Derivatives of Logical Formulas

Dmitriy Traytel

October 13, 2025

Abstract

We formalize new decision procedures for WS1S, M2L(Str), and Presburger Arithmetics. Formulas of these logics denote regular languages. Unlike traditional decision procedures, we do *not* translate formulas into automata (nor into regular expressions), at least not explicitly. Instead we devise notions of derivatives (inspired by Brzozowski derivatives for regular expressions) that operate on formulas directly and compute a syntactic bisimulation using these derivatives. The treatment of Boolean connectives and quantifiers is uniform for all mentioned logics and is abstracted into a locale. This locale is then instantiated by different atomic formulas and their derivatives (which may differ even for the same logic under different encodings of interpretations as formal words).

The WS1S instance is described in the draft paper *A Coalgebraic Decision Procedure for WS1S* [1] by the author.

Contents

1 Equivalence Framework **1**
1.1 Abstract Deterministic Automaton 2
1.2 The overall procedure 4
1.3 Abstract Deterministic Finite Automaton 4

2 Derivatives of Abstract Formulas **6**
2.1 Preliminaries 6
2.2 Abstract formulas 6
2.3 Normalization 17
2.4 Derivatives of Formulas 19
2.5 Finiteness of Derivatives Modulo ACI 20
2.6 Emptiness Check 21
2.7 Restrictions 24

3 WS1S Interpretations **26**

[1] http://www21.in.tum.de/~traytel/papers/ws1s_derivatives/index.html

4 Concrete Atomic WS1S Formulas (Minimum Semantics for FO Variables) 34

5 Concrete Atomic WS1S Formulas (Singleton Semantics for FO Variables) 43

6 Concrete Atomic Presburger Formulas 46

7 Comparing WS1S Formulas with Presburger Formulas 56

8 Nameful WS1S Formulas 58

1 Equivalence Framework

coinductive *rel-language* **where**
$[\![\mathfrak{o}\ L = \mathfrak{o}\ K; \bigwedge a\ b.\ R\ a\ b \Longrightarrow \textit{rel-language}\ R\ (\mathfrak{d}\ L\ a)\ (\mathfrak{d}\ K\ b)]\!] \Longrightarrow \textit{rel-language}\ R\ L\ K$

declare *rel-language.coinduct*[*consumes 1*, *case-names Lang*, *coinduct pred*]

lemma *rel-language-alt*:
$\textit{rel-language}\ R\ L\ K = \textit{rel-fun}\ (\textit{list-all2}\ R)\ (=)\ (\textit{in-language}\ L)\ (\textit{in-language}\ K)$
⟨*proof*⟩

lemma *rel-language-eq*: $\textit{rel-language}\ (=) = (=)$
⟨*proof*⟩

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

lemma *in-language-*$\mathfrak{d}s$: $\textit{in-language}\ (\mathfrak{d}s\ w\ L)\ v \longleftrightarrow \textit{in-language}\ L\ (w\ @\ v)$
⟨*proof*⟩

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

lemma *in-language-to-language*[*simp*]: $\textit{in-language}\ (\textit{to-language}\ L)\ w \longleftrightarrow w \in L$
⟨*proof*⟩

lemma *rtrancl-fold-product*:
shows $\{((r, s), (f\ a\ r, g\ b\ s)) \mid a\ b\ r\ s.\ a \in A \wedge b \in B \wedge R\ a\ b\}\hat{}* =$
$\{((r, s), (\textit{fold}\ f\ w1\ r, \textit{fold}\ g\ w2\ s)) \mid w1\ w2\ r\ s.\ w1 \in \textit{lists}\ A \wedge w2 \in \textit{lists}\ B \wedge \textit{list-all2}\ R\ w1\ w2\}$
(**is** $?L = ?R$)
⟨*proof*⟩

lemma *rtrancl-fold-product1*:
shows $\{(r, s).\ \exists\, a \in A.\ s = f\ a\ r\}\hat{}* = \{(r, s).\ \exists\, a \in \textit{lists}\ A.\ s = \textit{fold}\ f\ a\ r\}$ (**is**

$?L = ?R$)
$\langle proof \rangle$

lemma *lang-eq-ext-Nil-fold-Deriv*:
fixes $K\ L\ A\ R$
assumes
$\bigwedge w.\ in\text{-}language\ K\ w \Longrightarrow w \in lists\ A$
$\bigwedge w.\ in\text{-}language\ L\ w \Longrightarrow w \in lists\ B$
$\bigwedge a\ b.\ R\ a\ b \Longrightarrow a \in A \longleftrightarrow b \in B$
defines $\mathfrak{B} \equiv \{(\mathfrak{d}s\ w1\ K,\ \mathfrak{d}s\ w2\ L) \mid w1\ w2.\ w1 \in lists\ A \wedge w2 \in lists\ B \wedge list\text{-}all2\ R\ w1\ w2\}$
shows $rel\text{-}language\ R\ K\ L \longleftrightarrow (\forall (K,\ L) \in \mathfrak{B}.\ \mathfrak{o}\ K \longleftrightarrow \mathfrak{o}\ L)$
$\langle proof \rangle$

1.1 Abstract Deterministic Automaton

locale DA =
fixes $alphabet :: 'a\ list$
fixes $init :: 't \Rightarrow 's$
fixes $delta :: 'a \Rightarrow 's \Rightarrow 's$
fixes $accept :: 's \Rightarrow bool$
fixes $wellformed :: 's \Rightarrow bool$
fixes $Language :: 's \Rightarrow 'a\ language$
fixes $wf :: 't \Rightarrow bool$
fixes $Lang :: 't \Rightarrow 'a\ language$
assumes *distinct-alphabet*: *distinct alphabet*
assumes *Language-init*: $wf\ t \Longrightarrow Language\ (init\ t) = Lang\ t$
assumes *wellformed-init*: $wf\ t \Longrightarrow wellformed\ (init\ t)$
assumes *Language-alphabet*: $[\![wellformed\ s;\ in\text{-}language\ (Language\ s)\ w]\!] \Longrightarrow w \in lists\ (set\ alphabet)$
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 *this*: *DA alphabet init delta accept wellformed Language wf Lang* $\langle proof \rangle$

lemma *wellformed-deltas*:
$[\![wellformed\ s;\ w \in lists\ (set\ alphabet)]\!] \Longrightarrow wellformed\ (fold\ delta\ w\ s)$
$\langle proof \rangle$

lemma *Language-deltas*:
$[\![wellformed\ s;\ w \in lists\ (set\ alphabet)]\!] \Longrightarrow Language\ (fold\ delta\ w\ s) = \mathfrak{d}s\ w\ (Language\ s)$
$\langle proof \rangle$

Auxiliary functions:

definition *reachable* :: $'a$ *list* $\Rightarrow$ $'s$ $\Rightarrow$ $'s$ *set* **where**
reachable as s = *snd* (*the* (*rtrancl-while* (λ-. *True*) (λs. *map* (λa. *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 = λ((*ws*, *Z*), *A*). *ws* $\neq$ [];
step = λ((*ws*, *Z*), *A*).
(*let s* = *hd ws*;
new-edges = *map* (λa. ((*s*, *a*), *delta a s*)) *as*;
new = *remdups* (*filter* (λss. *ss* $\notin$ *Z*) (*map snd new-edges*))
in ((*new* @ *tl ws*, *set new* $\cup$ *Z*), *set new-edges* $\cup$ *A*))
in while-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-language* (*Language s*) *w*
$\langle$*proof*$\rangle$

end

locale *DAs* =
M: *DA alphabet1 init1 delta1 accept1 wellformed1 Language1 wf1 Lang1* +
N: *DA alphabet2 init2 delta2 accept2 wellformed2 Language2 wf2 Lang2*
for *alphabet1* :: $'a1$ *list* **and** *init1* :: $'t1$ $\Rightarrow$ $'s1$ **and** *delta1 accept1 wellformed1 Language1 wf1 Lang1*
and *alphabet2* :: $'a2$ *list* **and** *init2* :: $'t2$ $\Rightarrow$ $'s2$ **and** *delta2 accept2 wellformed2 Language2 wf2 Lang2* +
fixes *letter-eq* :: $'a1$ $\Rightarrow$ $'a2$ $\Rightarrow$ *bool*
assumes *letter-eq*: $\bigwedge a\ b$. *letter-eq a b* $\Longrightarrow$ *a* $\in$ *set alphabet1* $\longleftrightarrow$ *b* $\in$ *set alphabet2*
begin

abbreviation *step* **where**
step $\equiv$ (λ(*p*, *q*). *map* (λ(*a*, *b*). (*delta1 a p*, *delta2 b q*))
(*filter* (*case-prod letter-eq*) (*List.product alphabet1 alphabet2*)))

abbreviation *closure* :: $'s1$ $*$ $'s2$ $\Rightarrow$ (($'s1$ $*$ $'s2$) *list* $*$ ($'s1$ $*$ $'s2$) *set*) *option* **where**
closure $\equiv$ *rtrancl-while* (λ(*p*, *q*). *accept1 p* = *accept2 q*) *step*

theorem *closure-sound-complete*:
assumes *wf*: *wf1 r wf2 s*
and *result*: *closure* (*init1 r*, *init2 s*) = *Some* (*ws*, *R*) (**is** *closure* (*?r*, *?s*) = -)
shows *ws* = [] $\longleftrightarrow$ *rel-language letter-eq* (*Lang1 r*) (*Lang2 s*)

$\langle proof \rangle$

1.2 The overall procedure

definition *check-eqv* :: $'t1 \Rightarrow 't2 \Rightarrow bool$ **where**
$check\text{-}eqv\ r\ s = (wf1\ r \wedge wf2\ s \wedge (case\ closure\ (init1\ r,\ init2\ s)\ of\ Some([],\ \text{-}) \Rightarrow True \mid \text{-} \Rightarrow False))$

lemma *soundness*:
assumes *check-eqv r s* **shows** *rel-language letter-eq* (*Lang1 r*) (*Lang2 s*)
$\langle proof \rangle$

end

1.3 Abstract Deterministic Finite Automaton

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

lemma *finite-rtrancl-delta-Image1*:
$wellformed\ r \Longrightarrow finite\ (\{(r,\ s).\ \exists\, a \in set\ alphabet.\ s = delta\ a\ r\}\hat{}* \text{ `` } \{r\})$
$\langle proof \rangle$

lemma
assumes *wellformed r set as* $\subseteq$ *set alphabet*
shows *reachable*: $reachable\ as\ r = \{fold\ delta\ w\ r \mid w.\ w \in lists\ (set\ as)\}$
and *finite-reachable*: *finite* (*reachable as r*)
$\langle proof \rangle$

end

locale *DFAs* =
M: *DFA alphabet1 init1 delta1 accept1 wellformed1 Language1 wf1 Lang1* +
N: *DFA alphabet2 init2 delta2 accept2 wellformed2 Language2 wf2 Lang2*
for *alphabet1* :: $'a1$ *list* **and** *init1* :: $'t1 \Rightarrow 's1$ **and** *delta1 accept1 wellformed1 Language1 wf1 Lang1*
and *alphabet2* :: $'a2$ *list* **and** *init2* :: $'t2 \Rightarrow 's2$ **and** *delta2 accept2 wellformed2 Language2 wf2 Lang2* +
fixes *letter-eq* :: $'a1 \Rightarrow 'a2 \Rightarrow bool$
assumes *letter-eq*: $\bigwedge a\ b.\ letter\text{-}eq\ a\ b \Longrightarrow a \in set\ alphabet1 \longleftrightarrow b \in set\ alphabet2$
begin

interpretation *DAs* $\langle proof \rangle$

lemma *finite-rtrancl-delta-Image*:
$[\![wellformed1\ r;\ wellformed2\ s]\!] \Longrightarrow$
$finite\ (\{((r,\ s),\ (delta1\ a\ r,\ delta2\ b\ s)) \mid\ a\ b\ r\ s.$

$a \in set\ alphabet1 \wedge b \in set\ alphabet2 \wedge R\ a\ b\}\hat{}* \text{ `` } \{(r, s)\})$
⟨proof⟩

lemma *termination*:
assumes *wellformed1 r wellformed2 s*
shows $\exists st.\ closure\ (r, s) = Some\ st$ (**is** $\exists$ -. *closure ?i* = -)
⟨proof⟩

lemma *completeness*:
assumes *wf1 r wf2 s rel-language letter-eq* (*Lang1 r*) (*Lang2 s*) **shows** *check-eqv r s*
⟨proof⟩

end

sublocale *DA* < *DAs*
alphabet init delta accept wellformed Language wf Lang
alphabet init delta accept wellformed Language wf Lang (=)
⟨proof⟩

sublocale *DFA* < *DFAs*
alphabet init delta accept wellformed Language wf Lang
alphabet init delta accept wellformed Language wf Lang (=)
⟨proof⟩

lemma (**in** *DA*) *step-alt*: $step = (\lambda(p, q).\ map\ (\lambda a.\ (delta\ a\ p, delta\ a\ q))\ alphabet)$
⟨proof⟩

2 Derivatives of Abstract Formulas

2.1 Preliminaries

lemma *pred-Diff-0*[*simp*]: $0 \notin A \Longrightarrow i \in (\lambda x.\ x - Suc\ 0)\ \text{`}\ A \longleftrightarrow Suc\ i \in A$
⟨proof⟩

lemma *funpow-cycle-mult*: $(f \widehat{\ }\widehat{\ } k)\ x = x \Longrightarrow (f \widehat{\ }\widehat{\ } (m * k))\ x = x$
⟨proof⟩

lemma *funpow-cycle*: $(f \widehat{\ }\widehat{\ } k)\ x = x \Longrightarrow (f \widehat{\ }\widehat{\ } l)\ x = (f \widehat{\ }\widehat{\ } (l\ mod\ k))\ x$
⟨proof⟩

lemma *funpow-cycle-offset*:
fixes $f :: 'a \Rightarrow 'a$
assumes $(f \widehat{\ }\widehat{\ } k)\ x = (f \widehat{\ }\widehat{\ } i)\ x\ i \le k\ i \le l$
shows $(f \widehat{\ }\widehat{\ } l)\ x = (f \widehat{\ }\widehat{\ } ((l - i)\ mod\ (k - i) + i))\ x$
⟨proof⟩

lemma *in-set-tlD*: $x \in set\ (tl\ xs) \Longrightarrow x \in set\ xs$
⟨proof⟩

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

2.2 Abstract formulas

datatype (*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*
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-list-of-set*
definition *NFAND* = *nFAND o sorted-list-of-set*

fun *disjuncts* **where**
  disjuncts (*FOr* φ ψ) = *disjuncts* φ ∪ *disjuncts* ψ
| *disjuncts* φ = {φ}

fun *conjuncts* **where**
  conjuncts (*FAnd* φ ψ) = *conjuncts* φ ∪ *conjuncts* ψ
| *conjuncts* φ = {φ}

fun *disjuncts-list* **where**
  disjuncts-list (*FOr* φ ψ) = *disjuncts-list* φ @ *disjuncts-list* ψ
| *disjuncts-list* φ = [φ]

fun *conjuncts-list* **where**
  conjuncts-list (*FAnd* φ ψ) = *conjuncts-list* φ @ *conjuncts-list* ψ
| *conjuncts-list* φ = [φ]

lemma *finite-juncts*[*simp*]: *finite* (*disjuncts* φ) *finite* (*conjuncts* φ)
  and *nonempty-juncts*[*simp*]: *disjuncts* φ ≠ {} *conjuncts* φ ≠ {}
  ⟨*proof*⟩

lemma *juncts-eq-set-juncts-list*:

$disjuncts\ \varphi = set\ (disjuncts\text{-}list\ \varphi)$
$conjuncts\ \varphi = set\ (conjuncts\text{-}list\ \varphi)$
$\langle proof \rangle$

lemma *notin-juncts*:
$[\![\psi \in disjuncts\ \varphi;\ is\text{-}FOr\ \psi]\!] \Longrightarrow False$
$[\![\psi \in conjuncts\ \varphi;\ is\text{-}FAnd\ \psi]\!] \Longrightarrow False$
$\langle proof \rangle$

lemma *juncts-list-singleton*:
$\neg\ is\text{-}FOr\ \varphi \Longrightarrow disjuncts\text{-}list\ \varphi = [\varphi]$
$\neg\ is\text{-}FAnd\ \varphi \Longrightarrow conjuncts\text{-}list\ \varphi = [\varphi]$
$\langle proof \rangle$

lemma *juncts-singleton*:
$\neg\ is\text{-}FOr\ \varphi \Longrightarrow disjuncts\ \varphi = \{\varphi\}$
$\neg\ is\text{-}FAnd\ \varphi \Longrightarrow conjuncts\ \varphi = \{\varphi\}$
$\langle proof \rangle$

