next
fix *k* **and** *a* :: *ws1s* **and** *l* **assume** *left-formula0 a*
then show *left-formula0* (*decr0 k l a*) **by** (*induct a*) *simp-all*
next
fix *i k* **and** *a* :: *ws1s* **and** 𝔄 :: *interp* **and** *P* **assume** ¬ *find0 k i a LESS k i* (*SUC k* ($\#_V$ 𝔄))
then show *satisfies0* (*Extend k i* 𝔄 *P*) *a* = *satisfies0* 𝔄 (*decr0 k i a*)
by (*induct a*) (*auto split*: *if-splits order.splits*)
next
fix *idx* **and** *a* :: *ws1s* **and** *x* **assume** *wf0 idx a*
then show *Formula-Operations.wf SUC wf0 idx* (*lderiv0 x a*)
by (*induct rule*: *lderiv0.induct*)
(*auto simp*: *Formula-Operations.wf.simps Let-def split*: *bool.splits order.splits*)
next
fix *a* :: *ws1s* **and** *x* **assume** *left-formula0 a*

then show *Formula-Operations.left-formula left-formula0* (*lderiv0 x a*)
by (*induct a rule*: *lderiv0.induct*)
(*auto simp*: *Formula-Operations.left-formula.simps split*: *bool.splits*)
next
fix *idx* **and** *a* :: *ws1s* **and** *x* **assume** *wf0 idx a*
then show *Formula-Operations.wf SUC wf0 idx* (*rderiv0 x a*)
by (*induct rule*: *lderiv0.induct*)
(*auto simp*: *Formula-Operations.wf.simps Let-def split*: *bool.splits order.splits*)
next
fix $\mathfrak{A}$ :: *interp* **and** *a* :: *ws1s*
assume *Length* $\mathfrak{A}$ = *0*
then show *nullable0 a* = *satisfies0* $\mathfrak{A}$ *a*
by (*induct a*, *unfold wf0.simps nullable0.simps satisfies0.simps Let-def*)
(*transfer*, (*auto 0 2 dest*: *MSB-greater split*: *prod.splits if-splits*) [])+
next
note *Formula-Operations.satisfies-gen.simps*[*simp*] *Let-def*[*simp*] *upshift-def*[*simp*]
fix *x* :: *atom* **and** *a* :: *ws1s* **and** $\mathfrak{A}$ :: *interp*
assume *wf0* ($\#_V$ $\mathfrak{A}$) *a* $\#_V$ $\mathfrak{A}$ = *size-atom x*
then show *Formula-Operations.satisfies Extend Length satisfies0* $\mathfrak{A}$ (*lderiv0 x a*) = *satisfies0* (*CONS x* $\mathfrak{A}$) *a*
proof (*induct a*)
qed (*auto split*: *prod.splits bool.splits*)
next
note *Formula-Operations.satisfies-gen.simps*[*simp*] *Let-def*[*simp*] *upshift-def*[*simp*]
fix *x* :: *atom* **and** *a* :: *ws1s* **and** $\mathfrak{A}$ :: *interp*
assume *wf0* ($\#_V$ $\mathfrak{A}$) *a* $\#_V$ $\mathfrak{A}$ = *size-atom x*
then show *Formula-Operations.satisfies-bounded Extend Length len satisfies0* $\mathfrak{A}$ (*lderiv0 x a*) = *satisfies0* (*CONS x* $\mathfrak{A}$) *a*
proof (*induct a*)
qed (*auto split*: *prod.splits bool.splits*)
next
note *Formula-Operations.satisfies-gen.simps*[*simp*] *Let-def*[*simp*]
fix *x* :: *atom* **and** *a* :: *ws1s* **and** $\mathfrak{A}$ :: *interp*
assume *wf0* ($\#_V$ $\mathfrak{A}$) *a* $\#_V$ $\mathfrak{A}$ = *size-atom x*
then show *Formula-Operations.satisfies-bounded Extend Length len satisfies0* $\mathfrak{A}$ (*rderiv0 x a*) = *satisfies0* (*SNOC x* $\mathfrak{A}$) *a*
proof (*induct a*)
case *Less* **then show** *?case* **by** (*auto split*: *prod.splits*) (*metis fMin-less-Length less-not-sym*)
next
case *LessT* **then show** *?case* **by** (*auto split*: *prod.splits*) (*metis fMin-less-Length less-not-sym*)
next
case *LessF* **then show** *?case* **by** (*auto split*: *prod.splits*) (*metis fMin-less-Length less-not-sym*)
next
case *In* **then show** *?case* **by** (*auto split*: *prod.splits*) (*metis fMin-less-Length less-not-sym*)+
next