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

primrec *norm-ACI* (‹⟨-⟩›) **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**
$nf\text{-}ACI\ (FOr\ \psi 1\ \psi 2) = (\neg\ is\text{-}FOr\ \psi 1 \wedge (let\ \varphi s = \psi 1\ \#\ disjuncts\text{-}list\ \psi 2\ in$
$sorted\ \varphi s \wedge distinct\ \varphi s \wedge nf\text{-}ACI\ \psi 1 \wedge nf\text{-}ACI\ \psi 2))$
$|\ nf\text{-}ACI\ (FAnd\ \psi 1\ \psi 2) = (\neg\ is\text{-}FAnd\ \psi 1 \wedge (let\ \varphi s = \psi 1\ \#\ conjuncts\text{-}list\ \psi 2\ in$
$sorted\ \varphi s \wedge distinct\ \varphi s \wedge nf\text{-}ACI\ \psi 1 \wedge nf\text{-}ACI\ \psi 2))$
$|\ nf\text{-}ACI\ (FNot\ \varphi) = nf\text{-}ACI\ \varphi$
$|\ nf\text{-}ACI\ (FEx\ k\ \varphi) = nf\text{-}ACI\ \varphi$
$|\ nf\text{-}ACI\ (FAll\ k\ \varphi) = nf\text{-}ACI\ \varphi$
$|\ nf\text{-}ACI\ \varphi = True$

lemma *nf-ACI-D*:
$nf\text{-}ACI\ \varphi \Longrightarrow sorted\ (disjuncts\text{-}list\ \varphi)$
$nf\text{-}ACI\ \varphi \Longrightarrow sorted\ (conjuncts\text{-}list\ \varphi)$
$nf\text{-}ACI\ \varphi \Longrightarrow distinct\ (disjuncts\text{-}list\ \varphi)$
$nf\text{-}ACI\ \varphi \Longrightarrow distinct\ (conjuncts\text{-}list\ \varphi)$
$nf\text{-}ACI\ \varphi \Longrightarrow list\text{-}all\ nf\text{-}ACI\ (disjuncts\text{-}list\ \varphi)$
$nf\text{-}ACI\ \varphi \Longrightarrow list\text{-}all\ nf\text{-}ACI\ (conjuncts\text{-}list\ \varphi)$
$\langle proof \rangle$

lemma *disjuncts-list-nFOR*:
$[\![$*list-all* $(\lambda x.\ \neg$ *is-FOr* $x)$ φs; $\varphi s \neq [\,]]\!] \Longrightarrow$ *disjuncts-list* $($*nFOR* $\varphi s) = \varphi s$
⟨*proof*⟩

lemma *conjuncts-list-nFAND*:
$[\![$*list-all* $(\lambda x.\ \neg$ *is-FAnd* $x)$ φs; $\varphi s \neq [\,]]\!] \Longrightarrow$ *conjuncts-list* $($*nFAND* $\varphi s) = \varphi s$
⟨*proof*⟩

lemma *disjuncts-NFOR*:
$[\![$*finite* X; $X \neq \{\}$; $\forall x \in X.\ \neg$ *is-FOr* $x]\!] \Longrightarrow$ *disjuncts* $($*NFOR* $X) = X$
⟨*proof*⟩

lemma *conjuncts-NFAND*:
$[\![$*finite* X; $X \neq \{\}$; $\forall x \in X.\ \neg$ *is-FAnd* $x]\!] \Longrightarrow$ *conjuncts* $($*NFAND* $X) = X$
⟨*proof*⟩

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* $\varphi s)$
⟨*proof*⟩

lemma *nf-ACI-nFAND*:
$[\![$*sorted* φs; *distinct* φs; *list-all nf-ACI* φs; *list-all* $(\lambda x.\ \neg$ *is-FAnd* $x)$ $\varphi s]\!] \Longrightarrow$ *nf-ACI* $($*nFAND* $\varphi s)$
⟨*proof*⟩

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
⟨*proof*⟩

lemma *nf-ACI-norm-ACI*: *nf-ACI* $\langle\varphi\rangle$
⟨*proof*⟩

lemma *nFOR-Cons*: *nFOR* $(x \# xs) = ($*if* $xs = [\,]$ *then* x *else FOr* x $($*nFOR* $xs))$
⟨*proof*⟩

lemma *nFAND-Cons*: *nFAND* $(x \# xs) = ($*if* $xs = [\,]$ *then* x *else FAnd* x $($*nFAND* $xs))$
⟨*proof*⟩

lemma *nFOR-disjuncts*: *nf-ACI* $\psi \Longrightarrow$ *nFOR* $($*disjuncts-list* $\psi) = \psi$
⟨*proof*⟩

lemma *nFAND-conjuncts*: *nf-ACI* $\psi \Longrightarrow$ *nFAND* $($*conjuncts-list* $\psi) = \psi$
⟨*proof*⟩

lemma *NFOR-disjuncts*: *nf-ACI* $\psi \Longrightarrow$ *NFOR* $($*disjuncts* $\psi) = \psi$

⟨*proof*⟩

lemma *NFAND-conjuncts*: *nf-ACI* $\psi \Longrightarrow$ *NFAND* (*conjuncts* ψ) = ψ
⟨*proof*⟩

lemma *norm-ACI-if-nf-ACI*: *nf-ACI* $\varphi \Longrightarrow \langle\varphi\rangle = \varphi$
⟨*proof*⟩

lemma *norm-ACI-idem*: $\langle\langle\varphi\rangle\rangle = \langle\varphi\rangle$
⟨*proof*⟩

lemma *norm-ACI-juncts*:
nf-ACI $\varphi \Longrightarrow$ *norm-ACI* ‘ *disjuncts* φ = *disjuncts* φ
nf-ACI $\varphi \Longrightarrow$ *norm-ACI* ‘ *conjuncts* φ = *conjuncts* φ
⟨*proof*⟩

lemma
norm-ACI-NFOR: *nf-ACI* $\varphi \Longrightarrow \varphi$ = *NFOR* (*norm-ACI* ‘ *disjuncts* φ) **and**
norm-ACI-NFAND: *nf-ACI* $\varphi \Longrightarrow \varphi$ = *NFAND* (*norm-ACI* ‘ *conjuncts* φ)
⟨*proof*⟩

locale *Formula-Operations* =
fixes *TYPEVARS* :: 'a :: *linorder* × 'i × 'k :: {*linorder*, *enum*} × 'n × 'x × 'v

and *SUC* :: 'k ⇒ 'n ⇒ 'n
and *LESS* :: 'k ⇒ *nat* ⇒ 'n ⇒ *bool*

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

and *extend* :: 'k ⇒ *bool* ⇒ 'x ⇒ 'x
and *size* :: 'x ⇒ 'n
and *zero* :: 'n ⇒ 'x
and *alphabet* :: 'n ⇒ 'x *list*

and *eval* :: 'v ⇒ *nat* ⇒ *bool*
and *downshift* :: 'v ⇒ 'v
and *upshift* :: 'v ⇒ 'v
and *add* :: *nat* ⇒ 'v ⇒ 'v
and *cut* :: *nat* ⇒ 'v ⇒ 'v

and *len* :: $'v \Rightarrow nat$

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

and *lformula0* :: $'a \Rightarrow bool$
and *FV0* :: $'k \Rightarrow 'a \Rightarrow nat\ set$
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** $\langle \models_0 \rangle$ *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**
$FV\ (FBool\ \text{-})\ k = \{\}$
$|\ FV\ (FBase\ a)\ k = FV0\ k\ a$
$|\ FV\ (FNot\ \varphi)\ k = FV\ \varphi\ k$
$|\ FV\ (FOr\ \varphi\ \psi)\ k = FV\ \varphi\ k \cup FV\ \psi\ k$
$|\ FV\ (FAnd\ \varphi\ \psi)\ k = FV\ \varphi\ k \cup FV\ \psi\ k$
$|\ FV\ (FEx\ k'\ \varphi)\ k = (if\ k' = k\ then\ (\lambda x.\ x - 1)\ `\ (FV\ \varphi\ k - \{0\})\ else\ FV\ \varphi\ k)$
$|\ FV\ (FAll\ k'\ \varphi)\ k = (if\ k' = k\ then\ (\lambda x.\ x - 1)\ `\ (FV\ \varphi\ k - \{0\})\ else\ FV\ \varphi\ k)$

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

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

primrec *lformula* :: $('a, 'k)$ *aformula* $\Rightarrow bool$ **where**
$lformula\ (FBool\ \text{-}) = True$

| *lformula* (*FBase a*) = *lformula0 a*
| *lformula* (*FNot* φ) = *lformula* φ
| *lformula* (*FOr* φ ψ) = (*lformula* φ $\wedge$ *lformula* ψ)
| *lformula* (*FAnd* φ ψ) = (*lformula* φ $\wedge$ *lformula* ψ)
| *lformula* (*FEx k* φ) = *lformula* φ
| *lformula* (*FAll k* φ) = *lformula* φ

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* φ) = *FNot* (*decr k l* φ)
| *decr k l* (*FOr* φ ψ) = *FOr* (*decr k l* φ) (*decr k l* ψ)
| *decr k l* (*FAnd* φ ψ) = *FAnd* (*decr k l* φ) (*decr k l* ψ)
| *decr k l* (*FEx* k' φ) = *FEx* k' (*decr k* (*if* $k = k'$ *then Suc l else l*) φ)
| *decr k l* (*FAll* k' φ) = *FAll* k' (*decr k* (*if* $k = k'$ *then Suc l else l*) φ)

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* φ) = ($\neg$ *satisfies-gen r* $\mathfrak{A}$ φ)
| *satisfies-gen r* $\mathfrak{A}$ (*FOr* φ_1 φ_2) = (*satisfies-gen r* $\mathfrak{A}$ φ_1 $\vee$ *satisfies-gen r* $\mathfrak{A}$ φ_2)
| *satisfies-gen r* $\mathfrak{A}$ (*FAnd* φ_1 φ_2) = (*satisfies-gen r* $\mathfrak{A}$ φ_1 $\wedge$ *satisfies-gen r* $\mathfrak{A}$ φ_2)
| *satisfies-gen r* $\mathfrak{A}$ (*FEx k* φ) = ($\exists P.$ *r k P* (*Length* $\mathfrak{A}$) $\wedge$ *satisfies-gen r* (*Extend k 0* $\mathfrak{A}$ *P*) φ)
| *satisfies-gen r* $\mathfrak{A}$ (*FAll k* φ) = ($\forall P.$ *r k P* (*Length* $\mathfrak{A}$) $\longrightarrow$ *satisfies-gen r* (*Extend k 0* $\mathfrak{A}$ *P*) φ)

abbreviation *satisfies* (**infix** ‹$\models$› *50*) **where**
$\mathfrak{A} \models \varphi \equiv$ *satisfies-gen* (λ- - -. *True*) $\mathfrak{A}$ φ

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

abbreviation *sat-vars-gen* **where**
sat-vars-gen r K $\mathfrak{A}$ φ $\equiv$
satisfies-gen (λ*k P n. restrict k P* $\wedge$ *r k P n*) $\mathfrak{A}$ φ $\wedge$ ($\forall k \in K. \forall x \in FV\ \varphi\ k.$ *restrict k* ($x^{\mathfrak{A}}k$))

definition *sat* **where**
sat $\mathfrak{A}$ φ $\equiv$ *sat-vars-gen* (λ- - -. *True*) *UNIV* $\mathfrak{A}$ φ

definition sat_b **where**
sat_b $\mathfrak{A}$ φ $\equiv$ *sat-vars-gen* (λ- *P n. len P* $\leq$ *n*) *UNIV* $\mathfrak{A}$ φ

fun *RESTR* **where**
RESTR (*FOr* φ ψ) = *FOr* (*RESTR* φ) (*RESTR* ψ)
| *RESTR* (*FAnd* φ ψ) = *FAnd* (*RESTR* φ) (*RESTR* ψ)
| *RESTR* (*FNot* φ) = *FNot* (*RESTR* φ)

```
| RESTR (FEx k φ) = FEx k (FAnd (Restrict k 0) (RESTR φ))
| RESTR (FAll k φ) = FAll k (FOr (FNot (Restrict k 0)) (RESTR φ))
| RESTR φ = φ

abbreviation RESTRICT-VARS where
  RESTRICT-VARS ks V φ ≡
    foldr (%k φ. foldr (λx φ. FAnd (Restrict k x) φ) (V k) φ) ks (RESTR φ)

definition RESTRICT where
  RESTRICT φ ≡ RESTRICT-VARS Enum.enum (sorted-list-of-set o FV φ) φ

primrec nullable :: ('a, 'k) aformula ⇒ bool where
  nullable (FBool b) = b
| nullable (FBase a) = nullable0 a
| nullable (FNot φ) = (¬ nullable φ)
| nullable (FOr φ ψ) = (nullable φ ∨ nullable ψ)
| nullable (FAnd φ ψ) = (nullable φ ∧ nullable ψ)
| nullable (FEx k φ) = nullable φ
| nullable (FAll k φ) = nullable φ

fun nFOr :: ('a, 'k) aformula ⇒ ('a, 'k) aformula ⇒ ('a, 'k) aformula where
  nFOr (FBool b1) (FBool b2) = FBool (b1 ∨ b2)
| nFOr (FBool b) ψ = (if b then FBool True else ψ)
| nFOr φ (FBool b) = (if b then FBool True else φ)
| nFOr (FOr φ1 φ2) ψ = nFOr φ1 (nFOr φ2 ψ)
| nFOr φ (FOr ψ1 ψ2) =
  (if φ = ψ1 then FOr ψ1 ψ2
  else if φ < ψ1 then FOr φ (FOr ψ1 ψ2)
  else FOr ψ1 (nFOr φ ψ2))
| nFOr φ ψ =
  (if φ = ψ then φ
  else if φ < ψ then FOr φ ψ
  else FOr ψ φ)

fun nFAnd :: ('a, 'k) aformula ⇒ ('a, 'k) aformula ⇒ ('a, 'k) aformula where
  nFAnd (FBool b1) (FBool b2) = FBool (b1 ∧ b2)
| nFAnd (FBool b) ψ = (if b then ψ else FBool False)
| nFAnd φ (FBool b) = (if b then φ else FBool False)
| nFAnd (FAnd φ1 φ2) ψ = nFAnd φ1 (nFAnd φ2 ψ)
| nFAnd φ (FAnd ψ1 ψ2) =
  (if φ = ψ1 then FAnd ψ1 ψ2
  else if φ < ψ1 then FAnd φ (FAnd ψ1 ψ2)
  else FAnd ψ1 (nFAnd φ ψ2))
| nFAnd φ ψ =
  (if φ = ψ then φ
  else if φ < ψ then FAnd φ ψ
  else FAnd ψ φ)

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

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*: $fold\ (deriv\ d0)\ xs\ (FBool\ b) = FBool\ b$
$\langle proof \rangle$

lemma *fold-deriv-FNot*:
$fold\ (deriv\ d0)\ xs\ (FNot\ \varphi) = FNot\ (fold\ (deriv\ d0)\ xs\ \varphi)$

$\langle proof \rangle$

lemma *fold-deriv-FOr*:
$fold\ (deriv\ d0)\ xs\ (FOr\ \varphi\ \psi) = FOr\ (fold\ (deriv\ d0)\ xs\ \varphi)\ (fold\ (deriv\ d0)\ xs\ \psi)$
$\langle proof \rangle$

lemma *fold-deriv-FAnd*:
$fold\ (deriv\ d0)\ xs\ (FAnd\ \varphi\ \psi) = FAnd\ (fold\ (deriv\ d0)\ xs\ \varphi)\ (fold\ (deriv\ d0)\ xs\ \psi)$
$\langle proof \rangle$

lemma *fold-deriv-FEx*:
$\{\langle fold\ (deriv\ d0)\ xs\ (FEx\ k\ \varphi)\rangle \mid xs.\ True\} \subseteq$
$\{FEx\ k\ \psi \mid \psi.\ nf\text{-}ACI\ \psi \wedge disjuncts\ \psi \subseteq (\bigcup xs.\ disjuncts\ \langle fold\ (deriv\ d0)\ xs\ \varphi\rangle)\}$
$\langle proof \rangle$

lemma *fold-deriv-FAll*:
$\{\langle fold\ (deriv\ d0)\ xs\ (FAll\ k\ \varphi)\rangle \mid xs.\ True\} \subseteq$
$\{FAll\ k\ \psi \mid \psi.\ nf\text{-}ACI\ \psi \wedge conjuncts\ \psi \subseteq (\bigcup xs.\ conjuncts\ \langle fold\ (deriv\ d0)\ xs\ \varphi\rangle)\}$
$\langle proof \rangle$

lemma *finite-norm-ACI-juncts*:
fixes $f :: ('a, 'k)\ aformula \Rightarrow ('a, 'k)\ aformula$
shows $finite\ B \Longrightarrow finite\ \{f\ \varphi \mid \varphi.\ nf\text{-}ACI\ \varphi \wedge disjuncts\ \varphi \subseteq B\}$
$finite\ B \Longrightarrow finite\ \{f\ \varphi \mid \varphi.\ nf\text{-}ACI\ \varphi \wedge conjuncts\ \varphi \subseteq B\}$
$\langle proof \rangle$

end

locale *Formula* = *Formula-Operations*
where $TYPEVARS = TYPEVARS$
for $TYPEVARS :: 'a :: linorder \times 'i \times 'k :: \{linorder, enum\} \times 'n \times 'x \times 'v +$

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

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*: $[\![Length\ \mathfrak{A} = 0;\ Length\ \mathfrak{B} = 0;\ \#_V\ \mathfrak{A} = \#_V\ \mathfrak{B}]\!] \Longrightarrow \mathfrak{A} = \mathfrak{B}$
and *ex-Length-0*: $\exists\mathfrak{A}.\ Length\ \mathfrak{A} = 0 \wedge \#_V\ \mathfrak{A} = idx$
and *Extend-commute-safe*: $[\![j \leq i;\ LEQ\ k\ i\ (\#_V\ \mathfrak{A})]\!] \Longrightarrow$
$Extend\ k\ j\ (Extend\ k\ i\ \mathfrak{A}\ P)\ Q = Extend\ k\ (Suc\ i)\ (Extend\ k\ j\ \mathfrak{A}\ Q)\ P$
and *Extend-commute-unsafe*: $k \neq k' \Longrightarrow$

$Extend\ k\ j\ (Extend\ k'\ i\ \mathfrak{A}\ P)\ Q = Extend\ k'\ i\ (Extend\ k\ j\ \mathfrak{A}\ Q)\ P$

and *assigns-Extend*: $LEQ\ k\ i\ (\#_V\ \mathfrak{A}) \Longrightarrow$
$m^{Extend\ k\ i\ \mathfrak{A}\ P}{}_{k'} = (if\ k = k'\ then\ (if\ m = i\ then\ P\ else\ dec\ i\ m^{\mathfrak{A}}k)\ else\ m^{\mathfrak{A}}k')$

and *assigns-SNOC-zero*: $LESS\ k\ m\ (\#_V\ \mathfrak{A}) \Longrightarrow m^{SNOC\ (zero\ (\#_V\ \mathfrak{A}))\ \mathfrak{A}}{}_k = m^{\mathfrak{A}}k$

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*: $\#_V\ \mathfrak{A} = size\ x \Longrightarrow len\ P \le Length\ (SNOC\ x\ \mathfrak{A}) \Longrightarrow$
$Extend\ k\ 0\ (SNOC\ x\ \mathfrak{A})\ P =$
$SNOC\ (extend\ k\ (if\ eval\ P\ (Length\ \mathfrak{A})\ then\ True\ else\ False)\ x)\ (Extend\ k\ 0\ \mathfrak{A}\ (cut\ (Length\ \mathfrak{A})\ P))$

and *CONS-inj*: $size\ x = \#_V\ \mathfrak{A} \Longrightarrow size\ y = \#_V\ \mathfrak{A} \Longrightarrow \#_V\ \mathfrak{A} = \#_V\ \mathfrak{B} \Longrightarrow$
$CONS\ x\ \mathfrak{A} = CONS\ y\ \mathfrak{B} \longleftrightarrow (x = y \land \mathfrak{A} = \mathfrak{B})$

and *CONS-surj*: $Length\ \mathfrak{A} \neq 0 \Longrightarrow \#_V\ \mathfrak{A} = idx \Longrightarrow$
$\exists\, x\ \mathfrak{B}.\ \mathfrak{A} = CONS\ x\ \mathfrak{B} \land \#_V\ \mathfrak{B} = idx \land size\ x = idx$

and *size-zero*: $size\ (zero\ idx) = idx$

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

and *distinct-alphabet*: $distinct\ (alphabet\ idx)$

and *alphabet-size*: $x \in set\ (alphabet\ idx) \longleftrightarrow size\ x = idx$

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*: $p \ge len\ P \Longrightarrow \neg\ eval\ P\ p$

and *len-cut-le*: $len\ (cut\ n\ P) \le n$

and *len-cut*: $len\ P \le n \Longrightarrow cut\ n\ P = P$

and *cut-add*: $cut\ n\ (add\ m\ P) = (if\ m \ge n\ then\ cut\ n\ P\ else\ add\ m\ (cut\ n\ P))$

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 *lformula0-decr0*: $lformula0\ \varphi \Longrightarrow lformula0\ (decr0\ k\ l\ \varphi)$

and *Extend-satisfies0*: $[\![\neg\ find0\ k\ i\ a;\ LESS\ k\ i\ (SUC\ k\ (\#_V\ \mathfrak{A}));\ lformula0\ a \lor len\ P \le Length\ \mathfrak{A}]\!] \Longrightarrow$
$Extend\ k\ i\ \mathfrak{A}\ P \models_0 a \longleftrightarrow \mathfrak{A} \models_0 decr0\ k\ i\ a$

and *nullable0-satisfies0*: $Length\ \mathfrak{A} = 0 \Longrightarrow nullable0\ a \longleftrightarrow \mathfrak{A} \models_0 a$

and *satisfies0-eqI*: $wf0\ (\#_V\ \mathfrak{B})\ a \Longrightarrow \#_V\ \mathfrak{A} = \#_V\ \mathfrak{B} \Longrightarrow lformula0\ a \Longrightarrow$

$(\bigwedge m\ k.\ LESS\ k\ m\ (\#_V\ \mathfrak{B}) \Longrightarrow m^{\mathfrak{A}}k = m^{\mathfrak{B}}k) \Longrightarrow \mathfrak{A} \models_0 a \longleftrightarrow \mathfrak{B} \models_0 a$
and *wf-lderiv0*: $[\![wf0\ idx\ a;\ lformula0\ a]\!] \Longrightarrow wf\ idx\ (lderiv0\ x\ a)$
and *lformula-lderiv0*: $lformula0\ a \Longrightarrow lformula\ (lderiv0\ x\ a)$
and *wf-rderiv0*: $wf0\ idx\ a \Longrightarrow wf\ idx\ (rderiv0\ x\ a)$
and *satisfies-lderiv0*:
$[\![wf0\ (\#_V\ \mathfrak{A})\ a;\ \#_V\ \mathfrak{A} = size\ x;\ lformula0\ a]\!] \Longrightarrow \mathfrak{A} \models lderiv0\ x\ a \longleftrightarrow CONS\ x\ \mathfrak{A} \models_0 a$
and *satisfies-bounded-lderiv0*:
$[\![wf0\ (\#_V\ \mathfrak{A})\ a;\ \#_V\ \mathfrak{A} = size\ x;\ lformula0\ a]\!] \Longrightarrow \mathfrak{A} \models_b lderiv0\ x\ a \longleftrightarrow CONS\ x\ \mathfrak{A} \models_0 a$
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*: $[\![wf0\ idx\ a;\ LESS\ k\ l\ idx]\!] \Longrightarrow find0\ k\ l\ a \longleftrightarrow l \in FV0\ k\ a$
and *finite-FV0*: $finite\ (FV0\ k\ a)$
and *wf0-FV0-LESS*: $[\![wf0\ idx\ a;\ v \in FV0\ k\ a]\!] \Longrightarrow LESS\ k\ v\ idx$
and *restrict-Restrict*: $i^{\mathfrak{A}}k = P \Longrightarrow restrict\ k\ P \longleftrightarrow satisfies\text{-}gen\ r\ \mathfrak{A}\ (Restrict\ k\ i)$
and *wf-Restrict*: $LESS\ k\ i\ idx \Longrightarrow wf\ idx\ (Restrict\ k\ i)$
and *lformula-Restrict*: $lformula\ (Restrict\ k\ i)$
and *finite-lderiv0*: $lformula0\ a \Longrightarrow finite\ \{fold\ lderiv\ xs\ (FBase\ a)\ |\ xs.\ True\}$
and *finite-rderiv0*: $finite\ \{fold\ rderiv\ xs\ (FBase\ a)\ |\ xs.\ True\}$

context *Formula*
begin

lemma *satisfies-eqI*:
$[\![wf\ (\#_V\ \mathfrak{A})\ \varphi;\ \#_V\ \mathfrak{A} = \#_V\ \mathfrak{B};\ \bigwedge m\ k.\ LESS\ k\ m\ (\#_V\ \mathfrak{A}) \Longrightarrow m^{\mathfrak{A}}k = m^{\mathfrak{B}}k;\ lformula\ \varphi]\!] \Longrightarrow$
$\mathfrak{A} \models \varphi \longleftrightarrow \mathfrak{B} \models \varphi$
⟨*proof*⟩

lemma *wf-decr*:
$[\![wf\ (SUC\ k\ idx)\ \varphi;\ LEQ\ k\ l\ idx;\ \neg\ find\ k\ l\ \varphi]\!] \Longrightarrow wf\ idx\ (decr\ k\ l\ \varphi)$
⟨*proof*⟩

lemma *lformula-decr*:
$lformula\ \varphi \Longrightarrow lformula\ (decr\ k\ l\ \varphi)$
⟨*proof*⟩

lemma *Extend-satisfies-decr*:
$[\![\neg\ find\ k\ i\ \varphi;\ LEQ\ k\ i\ (\#_V\ \mathfrak{A});\ lformula\ \varphi]\!] \Longrightarrow Extend\ k\ i\ \mathfrak{A}\ P \models \varphi \longleftrightarrow \mathfrak{A} \models decr\ k\ i\ \varphi$
⟨*proof*⟩

lemma *LEQ-SUC*: $k \neq k' \Longrightarrow LEQ\ k\ i\ (SUC\ k'\ idx) = LEQ\ k\ i\ idx$
⟨*proof*⟩

lemma *Extend-satisfies-bounded-decr*:
$[\![\neg\ find\ k\ i\ \varphi;\ LEQ\ k\ i\ (\#_V\ \mathfrak{A});\ len\ P \leq Length\ \mathfrak{A}]\!] \Longrightarrow$

Extend k i $\mathfrak{A}$ *P* $\models_b \varphi \longleftrightarrow \mathfrak{A} \models_b$ *decr k i* φ
⟨*proof*⟩

2.3 Normalization

lemma *wf-nFOr*:
wf idx (*FOr* φ ψ) $\Longrightarrow$ *wf idx* (*nFOr* φ ψ)
⟨*proof*⟩

lemma *wf-nFAnd*:
wf idx (*FAnd* φ ψ) $\Longrightarrow$ *wf idx* (*nFAnd* φ ψ)
⟨*proof*⟩

lemma *wf-nFEx*:
wf idx (*FEx b* φ) $\Longrightarrow$ *wf idx* (*nFEx b* φ)
⟨*proof*⟩

lemma *wf-nFAll*:
wf idx (*FAll b* φ) $\Longrightarrow$ *wf idx* (*nFAll b* φ)
⟨*proof*⟩

lemma *wf-nFNot*:
wf idx (*FNot* φ) $\Longrightarrow$ *wf idx* (*nFNot* φ)
⟨*proof*⟩

lemma *wf-norm*: *wf idx* φ $\Longrightarrow$ *wf idx* (*norm* φ)
⟨*proof*⟩

lemma *lformula-nFOr*:
lformula (*FOr* φ ψ) $\Longrightarrow$ *lformula* (*nFOr* φ ψ)
⟨*proof*⟩

lemma *lformula-nFAnd*:
lformula (*FAnd* φ ψ) $\Longrightarrow$ *lformula* (*nFAnd* φ ψ)
⟨*proof*⟩

lemma *lformula-nFEx*:
lformula (*FEx b* φ) $\Longrightarrow$ *lformula* (*nFEx b* φ)
⟨*proof*⟩

lemma *lformula-nFAll*:
lformula (*FAll b* φ) $\Longrightarrow$ *lformula* (*nFAll b* φ)
⟨*proof*⟩

lemma *lformula-nFNot*:
lformula (*FNot* φ) $\Longrightarrow$ *lformula* (*nFNot* φ)
⟨*proof*⟩

lemma *lformula-norm*: *lformula* φ $\Longrightarrow$ *lformula* (*norm* φ)

$\langle proof \rangle$

lemma *satisfies-nFOr*:
$\mathfrak{A} \models nFOr\ \varphi\ \psi \longleftrightarrow \mathfrak{A} \models FOr\ \varphi\ \psi$
$\langle proof \rangle$

lemma *satisfies-nFAnd*:
$\mathfrak{A} \models nFAnd\ \varphi\ \psi \longleftrightarrow \mathfrak{A} \models FAnd\ \varphi\ \psi$
$\langle proof \rangle$

lemma *satisfies-nFEx*: $lformula\ \varphi \Longrightarrow \mathfrak{A} \models nFEx\ b\ \varphi \longleftrightarrow \mathfrak{A} \models FEx\ b\ \varphi$
$\langle proof \rangle$

lemma *satisfies-nFAll*: $lformula\ \varphi \Longrightarrow \mathfrak{A} \models nFAll\ b\ \varphi \longleftrightarrow \mathfrak{A} \models FAll\ b\ \varphi$
$\langle proof \rangle$

lemma *satisfies-nFNot*:
$lformula\ \varphi \Longrightarrow \mathfrak{A} \models nFNot\ \varphi \longleftrightarrow \mathfrak{A} \models FNot\ \varphi$
$\langle proof \rangle$

lemma *satisfies-norm*: $lformula\ \varphi \Longrightarrow \mathfrak{A} \models norm\ \varphi \longleftrightarrow \mathfrak{A} \models \varphi$
$\langle proof \rangle$

lemma *satisfies-bounded-nFOr*:
$\mathfrak{A} \models_b nFOr\ \varphi\ \psi \longleftrightarrow \mathfrak{A} \models_b FOr\ \varphi\ \psi$
$\langle proof \rangle$

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

lemma *len-cut-0*: $len\ (cut\ 0\ P) = 0$
$\langle proof \rangle$

lemma *satisfies-bounded-nFEx*: $\mathfrak{A} \models_b nFEx\ b\ \varphi \longleftrightarrow \mathfrak{A} \models_b FEx\ b\ \varphi$