case *InT* **then show** *?case* **by** (*auto split*: *prod.splits*) (*metis fMin-less-Length less-not-sym*)+
qed (*auto split*: *prod.splits*)
next
fix *a* :: *ws1s* **and** $\mathfrak{A}$ $\mathfrak{B}$ :: *interp*
assume *wf0* ($\#_V$ $\mathfrak{B}$) *a* $\#_V$ $\mathfrak{A}$ = $\#_V$ $\mathfrak{B}$ ($\bigwedge m\ k.$ *LESS k m* ($\#_V$ $\mathfrak{B}$) $\Longrightarrow$ $m^{\mathfrak{A}}k$ = $m^{\mathfrak{B}}k$)
left-formula0 a
then show *satisfies0* $\mathfrak{A}$ *a* $\longleftrightarrow$ *satisfies0* $\mathfrak{B}$ *a* **by** (*induct a*) *simp-all*
next
fix *a* :: *ws1s*
let *?d* = *Formula-Operations.deriv extend lderiv0*
def $\Phi \equiv \lambda a.$ (*case a of*
Q i $\Rightarrow$ {*FBase* (*Q i*), *FBool True*}
| *Less i j* $\Rightarrow$ {*FBase* (*Less i j*), *FBase* (*Q j*), *FBool True*, *FBool False*}
| *LessT i j* $\Rightarrow$ {*FBase* (*LessT i j*), *FBool True*, *FBool False*}
| *LessF i j* $\Rightarrow$ {*FBase* (*LessF i j*), *FBool True*, *FBool False*}
| *In i I* $\Rightarrow$ {*FBase* (*In i I*), *FBool True*, *FBool False*}
| *InT i I* $\Rightarrow$ {*FBase* (*InT i I*), *FBool True*, *FBool False*}) :: (*ws1s*, *order*) *aformula set*
{ **fix** *xs*
note *Formula-Operations.fold-deriv-FBool*[*simp*] *Formula-Operations.deriv.simps*[*simp*] Φ-*def*[*simp*]
have $\forall$ *a*. *fold ?d xs* (*FBase a*) $\in$ Φ *a*
by (*induct xs*) (*auto split*: *ws1s.splits bool.splits if-splits*, *metis*+)
}
moreover have *finite* (Φ *a*) **unfolding** Φ-*def* **by** (*auto split*: *ws1s.splits*)
ultimately show *finite* {*fold ?d xs* (*FBase a*) | *xs*. *True*}
by (*blast intro*: *finite-subset*)
next
fix *a* :: *ws1s*
let *?d* = *Formula-Operations.deriv extend rderiv0*
def $\Phi \equiv \lambda a.$ (*case a of*
Q i $\Rightarrow$ {*FBase* (*Q i*), *FBool True*}
| *Less i j* $\Rightarrow$ {*FBase* (*Less i j*), *FBase* (*LessF i j*), *FBase* (*LessT i j*)}
| *LessT i j* $\Rightarrow$ {*FBase* (*LessT i j*), *FBase* (*LessF i j*)}
| *LessF i j* $\Rightarrow$ {*FBase* (*LessF i j*), *FBase* (*LessT i j*)}
| *In i I* $\Rightarrow$ {*FBase* (*In i I*), *FBase* (*InT i I*)}
| *InT i I* $\Rightarrow$ {*FBase* (*InT i I*), *FBase* (*In i I*)}) :: (*ws1s*, *order*) *aformula set*
{ **fix** *x xs*
note *Formula-Operations.fold-deriv-FBool*[*simp*] *Formula-Operations.deriv.simps*[*simp*] Φ-*def*[*simp*]
have $\forall$ *a*. *fold ?d xs* (*FBase a*) $\in$ Φ *a*
by (*induct xs*) (*auto split*: *ws1s.splits bool.splits if-splits*, *metis*+)
}
moreover have *finite* (Φ *a*) **unfolding** Φ-*def* **by** (*auto split*: *ws1s.splits*)
ultimately show *finite* {*fold ?d xs* (*FBase a*) | *xs*. *True*}
by (*blast intro*: *finite-subset*)
next