$\langle proof \rangle$

lemma *satisfies-bounded-nFAll*: $\mathfrak{A} \models_b nFAll\ b\ \varphi \longleftrightarrow \mathfrak{A} \models_b FAll\ b\ \varphi$
$\langle proof \rangle$

lemma *satisfies-bounded-nFNot*:
$\mathfrak{A} \models_b nFNot\ \varphi \longleftrightarrow \mathfrak{A} \models_b FNot\ \varphi$
$\langle proof \rangle$

lemma *satisfies-bounded-norm*: $\mathfrak{A} \models_b norm\ \varphi \longleftrightarrow \mathfrak{A} \models_b \varphi$
$\langle proof \rangle$

2.4 Derivatives of Formulas

lemma *wf-lderiv*:
$[\![wf\ idx\ \varphi;\ lformula\ \varphi]\!] \Longrightarrow wf\ idx\ (lderiv\ x\ \varphi)$
⟨*proof*⟩

lemma *lformula-lderiv*:
$lformula\ \varphi \Longrightarrow lformula\ (lderiv\ x\ \varphi)$
⟨*proof*⟩

lemma *wf-rderiv*:
$wf\ idx\ \varphi \Longrightarrow wf\ idx\ (rderiv\ x\ \varphi)$
⟨*proof*⟩

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

theorem *satisfies-bounded-lderiv*:
$[\![wf\ (\#_V\ \mathfrak{A})\ \varphi;\ \#_V\ \mathfrak{A} = size\ x;\ lformula\ \varphi]\!] \Longrightarrow \mathfrak{A} \models_b lderiv\ x\ \varphi \longleftrightarrow CONS\ x\ \mathfrak{A} \models_b \varphi$
⟨*proof*⟩

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*⟩

lemma *wf-norm-rderivs*: $wf\ idx\ \varphi \Longrightarrow wf\ idx\ (((norm \circ rderiv\ (zero\ idx))\ \widehat{}\widehat{}\ k)\ \varphi)$
⟨*proof*⟩

2.5 Finiteness of Derivatives Modulo ACI

lemma *finite-fold-deriv*:
assumes $(d0 = lderiv0 \wedge lformula\ \varphi) \vee d0 = rderiv0$
shows $finite\ \{\langle fold\ (deriv\ d0)\ xs\ \varphi\rangle \mid xs.\ True\}$
⟨*proof*⟩

lemma *lformula-nFOR*: $lformula\ (nFOR\ \varphi s) = (\forall \varphi \in set\ \varphi s.\ lformula\ \varphi)$
⟨*proof*⟩

lemma *lformula-nFAND*: $lformula\ (nFAND\ \varphi s) = (\forall \varphi \in set\ \varphi s.\ lformula\ \varphi)$
⟨*proof*⟩

lemma *lformula-NFOR*: $finite\ \Phi \Longrightarrow lformula\ (NFOR\ \Phi) = (\forall \varphi \in \Phi.\ lformula\ \varphi)$
⟨*proof*⟩

lemma *lformula-NFAND*: $finite\ \Phi \Longrightarrow lformula\ (NFAND\ \Phi) = (\forall \varphi \in \Phi.\ lformula$

φ)
⟨*proof*⟩

lemma *lformula-disjuncts*: $(\forall\,\psi \in \mathit{disjuncts}\ \varphi.\ \mathit{lformula}\ \psi) = \mathit{lformula}\ \varphi$
⟨*proof*⟩

lemma *lformula-conjuncts*: $(\forall\,\psi \in \mathit{conjuncts}\ \varphi.\ \mathit{lformula}\ \psi) = \mathit{lformula}\ \varphi$
⟨*proof*⟩

lemma *lformula-norm-ACI*: $\mathit{lformula}\ \langle\varphi\rangle = \mathit{lformula}\ \varphi$
⟨*proof*⟩

theorem
finite-fold-lderiv: $\mathit{lformula}\ \varphi \Longrightarrow \mathit{finite}\ \{\langle \mathit{fold}\ \mathit{lderiv}\ \mathit{xs}\ \langle\varphi\rangle\rangle \mid \mathit{xs}.\ \mathit{True}\}$ **and**
finite-fold-rderiv: $\mathit{finite}\ \{\langle \mathit{fold}\ \mathit{rderiv}\ \mathit{xs}\ \langle\varphi\rangle\rangle \mid \mathit{xs}.\ \mathit{True}\}$
⟨*proof*⟩

lemma *wf-nFOR*: $\mathit{wf}\ \mathit{idx}\ (\mathit{nFOR}\ \varphi s) \longleftrightarrow (\forall\,\varphi \in \mathit{set}\ \varphi s.\ \mathit{wf}\ \mathit{idx}\ \varphi)$
⟨*proof*⟩

lemma *wf-nFAND*: $\mathit{wf}\ \mathit{idx}\ (\mathit{nFAND}\ \varphi s) \longleftrightarrow (\forall\,\varphi \in \mathit{set}\ \varphi s.\ \mathit{wf}\ \mathit{idx}\ \varphi)$
⟨*proof*⟩

lemma *wf-NFOR*: $\mathit{finite}\ \Phi \Longrightarrow \mathit{wf}\ \mathit{idx}\ (\mathit{NFOR}\ \Phi) \longleftrightarrow (\forall\,\varphi \in \Phi.\ \mathit{wf}\ \mathit{idx}\ \varphi)$
⟨*proof*⟩

lemma *wf-NFAND*: $\mathit{finite}\ \Phi \Longrightarrow \mathit{wf}\ \mathit{idx}\ (\mathit{NFAND}\ \Phi) \longleftrightarrow (\forall\,\varphi \in \Phi.\ \mathit{wf}\ \mathit{idx}\ \varphi)$
⟨*proof*⟩

lemma *satisfies-bounded-nFOR*: $\mathfrak{A} \models_b \mathit{nFOR}\ \varphi s \longleftrightarrow (\exists\,\varphi \in \mathit{set}\ \varphi s.\ \mathfrak{A} \models_b \varphi)$
⟨*proof*⟩

lemma *satisfies-bounded-nFAND*: $\mathfrak{A} \models_b \mathit{nFAND}\ \varphi s \longleftrightarrow (\forall\,\varphi \in \mathit{set}\ \varphi s.\ \mathfrak{A} \models_b \varphi)$
⟨*proof*⟩

lemma *satisfies-bounded-NFOR*: $\mathit{finite}\ \Phi \Longrightarrow \mathfrak{A} \models_b \mathit{NFOR}\ \Phi \longleftrightarrow (\exists\,\varphi \in \Phi.\ \mathfrak{A} \models_b$
$\varphi)$
⟨*proof*⟩

lemma *satisfies-bounded-NFAND*: $\mathit{finite}\ \Phi \Longrightarrow \mathfrak{A} \models_b \mathit{NFAND}\ \Phi \longleftrightarrow (\forall\,\varphi \in \Phi.\ \mathfrak{A}$
$\models_b \varphi)$
⟨*proof*⟩

lemma *wf-juncts*:
$\mathit{wf}\ \mathit{idx}\ \varphi \longleftrightarrow (\forall\,\psi \in \mathit{disjuncts}\ \varphi.\ \mathit{wf}\ \mathit{idx}\ \psi)$
$\mathit{wf}\ \mathit{idx}\ \varphi \longleftrightarrow (\forall\,\psi \in \mathit{conjuncts}\ \varphi.\ \mathit{wf}\ \mathit{idx}\ \psi)$
⟨*proof*⟩

lemma *wf-norm-ACI*: $\mathit{wf}\ \mathit{idx}\ \langle\varphi\rangle = \mathit{wf}\ \mathit{idx}\ \varphi$

$\langle proof \rangle$

lemma *satisfies-bounded-disjuncts*:
$\mathfrak{A} \models_b \varphi \longleftrightarrow (\exists \psi \in disjuncts\ \varphi.\ \mathfrak{A} \models_b \psi)$
$\langle proof \rangle$

lemma *satisfies-bounded-conjuncts*:
$\mathfrak{A} \models_b \varphi \longleftrightarrow (\forall \psi \in conjuncts\ \varphi.\ \mathfrak{A} \models_b \psi)$
$\langle proof \rangle$

lemma *satisfies-bounded-norm-ACI*: $\mathfrak{A} \models_b \langle\varphi\rangle \longleftrightarrow \mathfrak{A} \models_b \varphi$
$\langle proof \rangle$

lemma *nvars-SNOCs*: $\#_V\ ((SNOC\ x^{\frown\frown}k)\ \mathfrak{A}) = \#_V\ \mathfrak{A}$
$\langle proof \rangle$

lemma *wf-fold-rderiv*: $wf\ idx\ \varphi \Longrightarrow wf\ idx\ (fold\ rderiv\ (replicate\ k\ x)\ \varphi)$
$\langle proof \rangle$

lemma *satisfies-bounded-fold-rderiv*:
$[\![wf\ idx\ \varphi;\ \#_V\ \mathfrak{A} = idx;\ size\ x = idx]\!] \Longrightarrow$
$\mathfrak{A} \models_b fold\ rderiv\ (replicate\ k\ x)\ \varphi \longleftrightarrow (SNOC\ x^{\frown\frown}k)\ \mathfrak{A} \models_b \varphi$
$\langle proof \rangle$

2.6 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* $k\ \varphi \equiv ((norm\ o\ rderiv\ (zero\ idx))^{\frown\frown}k)\ \varphi$

lemma *fut-derivs-Suc*[*simp*]: $norm\ (rderiv\ (zero\ idx)\ (fut\text{-}derivs\ k\ \varphi)) = fut\text{-}derivs\ (Suc\ k)\ \varphi$
$\langle proof \rangle$

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

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*$\}$
  $\langle$*proof*$\rangle$

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)
  $\langle$*proof*$\rangle$

lemma *satisfies-bounded-fut-derivs*:
  $\#_V\ \mathfrak{A} = idx \Longrightarrow \mathfrak{A} \models_b$ *fut-derivs k* $\psi \longleftrightarrow$ (*SNOC* (*zero idx*)$\widehat{\ }\widehat{\ }$*k*) $\mathfrak{A} \models_b \psi$
  $\langle$*proof*$\rangle$

lemma *fut-init*: *fut-invariant* (ψ, [])
  $\langle$*proof*$\rangle$

lemma *fut-spec-default*: *fut-spec fut-default*
  $\langle$*proof*$\rangle$

lemma *fut-invariant*: *fut-invariant* $\varphi\Phi \Longrightarrow$ *fut-test* $\varphi\Phi \Longrightarrow$ *fut-invariant* (*fut-step* $\varphi\Phi$)
  $\langle$*proof*$\rangle$

lemma *fut-terminate*: *fut-invariant* $\varphi\Phi \Longrightarrow \neg$ *fut-test* $\varphi\Phi \Longrightarrow$ *fut-spec* $\varphi\Phi$
$\langle$*proof*$\rangle$

lemma *fut-spec-while-default*:
  fut-spec (*while-default fut-default fut-test fut-step* (ψ, []))
  $\langle$*proof*$\rangle$

lemma *wf-fut*: *wf idx fut*
  $\langle$*proof*$\rangle$

lemma *satisfies-bounded-fut*:
  assumes $\#_V\ \mathfrak{A} = idx$
  shows $\mathfrak{A} \models_b$ *fut* $\longleftrightarrow$
    (*if b then* ($\exists$ *k*. (*SNOC* (*zero idx*) $\widehat{\ }\widehat{\ }$ *k*) $\mathfrak{A} \models_b \psi$) *else* ($\forall$ *k*. (*SNOC* (*zero idx*) $\widehat{\ }\widehat{\ }$ *k*) $\mathfrak{A} \models_b \psi$))

$\langle proof \rangle$

end

end

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

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

lemma *wf-finalize*: $wf\ idx\ \varphi \Longrightarrow wf\ idx\ (finalize\ idx\ \varphi)$
$\langle proof \rangle$

lemma *Length-SNOCs*: $Length\ ((SNOC\ x\ \hat{}\hat{}\ i)\ \mathfrak{A}) = Length\ \mathfrak{A} + i$
$\langle proof \rangle$

lemma *assigns-SNOCs-zero*:
$[\![LESS\ k\ m\ (\#_V\ \mathfrak{A});\ \#_V\ \mathfrak{A} = idx]\!] \Longrightarrow m^{(SNOC\ (zero\ idx)\ \hat{}\hat{}\ i)\ \mathfrak{A}}k = m^{\mathfrak{A}}k$
$\langle proof \rangle$

lemma *Extend-SNOCs-zero-satisfies*: $[\![wf\ (SUC\ k\ idx)\ \varphi;\ \#_V\ \mathfrak{A} = idx;\ lformula\ \varphi]\!] \Longrightarrow$
$Extend\ k\ 0\ ((SNOC\ (zero\ (\#_V\ \mathfrak{A}))\ \hat{}\hat{}\ i)\ \mathfrak{A})\ P \models \varphi \longleftrightarrow Extend\ k\ 0\ \mathfrak{A}\ P \models \varphi$
$\langle proof \rangle$

lemma *finalize-satisfies*: $[\![wf\ idx\ \varphi;\ \#_V\ \mathfrak{A} = idx;\ lformula\ \varphi]\!] \Longrightarrow \mathfrak{A} \models_b finalize\ idx\ \varphi \longleftrightarrow \mathfrak{A} \models \varphi$
$\langle proof \rangle$

lemma *Extend-empty-satisfies0*:
$[\![Length\ \mathfrak{A} = 0;\ len\ P = 0]\!] \Longrightarrow Extend\ k\ i\ \mathfrak{A}\ P \models_0 a \longleftrightarrow \mathfrak{A} \models_0 a$
$\langle proof \rangle$

lemma *Extend-empty-satisfies-bounded*:
$[\![Length\ \mathfrak{A} = 0;\ len\ P = 0]\!] \Longrightarrow Extend\ k\ 0\ \mathfrak{A}\ P \models_b \varphi \longleftrightarrow \mathfrak{A} \models_b \varphi$
$\langle proof \rangle$

lemma *nullable-satisfies-bounded*: $Length\ \mathfrak{A} = 0 \Longrightarrow nullable\ \varphi \longleftrightarrow \mathfrak{A} \models_b \varphi$
$\langle proof \rangle$

lemma *final-satisfies*:
$[\![wf\ idx\ \varphi \wedge lformula\ \varphi;\ Length\ \mathfrak{A} = 0;\ \#_V\ \mathfrak{A} = idx]\!] \Longrightarrow final\ idx\ \varphi = (\mathfrak{A} \models \varphi)$

$\langle proof \rangle$

2.7 Restrictions

lemma *satisfies-gen-restrict-RESTR*:
$satisfies\text{-}gen\ (\lambda k\ P\ n.\ restrict\ k\ P \wedge r\ k\ P\ n)\ \mathfrak{A}\ \varphi \longleftrightarrow satisfies\text{-}gen\ r\ \mathfrak{A}\ (RESTR\ \varphi)$
$\langle proof \rangle$

lemma *finite-FV*: $finite\ (FV\ \varphi\ k)$
$\langle proof \rangle$

lemma *satisfies-gen-restrict*:
$satisfies\text{-}gen\ r\ \mathfrak{A}\ \varphi \wedge (\forall x \in set\ V.\ restrict\ k\ (x^{\mathfrak{A}}k)) \longleftrightarrow$
$satisfies\text{-}gen\ r\ \mathfrak{A}\ (foldr\ (\lambda x.\ FAnd\ (Restrict\ k\ x))\ V\ \varphi)$
$\langle proof \rangle$

lemma *sat-vars-RESTRICT-VARS*:
fixes φ
defines $vs \equiv sorted\text{-}list\text{-}of\text{-}set\ o\ FV\ \varphi$
assumes $\forall k \in set\ ks.\ finite\ (FV\ \varphi\ k)$
shows $sat\text{-}vars\text{-}gen\ r\ (set\ ks)\ \mathfrak{A}\ \varphi \longleftrightarrow satisfies\text{-}gen\ r\ \mathfrak{A}\ (RESTRICT\text{-}VARS\ ks\ vs\ \varphi)$
$\langle proof \rangle$

lemma *sat-RESTRICT*: $sat\ \mathfrak{A}\ \varphi \longleftrightarrow \mathfrak{A} \models RESTRICT\ \varphi$
$\langle proof \rangle$

lemma *sat$_b$-RESTRICT*: $sat_b\ \mathfrak{A}\ \varphi \longleftrightarrow \mathfrak{A} \models_b RESTRICT\ \varphi$
$\langle proof \rangle$

lemma *wf-RESTR*: $wf\ idx\ \varphi \Longrightarrow wf\ idx\ (RESTR\ \varphi)$
$\langle proof \rangle$

lemma *wf-RESTRICT-VARS*: $[\![wf\ idx\ \varphi;\ \forall k \in set\ ks.\ \forall v \in set\ (vs\ k).\ LESS\ k\ v\ idx]\!] \Longrightarrow$
$wf\ idx\ (RESTRICT\text{-}VARS\ ks\ vs\ \varphi)$
$\langle proof \rangle$

lemma *wf-FV-LESS*: $[\![wf\ idx\ \varphi;\ v \in FV\ \varphi\ k]\!] \Longrightarrow LESS\ k\ v\ idx$
$\langle proof \rangle$

lemma *wf-RESTRICT*: $wf\ idx\ \varphi \Longrightarrow wf\ idx\ (RESTRICT\ \varphi)$
$\langle proof \rangle$

lemma *lformula-RESTR*: $lformula\ \varphi \Longrightarrow lformula\ (RESTR\ \varphi)$
$\langle proof \rangle$

lemma *lformula-RESTRICT-VARS*: $lformula\ \varphi \Longrightarrow lformula\ (RESTRICT\text{-}VARS$

ks vs φ)
⟨*proof*⟩

lemma *lformula-RESTRICT*: *lformula* $\varphi \Longrightarrow$ *lformula* (*RESTRICT* φ)
⟨*proof*⟩

lemma *ex-fold-CONS*: $\exists$ *xs* $\mathfrak{B}$. $\mathfrak{A}$ = *fold CONS xs* $\mathfrak{B} \wedge$ *Length* $\mathfrak{B}$ = *0* $\wedge$ *Length* $\mathfrak{A}$ = *length xs* $\wedge$
$\#_V$ $\mathfrak{B}$ = $\#_V$ $\mathfrak{A} \wedge$ ($\forall$ *x* $\in$ *set xs*. *size x* = $\#_V$ $\mathfrak{A}$)
⟨*proof*⟩

primcorec *L* **where**
L idx I = *Lang* ($\exists\mathfrak{A}$. *Length* $\mathfrak{A}$ = *0* $\wedge$ $\#_V$ $\mathfrak{A}$ = *idx* $\wedge$ $\mathfrak{A} \in I$)
(λa. *if size a* = *idx then L idx* {$\mathfrak{B}$. *CONS a* $\mathfrak{B} \in I$} *else Zero*)

lemma *L-empty*: *L idx* {} = *Zero*
⟨*proof*⟩

lemma *L-alt*: *L idx I* =
to-language {*xs*. $\exists\mathfrak{A} \in I$. $\exists\mathfrak{B}$. $\mathfrak{A}$ = *fold CONS* (*rev xs*) $\mathfrak{B} \wedge$ *Length* $\mathfrak{B}$ = *0* $\wedge$
$\#_V$ $\mathfrak{B}$ = *idx* $\wedge$ ($\forall$ *x* $\in$ *set xs*. *size x* = *idx*)}
⟨*proof*⟩

definition *lang idx* φ = *L idx* {$\mathfrak{A}$. $\mathfrak{A} \models \varphi \wedge \#_V$ $\mathfrak{A}$ = *idx*}
definition $lang_b$ *idx* φ = *L idx* {$\mathfrak{A}$. $\mathfrak{A} \models_b \varphi \wedge \#_V$ $\mathfrak{A}$ = *idx*}
definition *language idx* φ = *L idx* {$\mathfrak{A}$. *sat* $\mathfrak{A}$ $\varphi \wedge \#_V$ $\mathfrak{A}$ = *idx*}
definition $language_b$ *idx* φ = *L idx* {$\mathfrak{A}$. sat_b $\mathfrak{A}$ $\varphi \wedge \#_V$ $\mathfrak{A}$ = *idx*}

lemma *lformula* $\varphi \Longrightarrow$ *lang n* (*norm* φ) = *lang n* φ
⟨*proof*⟩

lemma *in-language-Zero*[*simp*]: $\neg$ *in-language Zero w*
⟨*proof*⟩

lemma *in-language-L-size*: *in-language* (*L idx I*) *w* $\Longrightarrow$ *x* $\in$ *set w* $\Longrightarrow$ *size x* = *idx*
⟨*proof*⟩

end

sublocale *Formula* <
bounded: *DA alphabet idx* $\lambda\varphi$. *norm* (*RESTRICT* φ) λa φ. *norm* (*lderiv a* φ) *nullable*
$\lambda\varphi$. *wf idx* $\varphi \wedge$ *lformula* φ $lang_b$ *idx*
$\lambda\varphi$. *wf idx* $\varphi \wedge$ *lformula* φ $language_b$ *idx* **for** *idx*
⟨*proof*⟩

sublocale *Formula* <
unbounded?: *DA alphabet idx* $\lambda\varphi$. *norm* (*RESTRICT* φ) λa φ. *norm* (*lderiv a* φ)

final idx
  $\lambda\varphi.$ *wf idx* $\varphi \wedge$ *lformula* φ *lang idx*
  $\lambda\varphi.$ *wf idx* $\varphi \wedge$ *lformula* φ *language idx* **for** *idx*
⟨*proof*⟩

lemma (**in** *Formula*) *check-eqv-soundness*:
$[\![\#_V\ \mathfrak{A} = idx;\ check\text{-}eqv\ idx\ \varphi\ \psi]\!] \Longrightarrow sat\ \mathfrak{A}\ \varphi \longleftrightarrow sat\ \mathfrak{A}\ \psi$
⟨*proof*⟩

lemma (**in** *Formula*) *bounded-check-eqv-soundness*:
$[\![\#_V\ \mathfrak{A} = idx;\ bounded.check\text{-}eqv\ idx\ \varphi\ \psi]\!] \Longrightarrow sat_b\ \mathfrak{A}\ \varphi \longleftrightarrow sat_b\ \mathfrak{A}\ \psi$
⟨*proof*⟩

end

3 WS1S Interpretations

definition *eval P x* = (*x* |∈| *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* ($\lambda i.\ i < n$) *P*
definition *len P* = (*if P* = {||} *then 0 else Suc* (*fMax P*))

datatype *order* = *FO* | *SO*
derive *linorder order*
instantiation *order* :: *enum* **begin**
  definition *enum-order* = [*FO*, *SO*]
  definition *enum-all-order P* = (*P FO* ∧ *P SO*)
  definition *enum-ex-order P* = (*P FO* ∨ *P SO*)
  lemmas *enum-defs* = *enum-order-def enum-all-order-def enum-ex-order-def*
  instance ⟨*proof*⟩
end

typedef *idx* = *UNIV* :: (*nat* × *nat*) *set* ⟨*proof*⟩

setup-lifting *type-definition-idx*

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

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

lemma *MSB-Nil*[*simp*]: *MSB* [] = *0*
⟨*proof*⟩

lemma *MSB-Cons*[*simp*]: *MSB* (*I* # *Is*) = *max* (*if I* = {||} *then 0 else Suc* (*fMax I*)) (*MSB Is*)
⟨*proof*⟩ **including** *fset.lifting*
⟨*proof*⟩

lemma *MSB-append*[*simp*]: *MSB* (*I1* @ *I2*) = *max* (*MSB I1*) (*MSB I2*)
⟨*proof*⟩

lemma *MSB-insert-nth*[*simp*]:
MSB (*insert-nth n P Is*) = *max* (*if P* = {||} *then 0 else Suc* (*fMax P*)) (*MSB Is*)
⟨*proof*⟩

lemma *MSB-greater*:
[[*i* < *length Is*; *p* |∈| *Is* ! *i*]] ⟹ *p* < *MSB Is*
⟨*proof*⟩

lemma *MSB-mono*: *set I1* ⊆ *set I2* ⟹ *MSB I1* ≤ *MSB I2*
⟨*proof*⟩ **including** *fset.lifting*
⟨*proof*⟩

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*))
⟨*proof*⟩

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*)
⟨*proof*⟩

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

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

setup-lifting *type-definition-interp*

lift-definition *assigns* :: *nat* ⇒ *interp* ⇒ *order* ⇒ *nat fset* (‹-¯-› [*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* {||} ⟨*proof*⟩
lift-definition *nvars* :: *interp* ⇒ *idx* (‹$\#_V$ -› [*1000*] *900*)

is λ(-, *I1*, *I2*). (*length I1*, *length I2*) ⟨*proof*⟩
lift-definition *Length* :: *interp* ⇒ *nat*
is λ(*n*, -, -). *n* ⟨*proof*⟩
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*)
⟨*proof*⟩

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*)
⟨*proof*⟩

lift-definition *SNOC* :: (*bool list* × *bool list*) ⇒ *interp* ⇒ *interp*
is λ(*bs1*, *bs2*) (*n*, *I1*, *I2*).
(*Suc n*, *map-index* (*snoc n bs1*) *I1*, *map-index* (*snoc n bs2*) *I2*)
⟨*proof*⟩

type-synonym *atom* = *bool list* × *bool list*

lift-definition *zero* :: *idx* ⇒ *atom*
is λ(*m*, *n*). (*replicate m False*, *replicate n False*) ⟨*proof*⟩

definition *extend ord b* ≡
λ(*bs1*, *bs2*). *case ord of FO* ⇒ (*b* # *bs1*, *bs2*) | *SO* ⇒ (*bs1*, *b* # *bs2*)
lift-definition *size-atom* :: *bool list* × *bool list* ⇒ *idx*
is λ(*bs1*, *bs2*). (*length bs1*, *length bs2*) ⟨*proof*⟩

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*]))
⟨*proof*⟩

lemma *fMin-fimage-Suc*[*simp*]: *x* |∈| *A* ⟹ *fMin* (*Suc* |‘| *A*) = *Suc* (*fMin A*)
⟨*proof*⟩

lemma *fMin-eq-0*[*simp*]: *0* |∈| *A* ⟹ *fMin A* = (*0* :: *nat*)
⟨*proof*⟩

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*)
⟨*proof*⟩

lemma *insert-nth-commute*[*simp*]:
assumes *j* ≤ *i i* ≤ *length xs*
shows *insert-nth j y* (*insert-nth i x xs*) = *insert-nth* (*Suc i*) *x* (*insert-nth j y xs*)
⟨*proof*⟩

lemma *SUC-SUC*[*simp*]: $SUC\ ord\ (SUC\ ord'\ idx) = SUC\ ord'\ (SUC\ ord\ idx)$
⟨*proof*⟩

lemma *LESS-SUC*[*simp*]:
$LESS\ ord\ 0\ (SUC\ ord\ idx)$
$LESS\ ord\ (Suc\ l)\ (SUC\ ord\ idx) = LESS\ ord\ l\ idx$
$ord \neq ord' \Longrightarrow LESS\ ord\ l\ (SUC\ ord'\ idx) = LESS\ ord\ l\ idx$
$LESS\ ord\ l\ idx \Longrightarrow LESS\ ord\ l\ (SUC\ ord'\ idx)$
⟨*proof*⟩

lemma *nvars-Extend*[*simp*]:
$\#_V\ (Extend\ ord\ i\ \mathfrak{A}\ P) = SUC\ ord\ (\#_V\ \mathfrak{A})$
⟨*proof*⟩

lemma *Length-Extend*[*simp*]:
$Length\ (Extend\ k\ i\ \mathfrak{A}\ P) = max\ (Length\ \mathfrak{A})\ (if\ P = \{||\}\ then\ 0\ else\ Suc\ (fMax\ P))$
⟨*proof*⟩

lemma *assigns-Extend*[*simp*]:
$LEQ\ ord\ i\ (\#_V\ \mathfrak{A}) \Longrightarrow m^{Extend\ ord\ i\ \mathfrak{A}\ P}{}_{ord} = (if\ m = i\ then\ P\ else\ (if\ m > i\ then\ m - Suc\ 0\ else\ m)^{\mathfrak{A}}{}_{ord})$
$ord \neq ord' \Longrightarrow m^{Extend\ ord\ i\ \mathfrak{A}\ P}{}_{ord'} = m^{\mathfrak{A}}{}_{ord'}$
⟨*proof*⟩

lemma *Extend-commute-safe*[*simp*]:
$[\![j \leq i;\ LEQ\ ord\ i\ (\#_V\ \mathfrak{A})]\!] \Longrightarrow$
$Extend\ ord\ j\ (Extend\ ord\ i\ \mathfrak{A}\ P1)\ P2 = Extend\ ord\ (Suc\ i)\ (Extend\ ord\ j\ \mathfrak{A}\ P2)\ P1$
⟨*proof*⟩

lemma *Extend-commute-unsafe*:
$ord \neq ord' \Longrightarrow Extend\ ord\ j\ (Extend\ ord'\ i\ \mathfrak{A}\ P1)\ P2 = Extend\ ord'\ i\ (Extend\ ord\ j\ \mathfrak{A}\ P2)\ P1$
⟨*proof*⟩

lemma *Length-CONS*[*simp*]:
$Length\ (CONS\ x\ \mathfrak{A}) = Suc\ (Length\ \mathfrak{A})$
⟨*proof*⟩

lemma *Length-SNOC*[*simp*]:
$Length\ (SNOC\ x\ \mathfrak{A}) = Suc\ (Length\ \mathfrak{A})$
⟨*proof*⟩

lemma *nvars-CONS*[*simp*]:
$\#_V\ (CONS\ x\ \mathfrak{A}) = \#_V\ \mathfrak{A}$
⟨*proof*⟩

lemma *nvars-SNOC*[*simp*]:

$\#_V$ $(SNOC\ x\ \mathfrak{A}) = \#_V\ \mathfrak{A}$
⟨*proof*⟩

lemma *assigns-CONS*[*simp*]:
assumes $\#_V\ \mathfrak{A} = \textit{size-atom bs1-bs2}$
shows $\textit{LESS ord x}\ (\#_V\ \mathfrak{A}) \Longrightarrow x^{\textit{CONS bs1-bs2}\ \mathfrak{A}}{}_{ord} =$
$(\textit{if case-prod case-order bs1-bs2 ord ! x then finsert 0 (upshift}\ (x^{\mathfrak{A}}\textit{ord})) \textit{ else upshift}\ (x^{\mathfrak{A}}\textit{ord}))$
⟨*proof*⟩

lemma *assigns-SNOC*[*simp*]:
assumes $\#_V\ \mathfrak{A} = \textit{size-atom bs1-bs2}$
shows $\textit{LESS ord x}\ (\#_V\ \mathfrak{A}) \Longrightarrow x^{\textit{SNOC bs1-bs2}\ \mathfrak{A}}{}_{ord} =$
$(\textit{if case-prod case-order bs1-bs2 ord ! x then finsert}\ (\textit{Length}\ \mathfrak{A})\ (x^{\mathfrak{A}}\textit{ord}) \textit{ else } x^{\mathfrak{A}}\textit{ord})$
⟨*proof*⟩

lemma *map-index′-eq-conv*[*simp*]:
$\textit{map-index}'\ i\ f\ xs = \textit{map-index}'\ j\ g\ xs = (\forall k < \textit{length xs}.\ f\ (i + k)\ (xs\ !\ k) = g\ (j + k)\ (xs\ !\ k))$
⟨*proof*⟩

lemma *fMax-Diff-0*[*simp*]: $\textit{Suc m}\ |\in|\ P \Longrightarrow \textit{fMax}\ (P\ |-|\ \{|0|\}) = \textit{fMax}\ P$
⟨*proof*⟩

lemma *Suc-fMax-pred-fimage*[*simp*]:
assumes $\textit{Suc p}\ |\in|\ P\ \ 0\ |\notin|\ P$
shows $\textit{Suc}\ (\textit{fMax}\ ((\lambda x.\ x - \textit{Suc}\ 0)\ |`|\ P)) = \textit{fMax}\ P$
⟨*proof*⟩

lemma *Extend-CONS*[*simp*]: $\#_V\ \mathfrak{A} = \textit{size-atom}\ x \Longrightarrow \textit{Extend ord}\ 0\ (\textit{CONS}\ x\ \mathfrak{A})\ P =$
$\textit{CONS}\ (\textit{extend ord}\ (\textit{eval}\ P\ 0)\ x)\ (\textit{Extend ord}\ 0\ \mathfrak{A}\ (\textit{downshift}\ P))$
⟨*proof*⟩

lemma *size-atom-extend*[*simp*]:
$\textit{size-atom}\ (\textit{extend ord b x}) = \textit{SUC ord}\ (\textit{size-atom}\ x)$
⟨*proof*⟩

lemma *size-atom-zero*[*simp*]:
$\textit{size-atom}\ (\textit{zero idx}) = \textit{idx}$
⟨*proof*⟩

lemma *interp-eqI*:
$[\![\textit{Length}\ \mathfrak{A} = \textit{Length}\ \mathfrak{B};\ \#_V\ \mathfrak{A} = \#_V\ \mathfrak{B};\ \bigwedge m\ k.\ \textit{LESS k m}\ (\#_V\ \mathfrak{A}) \Longrightarrow m^{\mathfrak{A}}k = m^{\mathfrak{B}}k]\!] \Longrightarrow \mathfrak{A} = \mathfrak{B}$
⟨*proof*⟩

lemma *Extend-SNOC-cut*[*unfolded eval-def cut-def Length-SNOC*, *simp*]:

$\llbracket$*len P* $\leq$ *Length* (*SNOC x* $\mathfrak{A}$); $\#_V$ $\mathfrak{A}$ = *size-atom x*$\rrbracket$ $\Longrightarrow$
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*))
⟨*proof*⟩

lemma *nth-replicate-simp*: *replicate m x* ! *i* = (*if i* < *m then x else* [] ! (*i* − *m*))
⟨*proof*⟩

lemma *MSB-eq-SucD*: *MSB Is* = *Suc x* $\Longrightarrow$ $\exists$ *P*$\in$*set Is*. *x* |$\in$| *P*
⟨*proof*⟩

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*⟩

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*)
⟨*proof*⟩

lemma *ex-Length-0*[*simp*]:
$\exists$ $\mathfrak{A}$. *Length* $\mathfrak{A}$ = *0* $\wedge$ $\#_V$ $\mathfrak{A}$ = *idx*
⟨*proof*⟩

lemma *is-empty-inj*[*simp*]: $\llbracket$*Length* $\mathfrak{A}$ = *0*; *Length* $\mathfrak{B}$ = *0*; $\#_V$ $\mathfrak{A}$ = $\#_V$ $\mathfrak{B}$$\rrbracket$ $\Longrightarrow$ $\mathfrak{A}$ = $\mathfrak{B}$
⟨*proof*⟩

lemma *set-σ-length-atom*[*simp*]: (*x* $\in$ *set* (*σ idx*)) $\longleftrightarrow$ *idx* = *size-atom x*
⟨*proof*⟩

lemma *distinct-σ*[*simp*]: *distinct* (*σ idx*)
⟨*proof*⟩

lemma *fMin-less-Length*[*simp*]: *x* |$\in$| $m1^{\mathfrak{A}}k$ $\Longrightarrow$ *fMin* ($m1^{\mathfrak{A}}k$) < *Length* $\mathfrak{A}$
⟨*proof*⟩

lemma *fMax-less-Length*[*simp*]: *x* |$\in$| $m1^{\mathfrak{A}}k$ $\Longrightarrow$ *fMax* ($m1^{\mathfrak{A}}k$) < *Length* $\mathfrak{A}$
⟨*proof*⟩

lemma *min-Length-fMin*[*simp*]: *x* |$\in$| $m1^{\mathfrak{A}}k$ $\Longrightarrow$ *min* (*Length* $\mathfrak{A}$) (*fMin* ($m1^{\mathfrak{A}}k$)) = *fMin* ($m1^{\mathfrak{A}}k$)
⟨*proof*⟩

lemma *max-Length-fMin*[*simp*]: *x* |$\in$| $m1^{\mathfrak{A}}k$ $\Longrightarrow$ *max* (*Length* $\mathfrak{A}$) (*fMin* ($m1^{\mathfrak{A}}k$)) = *Length* $\mathfrak{A}$
⟨*proof*⟩

lemma *min-Length-fMax*[*simp*]: x |∈| $m1^{\mathfrak{A}}k \Longrightarrow$ *min* (*Length* $\mathfrak{A}$) (*fMax* ($m1^{\mathfrak{A}}k$)) = *fMax* ($m1^{\mathfrak{A}}k$)
⟨*proof*⟩

lemma *max-Length-fMax*[*simp*]: x |∈| $m1^{\mathfrak{A}}k \Longrightarrow$ *max* (*Length* $\mathfrak{A}$) (*fMax* ($m1^{\mathfrak{A}}k$)) = *Length* $\mathfrak{A}$
⟨*proof*⟩

lemma *assigns-less-Length*[*simp*]: x |∈| $m1^{\mathfrak{A}}k \Longrightarrow x <$ *Length* $\mathfrak{A}$
⟨*proof*⟩

lemma *Length-notin-assigns*[*simp*]: *Length* $\mathfrak{A}$ |∉| $m^{\mathfrak{A}}k$
⟨*proof*⟩