fix *k l* **and** *a* :: *ws1s*
show *find0 k l a* $\longleftrightarrow$ (*k*, *l*) $\in$ *set* (*FV0 a*) **by** (*induct a rule*: *find0.induct*) *auto*
next
fix *a* :: *ws1s*
show *distinct* (*FV0 a*) **by** (*induct a*) *auto*
next
fix *idx a k v*
assume *wf0 idx a* (*k*, *v*) $\in$ *set* (*FV0 a*)
then show *LESS k v idx* **by** (*induct a*) *auto*
next
fix *idx k i*
assume *LESS k i idx*
then show *Formula-Operations.wf SUC wf0 idx* (*Restrict k i*)
unfolding *Restrict-def* **by** (*cases k*) (*auto simp*: *Formula-Operations.wf.simps*)
next
fix *k i*
show *Formula-Operations.left-formula left-formula0* (*Restrict k i*)
unfolding *Restrict-def* **by** (*cases k*) (*auto simp*: *Formula-Operations.left-formula.simps*)
next
fix *i* $\mathfrak{A}$ *k P*
assume $i^{\mathfrak{A}}k = P$
then show *restrict k P* $\longleftrightarrow$
Formula-Operations.satisfies-gen Extend Length satisfies0 (λ- - -. *True*) $\mathfrak{A}$ (*Restrict k i*)
unfolding *restrict-def Restrict-def*
by (*cases k*) (*auto simp*: *Formula-Operations.satisfies-gen.simps*)
next
fix *i* $\mathfrak{A}$ *k P*
assume $i^{\mathfrak{A}}k = P$
then show *restrict k P* $\longleftrightarrow$
Formula-Operations.satisfies-gen Extend Length satisfies0 (λ- *P n*. *len P* $\leq$ *n*) $\mathfrak{A}$ (*Restrict k i*)
unfolding *restrict-def Restrict-def*
by (*cases k*) (*auto simp*: *Formula-Operations.satisfies-gen.simps*)
qed (*auto simp*: *Extend-commute-unsafe downshift-def upshift-def fimage-iff Suc-le-eq len-def*
eval-def cut-def len-downshift-helper
dest: *fMax-ge fMax-ffilter-less fMax-boundedD fsubset-fsingletonD*
split: *order.splits if-splits*)

lemma [*code*]: *check-eqv idx r s* =
((*ws1s-wf idx r* $\wedge$ *ws1s-left-formula r*) $\wedge$ (*ws1s-wf idx s* $\wedge$ *ws1s-left-formula s*) $\wedge$
(*case rtrancl-while* (λ(*p*, *q*). *final idx p* = *final idx q*)
(λ(*p*, *q*). *map* (λ*a*. (*norm* (*deriv lderiv0 a p*), *norm* (*deriv lderiv0 a q*))) (σ *idx*))
(*norm* (*RESTRICT r*), *norm* (*RESTRICT s*)) *of*
None $\Rightarrow$ *False*
| *Some* ([], *x*) $\Rightarrow$ *True*

| *Some* (*a* # *list*, *x*) ⇒ *False*))
unfolding *check-eqv-def WS1S.check-eqv-def* ..

definition *while* **where** [*code del*, *code-abbrev*]: *while idx* φ = *while-default* (*fut-default idx* φ)
declare *while-default-code*[*of fut-default idx* φ **for** *idx* φ, *folded while-def*, *code*]

export-code *check-eqv* **in** *SML* **module-name** *WS1S-Generated*

lemma *check-eqv-sound*:
$[\![\#_V\ \mathfrak{A} = idx;\ check\text{-}eqv\ idx\ \varphi\ \psi]\!] \Longrightarrow (WS1S.sat\ \mathfrak{A}\ \varphi \longleftrightarrow WS1S.sat\ \mathfrak{A}\ \psi)$
unfolding *check-eqv-def* **by** (*rule WS1S.check-eqv-soundness*)

lemma *bounded-check-eqv-sound*:
$[\![\#_V\ \mathfrak{A} = idx;\ bounded\text{-}check\text{-}eqv\ idx\ \varphi\ \psi]\!] \Longrightarrow (WS1S.sat_b\ \mathfrak{A}\ \varphi \longleftrightarrow WS1S.sat_b\ \mathfrak{A}\ \psi)$
unfolding *bounded-check-eqv-def* **by** (*rule WS1S.bounded-check-eqv-soundness*)

end

11 Examples

abbreviation (*input*) *FImp* **where** *FImp* φ ψ ≡ *FOr* (*FNot* φ) ψ
abbreviation *FIff* **where** *FIff* φ ψ ≡ *FAnd* (*FImp* φ ψ) (*FImp* ψ φ)
abbreviation *FBall* **where** *FBall X* φ ≡ *FAll FO* (*FImp* (*FBase* (*In 0 X*)) φ)