lemma *nth-zero*[*simp*]: *LESS ord m* ($\#_V$ $\mathfrak{A}$) $\Longrightarrow \neg$ *case-prod case-order* (*zero* ($\#_V$ $\mathfrak{A}$)) *ord* ! *m*
⟨*proof*⟩

lemma *in-fimage-Suc*[*simp*]: x |∈| *Suc* |'| $A \longleftrightarrow (\exists\, y.\ y$ |∈| $A \wedge x =$ *Suc* y)
⟨*proof*⟩

lemma *fimage-Suc-inj*[*simp*]: *Suc* |'| $A =$ *Suc* |'| $B \longleftrightarrow A = B$
⟨*proof*⟩

lemma *MSB-eq0-D*: *MSB* $I = 0 \Longrightarrow x <$ *length* $I \Longrightarrow I$! x = {||}
⟨*proof*⟩

lemma *Suc-in-fimage-Suc*: *Suc* x |∈| *Suc* |'| $X \longleftrightarrow x$ |∈| X
⟨*proof*⟩

lemma *Suc-in-fimage-Suc-o-Suc*[*simp*]: *Suc* x |∈| (*Suc* ∘ *Suc*) |'| $X \longleftrightarrow x$ |∈| *Suc* |'| X
⟨*proof*⟩

lemma *finsert-same-eq-iff*[*simp*]: *finsert* k X = *finsert* k $Y \longleftrightarrow X$ |−| {|k|} = Y |−| {|k|}
⟨*proof*⟩

lemma *fimage-Suc-o-Suc-eq-fimage-Suc-iff*[*simp*]:
((*Suc* ∘ *Suc*) |'| X = *Suc* |'| Y) $\longleftrightarrow$ (*Suc* |'| $X = Y$)
⟨*proof*⟩

lemma *fMax-image-Suc*[*simp*]: x |∈| $P \Longrightarrow$ *fMax* (*Suc* |'| P) = *Suc* (*fMax* P)
⟨*proof*⟩

lemma *fimage-Suc-eq-singleton*[*simp*]: (*fimage Suc* Z = {|y|}) $\longleftrightarrow (\exists\, x.\ Z$ = {|x|} $\wedge$ *Suc* $x = y$)
⟨*proof*⟩

lemma *len-downshift-helper*:
$x \mid\in\mid P \Longrightarrow Suc\ (fMax\ ((\lambda x.\ x - Suc\ 0) \mid`\mid (P \mid-\mid \{\mid 0\mid\}))) \neq fMax\ P \Longrightarrow xa \mid\in\mid P \Longrightarrow xa = 0$
$\langle proof \rangle$

lemma *CONS-inj*[*simp*]: $size\text{-}atom\ x = \#_V\ \mathfrak{A} \Longrightarrow size\text{-}atom\ y = \#_V\ \mathfrak{A} \Longrightarrow \#_V\ \mathfrak{A} = \#_V\ \mathfrak{B} \Longrightarrow$
$CONS\ x\ \mathfrak{A} = CONS\ y\ \mathfrak{B} \longleftrightarrow (x = y \wedge \mathfrak{A} = \mathfrak{B})$
$\langle proof \rangle$

lemma *Suc-minus1*: $Suc\ (x - Suc\ 0) = (if\ x = 0\ then\ Suc\ 0\ else\ x)$
$\langle proof \rangle$

lemma *fset-eq-empty-iff*: $(fset\ X = \{\}) = (X = \{\mid\mid\})$
$\langle proof \rangle$

lemma *fset-le-singleton-iff*: $(fset\ X \subseteq \{x\}) = (X = \{\mid\mid\} \vee X = \{\mid x\mid\})$
$\langle proof \rangle$

lemma *MSB-decreases*:
$MSB\ I \leq Suc\ m \Longrightarrow MSB\ (map\ (\lambda X.\ (\lambda I1.\ I1 - Suc\ 0) \mid`\mid (X \mid-\mid \{\mid 0\mid\}))\ I) \leq m$
$\langle proof \rangle$

lemma *CONS-surj*[*dest*]:
assumes $Length\ \mathfrak{A} > 0$
shows $\exists\, x\ \mathfrak{B}.\ \mathfrak{A} = CONS\ x\ \mathfrak{B} \wedge \#_V\ \mathfrak{B} = \#_V\ \mathfrak{A} \wedge size\text{-}atom\ x = \#_V\ \mathfrak{A}$
$\langle proof \rangle$

4 Concrete Atomic WS1S Formulas (Minimum Semantics for FO Variables)

datatype (*FOV0*: *'fo*, *SOV0*: *'so*) *atomic* =
Fo 'fo |
Eq-Const nat option 'fo nat |
Less bool option 'fo 'fo |
Plus-FO nat option 'fo 'fo nat |
Eq-FO bool 'fo 'fo |
Eq-SO 'so 'so |
Suc-SO bool bool 'so 'so |
Empty 'so |
Singleton 'so |
Subset 'so 'so |
In bool 'fo 'so |
Eq-Max bool 'fo 'so |
Eq-Min bool 'fo 'so |

Eq-Union 'so 'so 'so |
Eq-Inter 'so 'so 'so |
Eq-Diff 'so 'so 'so |
Disjoint 'so 'so |
Eq-Presb nat option 'so nat

derive *linorder option*
derive *linorder atomic* — very slow

type-synonym *fo* = *nat*
type-synonym *so* = *nat*
type-synonym *ws1s* = (*fo*, *so*) *atomic*
type-synonym *formula* = (*ws1s*, *order*) *aformula*

primrec *wf0* **where**
wf0 idx (*Fo m*) = *LESS FO m idx*
| *wf0 idx* (*Eq-Const i m n*) = (*LESS FO m idx* ∧ (*case i of Some i* ⇒ *i* ≤ *n* | - ⇒ *True*))
| *wf0 idx* (*Less - m1 m2*) = (*LESS FO m1 idx* ∧ *LESS FO m2 idx*)
| *wf0 idx* (*Plus-FO i m1 m2 n*) =
(*LESS FO m1 idx* ∧ *LESS FO m2 idx* ∧ (*case i of Some i* ⇒ *i* ≤ *n* | - ⇒ *True*))
| *wf0 idx* (*Eq-FO - m1 m2*) = (*LESS FO m1 idx* ∧ *LESS FO m2 idx*)
| *wf0 idx* (*Eq-SO M1 M2*) = (*LESS SO M1 idx* ∧ *LESS SO M2 idx*)
| *wf0 idx* (*Suc-SO br bl M1 M2*) = (*LESS SO M1 idx* ∧ *LESS SO M2 idx*)
| *wf0 idx* (*Empty M*) = *LESS SO M idx*
| *wf0 idx* (*Singleton M*) = *LESS SO M idx*
| *wf0 idx* (*Subset M1 M2*) = (*LESS SO M1 idx* ∧ *LESS SO M2 idx*)
| *wf0 idx* (*In - m M*) = (*LESS FO m idx* ∧ *LESS SO M idx*)
| *wf0 idx* (*Eq-Max - m M*) = (*LESS FO m idx* ∧ *LESS SO M idx*)
| *wf0 idx* (*Eq-Min - m M*) = (*LESS FO m idx* ∧ *LESS SO M idx*)
| *wf0 idx* (*Eq-Union M1 M2 M3*) = (*LESS SO M1 idx* ∧ *LESS SO M2 idx* ∧ *LESS SO M3 idx*)
| *wf0 idx* (*Eq-Inter M1 M2 M3*) = (*LESS SO M1 idx* ∧ *LESS SO M2 idx* ∧ *LESS SO M3 idx*)
| *wf0 idx* (*Eq-Diff M1 M2 M3*) = (*LESS SO M1 idx* ∧ *LESS SO M2 idx* ∧ *LESS SO M3 idx*)
| *wf0 idx* (*Disjoint M1 M2*) = (*LESS SO M1 idx* ∧ *LESS SO M2 idx*)
| *wf0 idx* (*Eq-Presb - M n*) = *LESS SO M idx*

inductive *lformula0* **where**
lformula0 (*Fo m*)
| *lformula0* (*Eq-Const None m n*)
| *lformula0* (*Less None m1 m2*)
| *lformula0* (*Plus-FO None m1 m2 n*)
| *lformula0* (*Eq-FO False m1 m2*)
| *lformula0* (*Eq-SO M1 M2*)
| *lformula0* (*Suc-SO False bl M1 M2*)
| *lformula0* (*Empty M*)
| *lformula0* (*Singleton M*)

| *lformula0* (*Subset M1 M2*)
| *lformula0* (*In False m M*)
| *lformula0* (*Eq-Max False m M*)
| *lformula0* (*Eq-Min False m M*)
| *lformula0* (*Eq-Union M1 M2 M3*)
| *lformula0* (*Eq-Inter M1 M2 M3*)
| *lformula0* (*Eq-Diff M1 M2 M3*)
| *lformula0* (*Disjoint M1 M2*)
| *lformula0* (*Eq-Presb None M n*)

code-pred *lformula0* ⟨*proof*⟩

declare *lformula0.intros*[*simp*]

inductive-cases *lformula0E*[*elim*]: *lformula0 a*

abbreviation *FV0* ≡ *case-order FOV0 SOV0*

fun *find0* **where**
find0 FO i (*Fo m*) = (*i* = *m*)
| *find0 FO i* (*Eq-Const - m -*) = (*i* = *m*)
| *find0 FO i* (*Less - m1 m2*) = (*i* = *m1* ∨ *i* = *m2*)
| *find0 FO i* (*Plus-FO - m1 m2 -*) = (*i* = *m1* ∨ *i* = *m2*)
| *find0 FO i* (*Eq-FO - m1 m2*) = (*i* = *m1* ∨ *i* = *m2*)
| *find0 SO i* (*Eq-SO M1 M2*) = (*i* = *M1* ∨ *i* = *M2*)
| *find0 SO i* (*Suc-SO - - M1 M2*) = (*i* = *M1* ∨ *i* = *M2*)
| *find0 SO i* (*Empty M*) = (*i* = *M*)
| *find0 SO i* (*Singleton M*) = (*i* = *M*)
| *find0 SO i* (*Subset M1 M2*) = (*i* = *M1* ∨ *i* = *M2*)
| *find0 FO i* (*In - m -*) = (*i* = *m*)
| *find0 SO i* (*In - - M*) = (*i* = *M*)
| *find0 FO i* (*Eq-Max - m -*) = (*i* = *m*)
| *find0 SO i* (*Eq-Max - - M*) = (*i* = *M*)
| *find0 FO i* (*Eq-Min - m -*) = (*i* = *m*)
| *find0 SO i* (*Eq-Min - - M*) = (*i* = *M*)
| *find0 SO i* (*Eq-Union M1 M2 M3*) = (*i* = *M1* ∨ *i* = *M2* ∨ *i* = *M3*)
| *find0 SO i* (*Eq-Inter M1 M2 M3*) = (*i* = *M1* ∨ *i* = *M2* ∨ *i* = *M3*)
| *find0 SO i* (*Eq-Diff M1 M2 M3*) = (*i* = *M1* ∨ *i* = *M2* ∨ *i* = *M3*)
| *find0 SO i* (*Disjoint M1 M2*) = (*i* = *M1* ∨ *i* = *M2*)
| *find0 SO i* (*Eq-Presb - M -*) = (*i* = *M*)
| *find0 - - -* = *False*

abbreviation *decr0 ord k* ≡ *map-atomic* (*case-order* (*dec k*) *id ord*) (*case-order id* (*dec k*) *ord*)

lemma *sum-pow2-image-Suc*:
finite X ⟹ *sum* ((^) (*2* :: *nat*)) (*Suc* ` *X*) = *2* ∗ *sum* ((^) *2*) *X*
⟨*proof*⟩

lemma *sum-pow2-insert0*:
$[\![$*finite X*; $0 \notin X]\!] \Longrightarrow$ *sum* $((\widehat{\ })\ (2 :: nat))\ (insert\ 0\ X) = Suc\ (sum\ ((\widehat{\ })\ 2)\ X)$
⟨*proof*⟩

lemma *sum-pow2-upto*: $sum\ ((\widehat{\ })\ (2 :: nat))\ \{0\ ..< x\} = 2\ \widehat{\ }\ x - 1$
⟨*proof*⟩

lemma *sum-pow2-inj*:
$[\![$*finite X*; *finite Y*; $(\sum x{\in}X.\ 2\ \widehat{\ }\ x :: nat) = (\sum x{\in}Y.\ 2\ \widehat{\ }\ x)]\!] \Longrightarrow X = Y$
(**is** - $\Longrightarrow$ - $\Longrightarrow$ *?f X* = *?f Y* $\Longrightarrow$ -)
⟨*proof*⟩

lemma *finite-pow2-eq*:
fixes *n* :: *nat*
shows *finite* $\{i.\ 2\ \widehat{\ }\ i = n\}$
⟨*proof*⟩

lemma *finite-pow2-le*[*simp*]:
fixes *n* :: *nat*
shows *finite* $\{i.\ 2\ \widehat{\ }\ i \le n\}$
⟨*proof*⟩

lemma *le-pow2*[*simp*]: $x \le y \Longrightarrow x \le 2\ \widehat{\ }\ y$
⟨*proof*⟩

lemma *ld-bounded*: $Max\ \{i.\ 2\ \widehat{\ }\ i \le Suc\ n\} \le Suc\ n$ (**is** $?m \le Suc\ n$)
⟨*proof*⟩

primrec *satisfies0* **where**
satisfies0 $\mathfrak{A}$ (*Fo m*) = ($m^{\mathfrak{A}}FO \neq \{|\,|\}$)
| *satisfies0* $\mathfrak{A}$ (*Eq-Const i m n*) =
(*let P* = $m^{\mathfrak{A}}FO$ *in if P* = $\{|\,|\}$
then (*case i of Some i* ⇒ *Length* $\mathfrak{A}$ = *i* | - ⇒ *False*)
else fMin P = *n*)
| *satisfies0* $\mathfrak{A}$ (*Less b m1 m2*) =
(*let P1* = $m1^{\mathfrak{A}}FO$; *P2* = $m2^{\mathfrak{A}}FO$ *in if P1* = $\{|\,|\}$ ∨ *P2* = $\{|\,|\}$
then (*case b of None* ⇒ *False* | *Some True* ⇒ *P2* = $\{|\,|\}$ | *Some False* ⇒ *P1* ≠ $\{|\,|\}$)
else fMin P1 < *fMin P2*)
| *satisfies0* $\mathfrak{A}$ (*Plus-FO i m1 m2 n*) =
(*let P1* = $m1^{\mathfrak{A}}FO$; *P2* = $m2^{\mathfrak{A}}FO$ *in if P1* = $\{|\,|\}$ ∨ *P2* = $\{|\,|\}$
then (*case i of Some 0* ⇒ *P1* = *P2* | *Some i* ⇒ *P2* ≠ $\{|\,|\}$ ∧ *fMin P2* + *i* = *Length* $\mathfrak{A}$ | - ⇒ *False*)
else fMin P1 = *fMin P2* + *n*)
| *satisfies0* $\mathfrak{A}$ (*Eq-FO b m1 m2*) =
(*let P1* = $m1^{\mathfrak{A}}FO$; *P2* = $m2^{\mathfrak{A}}FO$ *in if P1* = $\{|\,|\}$ ∨ *P2* = $\{|\,|\}$
then b ∧ *P1* = *P2*
else fMin P1 = *fMin P2*)
| *satisfies0* $\mathfrak{A}$ (*Eq-SO M1 M2*) = ($M1^{\mathfrak{A}}SO = M2^{\mathfrak{A}}SO$)

| *satisfies0* $\mathfrak{A}$ (*Suc-SO br bl M1 M2*) =
((*if br then finsert* (*Length* $\mathfrak{A}$) *else id*) ($M1^{\mathfrak{A}}SO$) =
(*if bl then finsert 0 else id*) (*Suc* |`| $M2^{\mathfrak{A}}SO$))
| *satisfies0* $\mathfrak{A}$ (*Empty M*) = ($M^{\mathfrak{A}}SO = \{||\}$)
| *satisfies0* $\mathfrak{A}$ (*Singleton M*) = ($\exists x.\ M^{\mathfrak{A}}SO = \{|x|\}$)
| *satisfies0* $\mathfrak{A}$ (*Subset M1 M2*) = ($M1^{\mathfrak{A}}SO\ |\subseteq|\ M2^{\mathfrak{A}}SO$)
| *satisfies0* $\mathfrak{A}$ (*In b m M*) =
(*let* $P = m^{\mathfrak{A}}FO$ *in if* $P = \{||\}$ *then b else fMin P* $|\in|\ M^{\mathfrak{A}}SO$)
| *satisfies0* $\mathfrak{A}$ (*Eq-Max b m M*) =
(*let* $P1 = m^{\mathfrak{A}}FO$; $P2 = M^{\mathfrak{A}}SO$ *in if b then* $P1 = \{||\}$
else if $P1 = \{||\} \vee P2 = \{||\}$ *then False else fMin P1 = fMax P2*)
| *satisfies0* $\mathfrak{A}$ (*Eq-Min b m M*) =
(*let* $P1 = m^{\mathfrak{A}}FO$; $P2 = M^{\mathfrak{A}}SO$ *in if* $P1 = \{||\} \vee P2 = \{||\}$ *then* $b \wedge P1 = P2$
else fMin P1 = fMin P2)
| *satisfies0* $\mathfrak{A}$ (*Eq-Union M1 M2 M3*) = ($M1^{\mathfrak{A}}SO = M2^{\mathfrak{A}}SO\ |\cup|\ M3^{\mathfrak{A}}SO$)
| *satisfies0* $\mathfrak{A}$ (*Eq-Inter M1 M2 M3*) = ($M1^{\mathfrak{A}}SO = M2^{\mathfrak{A}}SO\ |\cap|\ M3^{\mathfrak{A}}SO$)
| *satisfies0* $\mathfrak{A}$ (*Eq-Diff M1 M2 M3*) = ($M1^{\mathfrak{A}}SO = M2^{\mathfrak{A}}SO\ |-|\ M3^{\mathfrak{A}}SO$)
| *satisfies0* $\mathfrak{A}$ (*Disjoint M1 M2*) = ($M1^{\mathfrak{A}}SO\ |\cap|\ M2^{\mathfrak{A}}SO = \{||\}$)
| *satisfies0* $\mathfrak{A}$ (*Eq-Presb i M n*) = ((($\sum x \in fset\ (M^{\mathfrak{A}}SO).\ 2\ \hat{}\ x$) = *n*) $\wedge$
(*case i of None* $\Rightarrow$ *True* | *Some l* $\Rightarrow$ *Length* $\mathfrak{A}$ = *l*))

fun *lderiv0* **where**
lderiv0 (*bs1*, *bs2*) (*Fo m*) = (*if bs1* ! *m then FBool True else FBase* (*Fo m*))
| *lderiv0* (*bs1*, *bs2*) (*Eq-Const None m n*) = (*if n* = *0* $\wedge$ *bs1* ! *m then FBool True*
else if n = *0* $\vee$ *bs1* ! *m then FBool False else FBase* (*Eq-Const None m* (*n* −
1)))
| *lderiv0* (*bs1*, *bs2*) (*Less None m1 m2*) = (*case* (*bs1* ! *m1*, *bs1* ! *m2*) *of*
(*False*, *False*) $\Rightarrow$ *FBase* (*Less None m1 m2*)
| (*True*, *False*) $\Rightarrow$ *FBase* (*Fo m2*)
| - $\Rightarrow$ *FBool False*)|
lderiv0 (*bs1*, *bs2*) (*Eq-FO False m1 m2*) = (*case* (*bs1* ! *m1*, *bs1* ! *m2*) *of*
(*False*, *False*) $\Rightarrow$ *FBase* (*Eq-FO False m1 m2*)
| (*True*, *True*) $\Rightarrow$ *FBool True*
| - $\Rightarrow$ *FBool False*)
| *lderiv0* (*bs1*, *bs2*) (*Plus-FO None m1 m2 n*) = (*if n* = *0*
then
(*case* (*bs1* ! *m1*, *bs1* ! *m2*) *of*
(*False*, *False*) $\Rightarrow$ *FBase* (*Plus-FO None m1 m2 n*)
| (*True*, *True*) $\Rightarrow$ *FBool True*
| - $\Rightarrow$ *FBool False*)
else
(*case* (*bs1* ! *m1*, *bs1* ! *m2*) *of*
(*False*, *False*) $\Rightarrow$ *FBase* (*Plus-FO None m1 m2 n*)
| (*False*, *True*) $\Rightarrow$ *FBase* (*Eq-Const None m1* (*n* − *1*))
| - $\Rightarrow$ *FBool False*))
| *lderiv0* (*bs1*, *bs2*) (*Eq-SO M1 M2*) =
(*if bs2* ! *M1* = *bs2* ! *M2 then FBase* (*Eq-SO M1 M2*) *else FBool False*)
| *lderiv0* (*bs1*, *bs2*) (*Suc-SO False bl M1 M2*) = (*if bl* = *bs2* ! *M1*

```
    then FBase (Suc-SO False (bs2 ! M2) M1 M2) else FBool False)
| lderiv0 (bs1, bs2) (Empty M) = (case bs2 ! M of
    True ⇒ FBool False
  | False ⇒ FBase (Empty M))
| lderiv0 (bs1, bs2) (Singleton M) = (case bs2 ! M of
    True ⇒ FBase (Empty M)
  | False ⇒ FBase (Singleton M))
| lderiv0 (bs1, bs2) (Subset M1 M2) = (case (bs2 ! M1, bs2 ! M2) of
    (True, False) ⇒ FBool False
  | - ⇒ FBase (Subset M1 M2))
| lderiv0 (bs1, bs2) (In False m M) = (case (bs1 ! m, bs2 ! M) of
    (False, -) ⇒ FBase (In False m M)
  | (True, True) ⇒ FBool True
  | - ⇒ FBool False)
| lderiv0 (bs1, bs2) (Eq-Max False m M) = (case (bs1 ! m, bs2 ! M) of
    (False, -) ⇒ FBase (Eq-Max False m M)
  | (True, True) ⇒ FBase (Empty M)
  | - ⇒ FBool False)
| lderiv0 (bs1, bs2) (Eq-Min False m M) = (case (bs1 ! m, bs2 ! M) of
    (False, False) ⇒ FBase (Eq-Min False m M)
  | (True, True) ⇒ FBool True
  | - ⇒ FBool False)
| lderiv0 (bs1, bs2) (Eq-Union M1 M2 M3) = (if bs2 ! M1 = (bs2 ! M2 ∨ bs2 !
M3)
   then FBase (Eq-Union M1 M2 M3) else FBool False)
| lderiv0 (bs1, bs2) (Eq-Inter M1 M2 M3) = (if bs2 ! M1 = (bs2 ! M2 ∧ bs2 !
M3)
   then FBase (Eq-Inter M1 M2 M3) else FBool False)
| lderiv0 (bs1, bs2) (Eq-Diff M1 M2 M3) = (if bs2 ! M1 = (bs2 ! M2 ∧ ¬ bs2 !
M3)
   then FBase (Eq-Diff M1 M2 M3) else FBool False)
| lderiv0 (bs1, bs2) (Disjoint M1 M2) =
  (if bs2 ! M1 ∧ bs2 ! M2 then FBool False else FBase (Disjoint M1 M2))
| lderiv0 (bs1, bs2) (Eq-Presb None M n) = (if bs2 ! M = (n mod 2 = 0)
   then FBool False else FBase (Eq-Presb None M (n div 2)))
| lderiv0 - - = undefined
```

fun *ld* **where**