abbreviation *SUBSET* **where** *SUBSET X Y* ≡ *FBall X* (*FBase* (*In 0 Y*))
abbreviation *EQ* **where** *EQ X Y* ≡ *FAnd* (*SUBSET X Y*) (*SUBSET Y X*)
abbreviation *First* **where** *First x* ≡ *FNot* (*FEx FO* (*FBase* (*Less 0* (*x+1*))))
abbreviation *Last* **where** *Last x* ≡ *FNot* (*FEx FO* (*FBase* (*Less* (*x+1*) *0*)))
abbreviation *Succ* **where** *Succ sucx x* ≡ *FAnd* (*FBase* (*Less x sucx*)) (*FNot* (*FEx FO* (*FAnd* (*FBase* (*Less* (*x+1*) *0*)) (*FBase* (*Less 0* (*sucx+1*))))))

definition *Thm idx* φ = *check-eqv idx* φ (*FBool True*)

export-code *Thm* **in** *SML* **module-name** *Thm*

abbreviation *FTrue* (⊤) **where** *FTrue* ≡ *FBool True*
abbreviation *FFalse* (⊥) **where** *FFalse* ≡ *FBool False*

notation *FImp* (**infixr** −−> *25*)
notation (**output**) *FO* (*1*)
notation (**output**) *SO* (*2*)
notation *FEx* (∃$_{-}$ - [*10*] *10*)
notation *FAll* (∀$_{-}$ - [*10*] *10*)
notation *FNot* (¬ - [*40*] *40*)
notation *FOr* (**infixr** ∨ *30*)
notation *FAnd* (**infixr** ∧ *35*)
abbreviation *FLess* (x_{-} < x_{-} [*65*, *66*] *65*) **where** *FLess m1 m2* ≡ *FBase* (*Less*

m1 m2)
abbreviation *FIn* ($x_- \in X_-$ [*65*, *66*] *65*) **where** *FIn m M* ≡ *FBase* (*In m M*)
abbreviation *FQ* ([x_-] [*66*] *65*) **where** *FQ m* ≡ *FBase* (*Q m*)

definition *M2L* = (*FEx SO* (*FAll FO* (*FBase* (*In 0 0*))) :: *formula*)

definition Φ = (*FAll FO* (*FEx FO* (*FBase* (*Less 1 0*))) :: *formula*)

lemma *Thm* (*Abs-idx* (*0*, *1*)) (*FNot M2L*)
  by *eval*

lemma *Thm* (*Abs-idx* (*0*, *0*)) Φ
  by *eval*

abbreviation *Globally* (□- [*40*] *40*) **where** *Globally P* == %*n*. *FAll FO* (*FImp* (*FNot* (*FBase* (*Less* (*n*+*1*) *0*))) (*P 0*))
abbreviation *Future* (◇- [*40*] *40*) **where** *Future P* == %*n*. *FEx FO* (*FAnd* (*FNot* (*FBase* (*Less* (*n*+*1*) *0*))) (*P 0*))
abbreviation *IMP* (**infixr** → *50*) **where** *IMP P1 P2* == %*n*. *FImp* (*P1 n*) (*P2 n*)

definition Ψ :: *nat* ⇒ *formula* **where**
  Ψ *n* = *FAll FO* (((□(*foldr* (λ*i* φ. (λ*m*. *FBase* (*In m i*)) → φ) [*0*..<*n*] (λ*m*. *FBase* (*In m n*)))) →
    foldr (λ*i* φ. (□(λ*m*. *FBase* (*In m i*))) → φ) [*0*..<*n*] (□(λ*m*. *FBase* (*In m n*)))) *0*)

lemma *Thm* (*Abs-idx* (*0*, *1*)) (Ψ *0*) **by** *eval*
lemma *Thm* (*Abs-idx* (*0*, *2*)) (Ψ *1*) **by** *eval*
lemma *Thm* (*Abs-idx* (*0*, *3*)) (Ψ *2*) **by** *eval*
lemma *Thm* (*Abs-idx* (*0*, *4*)) (Ψ *3*) **by** *eval*
lemma *Thm* (*Abs-idx* (*0*, *5*)) (Ψ *4*) **by** *eval*
lemma *Thm* (*Abs-idx* (*0*, *6*)) (Ψ *5*) **by** *eval*
lemma *Thm* (*Abs-idx* (*0*, *11*)) (Ψ *10*) **by** *eval*
lemma *Thm* (*Abs-idx* (*0*, *16*)) (Ψ *15*) **by** *eval*