```
  ld 0 = 0
| ld (Suc 0) = 0
| ld n = Suc (ld (n div 2))
```

lemma *ld-alt*[*simp*]: $n > 0 \Longrightarrow ld\ n = Max\ \{i.\ 2\ \hat{}\ i \le n\}$
⟨*proof*⟩

fun *rderiv0* **where**

```
  rderiv0 (bs1, bs2) (Fo m) = (if bs1 ! m then FBool True else FBase (Fo m))
| rderiv0 (bs1, bs2) (Eq-Const i m n) = (case bs1 ! m of
    False ⇒ FBase (Eq-Const (case i of Some (Suc i) ⇒ Some i | - ⇒ None) m n)
```

```
  | True ⇒ FBase (Eq-Const (Some n) m n))
| rderiv0 (bs1, bs2) (Less b m1 m2) = (case bs1 ! m2 of
    False ⇒ (case b of
      Some False ⇒ (case bs1 ! m1 of
        True ⇒ FBase (Less (Some True) m1 m2)
      | False ⇒ FBase (Less (Some False) m1 m2))
    | - ⇒ FBase (Less b m1 m2))
  | True ⇒ FBase (Less (Some False) m1 m2))
| rderiv0 (bs1, bs2) (Plus-FO i m1 m2 n) = (if n = 0
  then
    (case (bs1 ! m1, bs1 ! m2) of
      (False, False) ⇒ FBase (Plus-FO i m1 m2 n)
    | (True, True) ⇒ FBase (Plus-FO (Some 0) m1 m2 n)
    | - ⇒ FBase (Plus-FO None m1 m2 n))
  else
    (case bs1 ! m1 of
      True ⇒ FBase (Plus-FO (Some n) m1 m2 n)
    | False ⇒ (case bs1 ! m2 of
        False ⇒ (case i of
          Some (Suc (Suc i)) ⇒ FBase (Plus-FO (Some (Suc i)) m1 m2 n)
        | Some (Suc 0) ⇒ FBase (Plus-FO None m1 m2 n)
        | - ⇒ FBase (Plus-FO i m1 m2 n))
      | True ⇒ (case i of
          Some (Suc i) ⇒ FBase (Plus-FO (Some i) m1 m2 n)
        | - ⇒ FBase (Plus-FO None m1 m2 n)))))
| rderiv0 (bs1, bs2) (Eq-FO b m1 m2) = (case (bs1 ! m1, bs1 ! m2) of
    (False, False) ⇒ FBase (Eq-FO b m1 m2)
  | (True, True) ⇒ FBase (Eq-FO True m1 m2)
  | - ⇒ FBase (Eq-FO False m1 m2))
| rderiv0 (bs1, bs2) (Eq-SO M1 M2) =
  (if bs2 ! M1 = bs2 ! M2 then FBase (Eq-SO M1 M2) else FBool False)
| rderiv0 (bs1, bs2) (Suc-SO br bl M1 M2) = (if br = bs2 ! M2
    then FBase (Suc-SO (bs2 ! M1) bl M1 M2) else FBool False)
| rderiv0 (bs1, bs2) (Empty M) = (case bs2 ! M of
    True ⇒ FBool False
  | False ⇒ FBase (Empty M))
| rderiv0 (bs1, bs2) (Singleton M) = (case bs2 ! M of
    True ⇒ FBase (Empty M)
  | False ⇒ FBase (Singleton M))
| rderiv0 (bs1, bs2) (Subset M1 M2) = (case (bs2 ! M1, bs2 ! M2) of
    (True, False) ⇒ FBool False
  | - ⇒ FBase (Subset M1 M2))
| rderiv0 (bs1, bs2) (In b m M) = (case (bs1 ! m, bs2 ! M) of
    (True, True) ⇒ FBase (In True m M)
  | (True, False) ⇒ FBase (In False m M)
  | - ⇒ FBase (In b m M))
| rderiv0 (bs1, bs2) (Eq-Max b m M) = (case (bs1 ! m, bs2 ! M) of
    (True, True) ⇒ if b then FBool False else FBase (Eq-Max True m M)
  | (True, False) ⇒ if b then FBool False else FBase (Eq-Max False m M)
```

| (*False*, *True*) ⇒ *if b then FBase* (*Eq-Max True m M*) *else FBool False*
| (*False*, *False*) ⇒ *FBase* (*Eq-Max b m M*))
| *rderiv0* (*bs1*, *bs2*) (*Eq-Min b m M*) = (*case* (*bs1* ! *m*, *bs2* ! *M*) *of*
(*True*, *True*) ⇒ *FBase* (*Eq-Min True m M*)
| (*False*, *False*) ⇒ *FBase* (*Eq-Min b m M*)
| - ⇒ *FBase* (*Eq-Min False m M*))
| *rderiv0* (*bs1*, *bs2*) (*Eq-Union M1 M2 M3*) = (*if bs2* ! *M1* = (*bs2* ! *M2* ∨ *bs2* ! *M3*)
then FBase (*Eq-Union M1 M2 M3*) *else FBool False*)
| *rderiv0* (*bs1*, *bs2*) (*Eq-Inter M1 M2 M3*) = (*if bs2* ! *M1* = (*bs2* ! *M2* ∧ *bs2* ! *M3*)
then FBase (*Eq-Inter M1 M2 M3*) *else FBool False*)
| *rderiv0* (*bs1*, *bs2*) (*Eq-Diff M1 M2 M3*) = (*if bs2* ! *M1* = (*bs2* ! *M2* ∧ ¬ *bs2* ! *M3*)
then FBase (*Eq-Diff M1 M2 M3*) *else FBool False*)
| *rderiv0* (*bs1*, *bs2*) (*Disjoint M1 M2*) =
(*if bs2* ! *M1* ∧ *bs2* ! *M2 then FBool False else FBase* (*Disjoint M1 M2*))
| *rderiv0* (*bs1*, *bs2*) (*Eq-Presb l M n*) = (*case l of*
None ⇒ *if bs2* ! *M then*
if n = *0 then FBool False*
else let l = *ld n in FBase* (*Eq-Presb* (*Some l*) *M* (*n* − *2* ^ *l*))
else FBase (*Eq-Presb l M n*)
| *Some 0* ⇒ *FBool False*
| *Some* (*Suc l*) ⇒ *if bs2* ! *M* ∧ *n* ≥ *2* ^ *l then FBase* (*Eq-Presb* (*Some l*) *M* (*n* − *2* ^ *l*))
else if ¬ *bs2* ! *M* ∧ *n* < *2* ^ *l then FBase* (*Eq-Presb* (*Some l*) *M n*)
else FBool False)

primrec *nullable0* **where**
nullable0 (*Fo m*) = *False*
| *nullable0* (*Eq-Const i m n*) = (*i* = *Some 0*)
| *nullable0* (*Less b m1 m2*) = (*case b of None* ⇒ *False* | *Some b* ⇒ *b*)
| *nullable0* (*Plus-FO i m1 m2 n*) = (*i* = *Some 0*)
| *nullable0* (*Eq-FO b m1 m2*) = *b*
| *nullable0* (*Eq-SO M1 M2*) = *True*
| *nullable0* (*Suc-SO br bl M1 M2*) = (*bl* = *br*)
| *nullable0* (*Empty M*) = *True*
| *nullable0* (*Singleton M*) = *False*
| *nullable0* (*Subset M1 M2*) = *True*
| *nullable0* (*In b m M*) = *b*
| *nullable0* (*Eq-Max b m M*) = *b*
| *nullable0* (*Eq-Min b m M*) = *b*
| *nullable0* (*Eq-Union M1 M2 M3*) = *True*
| *nullable0* (*Eq-Inter M1 M2 M3*) = *True*
| *nullable0* (*Eq-Diff M1 M2 M3*) = *True*
| *nullable0* (*Disjoint M1 M2*) = *True*
| *nullable0* (*Eq-Presb l M n*) = (*n* = *0* ∧ (*l* = *Some 0* ∨ *l* = *None*))

definition *restrict ord P* = (*case ord of FO* ⇒ *P* ≠ {||} | *SO* ⇒ *True*)

definition *Restrict ord i* = (*case ord of FO* ⇒ *FBase* (*Fo i*) | *SO* ⇒ *FBool True*)

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

global-interpretation *WS1S*: *Formula SUC LESS assigns nvars Extend CONS SNOC Length*
extend size-atom zero σ eval downshift upshift finsert cut len restrict Restrict
lformula0 FV0 find0 wf0 decr0 satisfies0 nullable0 lderiv0 rderiv0 undefined
defines *norm* = *Formula-Operations.norm find0 decr0*
and *nFOr* = *Formula-Operations.nFOr* :: *formula* ⇒ -
and *nFAnd* = *Formula-Operations.nFAnd* :: *formula* ⇒ -
and *nFNot* = *Formula-Operations.nFNot find0 decr0* :: *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* :: - ⇒ *formula*
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-lformula* = *Formula-Operations.lformula lformula0* :: *formula* ⇒ -
and *check-eqv* = *λidx*. *DAs.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-lformula φ*)
(*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-lformula φ*) (=)
and *bounded-check-eqv* = *λidx*. *DAs.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-lformula φ*)
(*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-lformula φ*) (=)
and *automaton* = *DA.automaton*
(*λa φ*. *norm* (*deriv lderiv0* (*a* :: *atom*) *φ* :: *formula*))
⟨*proof*⟩

lemma *check-eqv-code*[*code*]: *check-eqv idx r s* =
  ((*ws1s-wf idx r* ∧ *ws1s-lformula r*) ∧ (*ws1s-wf idx s* ∧ *ws1s-lformula s*) ∧
  (*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 ⇒ *False*
  | *Some* ([], *x*) ⇒ *True*
  | *Some* (*a* # *list*, *x*) ⇒ *False*))
  ⟨*proof*⟩

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*]

lemma *check-eqv-sound*:
  $[\![\#_V\ \mathfrak{A} = idx;\ \textit{check-eqv idx}\ \varphi\ \psi]\!] \Longrightarrow (\textit{WS1S.sat}\ \mathfrak{A}\ \varphi \longleftrightarrow \textit{WS1S.sat}\ \mathfrak{A}\ \psi)$
  ⟨*proof*⟩

lemma *bounded-check-eqv-sound*:
  $[\![\#_V\ \mathfrak{A} = idx;\ \textit{bounded-check-eqv idx}\ \varphi\ \psi]\!] \Longrightarrow (\textit{WS1S.sat}_b\ \mathfrak{A}\ \varphi \longleftrightarrow \textit{WS1S.sat}_b\ \mathfrak{A}\ \psi)$
  ⟨*proof*⟩

⟨*ML*⟩

end

5 Concrete Atomic WS1S Formulas (Singleton Semantics for FO Variables)

datatype (*FOV0*: *'fo*, *SOV0*: *'so*) *atomic* =
  Fo 'fo |
  Z 'fo |
  Less 'fo 'fo |
  In 'fo 'so

derive *linorder atomic*

type-synonym *fo* = *nat*
type-synonym *so* = *nat*
type-synonym *ws1s* = (*fo*, *so*) *atomic*
type-synonym *formula* = (*ws1s*, *order*) *aformula*

primrec *wf0* **where**
  wf0 idx (*Fo m*) = *LESS FO m idx*
| *wf0 idx* (*Z m*) = *LESS FO m idx*
| *wf0 idx* (*Less m1 m2*) = (*LESS FO m1 idx* ∧ *LESS FO m2 idx*)

| *wf0 idx* (*In m M*) = (*LESS FO m idx* ∧ *LESS SO M idx*)

inductive *lformula0* **where**
  lformula0 (*Fo m*)
| *lformula0* (*Z m*)
| *lformula0* (*Less m1 m2*)
| *lformula0* (*In m M*)

code-pred *lformula0* ⟨*proof*⟩

declare *lformula0.intros*[*simp*]

inductive-cases *lformula0E*[*elim*]: *lformula0 a*

abbreviation *FV0* ≡ *case-order FOV0 SOV0*

fun *find0* **where**
  find0 FO i (*Fo m*) = (*i* = *m*)
| *find0 FO i* (*Z m*) = (*i* = *m*)
| *find0 FO i* (*Less m1 m2*) = (*i* = *m1* ∨ *i* = *m2*)
| *find0 FO i* (*In m -*) = (*i* = *m*)
| *find0 SO i* (*In - M*) = (*i* = *M*)
| *find0 - - -* = *False*

abbreviation *decr0 ord k* ≡ *map-atomic* (*case-order* (*dec k*) *id ord*) (*case-order id* (*dec k*) *ord*)

primrec *satisfies0* **where**
  satisfies0 $\mathfrak{A}$ (*Fo m*) = (∃ *x*. $m^{\mathfrak{A}}$*FO* = {|*x*|})
| *satisfies0* $\mathfrak{A}$ (*Z m*) = ($m^{\mathfrak{A}}$*FO* = {||})
| *satisfies0* $\mathfrak{A}$ (*Less m1 m2*) =
  (*let P1* = $m1^{\mathfrak{A}}$*FO*; *P2* = $m2^{\mathfrak{A}}$*FO in if* ¬(∃ *x*. *P1* = {|*x*|}) ∨ ¬(∃ *x*. *P2* = {|*x*|})
  then False
  else fthe-elem P1 < *fthe-elem P2*)
| *satisfies0* $\mathfrak{A}$ (*In m M*) =
  (*let P* = $m^{\mathfrak{A}}$*FO in if* ¬(∃ *x*. *P* = {|*x*|}) *then False else fMin P* |∈| $M^{\mathfrak{A}}$*SO*)

fun *lderiv0* **where**
  lderiv0 (*bs1*, *bs2*) (*Fo m*) = (*if bs1* ! *m then FBase* (*Z m*) *else FBase* (*Fo m*))
| *lderiv0* (*bs1*, *bs2*) (*Z m*) = (*if bs1* ! *m then FBool False else FBase* (*Z m*))
| *lderiv0* (*bs1*, *bs2*) (*Less m1 m2*) = (*case* (*bs1* ! *m1*, *bs1* ! *m2*) *of*
   (*False*, *False*) ⇒ *FBase* (*Less m1 m2*)
  | (*True*, *False*) ⇒ *FAnd* (*FBase* (*Z m1*)) (*FBase* (*Fo m2*))
  | - ⇒ *FBool False*)
| *lderiv0* (*bs1*, *bs2*) (*In m M*) = (*case* (*bs1* ! *m*, *bs2* ! *M*) *of*
   (*False*, -) ⇒ *FBase* (*In m M*)
  | (*True*, *True*) ⇒ *FBase* (*Z m*)
  | - ⇒ *FBool False*)

primrec *rev* **where**
  rev (*Fo m*) = *Fo m*
| *rev* (*Z m*) = *Z m*
| *rev* (*Less m1 m2*) = *Less m2 m1*
| *rev* (*In m M*) = *In m M*

abbreviation *rderiv0 v* ≡ *map-aformula rev id o lderiv0 v o rev*

primrec *nullable0* **where**
  nullable0 (*Fo m*) = *False*
| *nullable0* (*Z m*) = *True*
| *nullable0* (*Less m1 m2*) = *False*
| *nullable0* (*In m M*) = *False*

lemma *fimage-Suc-fsubset0*[*simp*]: *Suc* |'| *A* |⊆| {|*0*|} ⟷ *A* = {||}
  ⟨*proof*⟩

lemma *fsubset-singleton-iff*: *A* |⊆| {|*x*|} ⟷ *A* = {||} ∨ *A* = {|*x*|}
  ⟨*proof*⟩

definition *restrict ord P* = (*case ord of FO* ⇒ ∃ *x*. *P* = {|*x*|} | *SO* ⇒ *True*)
definition *Restrict ord i* = (*case ord of FO* ⇒ *FBase* (*Fo i*) | *SO* ⇒ *FBool True*)

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

global-interpretation *WS1S-Alt*: *Formula SUC LESS assigns nvars Extend CONS SNOC Length*
  extend size-atom zero σ eval downshift upshift finsert cut len restrict Restrict
  lformula0 FV0 find0 wf0 decr0 satisfies0 nullable0 lderiv0 rderiv0 undefined
  defines *norm* = *Formula-Operations.norm find0 decr0*
  and *nFOr* = *Formula-Operations.nFOr* :: *formula* ⇒ -
  and *nFAnd* = *Formula-Operations.nFAnd* :: *formula* ⇒ -
  and *nFNot* = *Formula-Operations.nFNot find0 decr0* :: *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* :: - ⇒ *formula*
  and *RESTRICT* = *Formula-Operations.RESTRICT Restrict FV0*
  and *deriv* = λ*d0* (*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-lformula* = *Formula-Operations.lformula lformula0* :: *formula* ⇒ -
and *check-eqv* = *λidx. DAs.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-lformula φ*)
(*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-lformula φ*) (=)
and *bounded-check-eqv* = *λidx. DAs.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-lformula φ*)
(*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-lformula φ*) (=)
and *automaton* = *DA.automaton*
(*λa φ. norm* (*deriv lderiv0* (*a* :: *atom*) *φ* :: *formula*))
⟨*proof*⟩

lemma *check-eqv-code*[*code*]: *check-eqv idx r s* =
((*ws1s-wf idx r* ∧ *ws1s-lformula r*) ∧ (*ws1s-wf idx s* ∧ *ws1s-lformula s*) ∧
(*case rtrancl-while* (λ(*p*, *q*). *final idx p* = *final idx q*)
(λ(*p*, *q*). *map* (*λa.* (*norm* (*deriv lderiv0 a p*), *norm* (*deriv lderiv0 a q*))) (*σ idx*))
(*norm* (*RESTRICT r*), *norm* (*RESTRICT s*)) *of*
None ⇒ *False*
| *Some* ([], *x*) ⇒ *True*
| *Some* (*a* # *list*, *x*) ⇒ *False*))
⟨*proof*⟩

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*]

lemma *check-eqv-sound*:
$[\![\#_V\ \mathfrak{A} = idx;\ check\text{-}eqv\ idx\ \varphi\ \psi]\!] \Longrightarrow (WS1S\text{-}Alt.sat\ \mathfrak{A}\ \varphi \longleftrightarrow WS1S\text{-}Alt.sat\ \mathfrak{A}\ \psi)$
⟨*proof*⟩

lemma *bounded-check-eqv-sound*:
$[\![\#_V\ \mathfrak{A} = idx;\ bounded\text{-}check\text{-}eqv\ idx\ \varphi\ \psi]\!] \Longrightarrow (WS1S\text{-}Alt.sat_b\ \mathfrak{A}\ \varphi \longleftrightarrow WS1S\text{-}Alt.sat_b\ \mathfrak{A}\ \psi)$
⟨*proof*⟩

⟨*ML*⟩

end

6 Concrete Atomic Presburger Formulas

declare [[*coercion of-bool* :: *bool* ⇒ *nat*]]
declare [[*coercion int*]]
declare [[*coercion-map map*]]
declare [[*coercion-enabled*]]

fun *len* :: *nat* ⇒ *nat* **where** — FIXME yet another logarithm
len 0 = *0*
| *len* (*Suc 0*) = *1*
| *len n* = *Suc* (*len* (*n div 2*))

lemma *len-eq0-iff*: *len n* = *0* ⟷ *n* = *0*
⟨*proof*⟩

lemma *len-mult2*[*simp*]: *len* (*2* ∗ *x*) = (*if x* = *0 then 0 else Suc* (*len x*))
⟨*proof*⟩

lemma *len-mult2'*[*simp*]: *len* (*x* ∗ *2*) = (*if x* = *0 then 0 else Suc* (*len x*))
⟨*proof*⟩

lemma *len-Suc-mult2*[*simp*]: *len* (*Suc* (*2* ∗ *x*)) = *Suc* (*len x*)
⟨*proof*⟩

lemma *len-le-iff*: *len x* ≤ *l* ⟷ *x* < *2* ^ *l*
⟨*proof*⟩

lemma *len-pow2*[*simp*]: *len* (*2* ^ *x*) = *Suc x*
⟨*proof*⟩

lemma *len-div2*[*simp*]: *len* (*x div 2*) = *len x* − *1*
⟨*proof*⟩

lemma *less-pow2-len*[*simp*]: *x* < *2* ^ *len x*
⟨*proof*⟩

lemma *len-alt*: *len x* = (*LEAST i*. *x* < *2* ^ *i*)
⟨*proof*⟩

lemma *len-mono*[*simp*]: *x* ≤ *y* ⟹ *len x* ≤ *len y*
⟨*proof*⟩

lemma *len-div-pow2*[*simp*]: *len* (*x div 2* ^ *m*) = *len x* − *m*
⟨*proof*⟩

lemma *len-mult-pow2*[*simp*]: *len* (*x* ∗ *2* ^ *m*) = (*if x* = *0 then 0 else len x* + *m*)
⟨*proof*⟩

lemma *map-index′-Suc*[*simp*]: *map-index′* (*Suc i*) *f xs* = *map-index′ i* (λ*i*. *f* (*Suc i*)) *xs*
⟨*proof*⟩

abbreviation (*input*) *zero n* ≡ *replicate n False*
abbreviation (*input*) *SUC* ≡ λ-::*unit*. *Suc*
definition *test-bit m n* ≡ (*m* :: *nat*) *div 2* ^ *n mod 2* = *1*
lemma *test-bit-eq-bit*: ‹*test-bit* = *bit*›
⟨*proof*⟩
definition *downshift m* ≡ (*m* :: *nat*) *div 2*
definition *upshift m* ≡ (*m* :: *nat*) ∗ *2*
lemma *set-bit-def*: *set-bit n m* ≡ *m* + (*if* ¬ *test-bit m n then 2* ^ *n else* (*0* :: *nat*))
⟨*proof*⟩
definition *cut-bits n m* ≡ (*m* :: *nat*) *mod 2* ^ *n*
lemma *cut-bits-eq-take-bit*: ‹*cut-bits* = *take-bit*›
⟨*proof*⟩

typedef *interp* = {(*n* :: *nat*, *xs* :: *nat list*). ∀ *x* ∈ *set xs*. *len x* ≤ *n*}
⟨*proof*⟩

setup-lifting *type-definition-interp*
type-synonym *atom* = *bool list*
type-synonym *value* = *nat*
datatype *presb* = *Eq* (*tm*: *int list*) (*const*: *int*) (*offset*: *int*)
derive *linorder list*
derive *linorder presb*
type-synonym *formula* = (*presb*, *unit*) *aformula*

lift-definition *assigns* :: *nat* ⇒ *interp* ⇒ *unit* ⇒ *value* (‹-ˉ-› [*900*, *999*, *999*] *999*)
is
λ*n* (-, *I*) -. *if n* < *length I then I* ! *n else 0* ⟨*proof*⟩

lift-definition *nvars* :: *interp* ⇒ *nat* (‹$\#_V$ -› [*1000*] *900*) **is**
λ(-, *I*). *length I* ⟨*proof*⟩

lift-definition *Length* :: *interp* ⇒ *nat* **is** λ(*n*, -). *n* ⟨*proof*⟩

lift-definition *Extend* :: *unit* ⇒ *nat* ⇒ *interp* ⇒ *value* ⇒ *interp* **is**
λ- *i* (*n*, *I*) *m*. (*max n* (*len m*), *insert-nth i m I*)
⟨*proof*⟩

lift-definition *CONS* :: *atom* ⇒ *interp* ⇒ *interp* **is**
λ*bs* (*n*, *I*). (*Suc n*, *map-index* (λ*i n*. *2* ∗ *n* + (*if bs* ! *i then 1 else 0*)) *I*)
⟨*proof*⟩

lift-definition *SNOC* :: *atom* ⇒ *interp* ⇒ *interp* **is**
λ*bs* (*n*, *I*). (*Suc n*, *map-index* (λ*i m*. *m* + (*if bs* ! *i then 2* ^ *n else 0*)) *I*)
⟨*proof*⟩

definition *extend* :: *unit* ⇒ *bool* ⇒ *atom* ⇒ *atom* **where**
extend - *b bs* ≡ *b* # *bs*

abbreviation (*input*) *size-atom* :: *atom* ⇒ *nat* **where**
size-atom ≡ *length*

definition *FV0* :: *unit* ⇒ *presb* ⇒ *nat set* **where**
FV0 - *fm* = (*case fm of Eq is* - - ⇒ {*n*. *n* < *length is* ∧ *is*!*n* ≠ *0*})

lemma *FV0-code*[*code*]:
FV0 x (*Eq is i off*) = *Option.these* (*set* (*map-index* (λ*i x*. *if x* = *0 then None else Some i*) *is*))
⟨*proof*⟩

primrec *wf0* :: *nat* ⇒ *presb* ⇒ *bool* **where**
wf0 idx (*Eq is* - -) = (*length is* = *idx*)

fun *find0* **where**
find0 (-::*unit*) *n* (*Eq is* - -) = (*is* ! *n* ≠ *0*)

primrec *decr0* **where**
decr0 (-::*unit*) *k* (*Eq is i d*) = *Eq* (*take k is* @ *drop* (*Suc k*) *is*) *i d*

definition *scalar-product* :: *nat list* ⇒ *int list* ⇒ *int* **where**
scalar-product ns is =
sum-list (*map-index* (λ*i b*. (*if i* < *length ns then ns* ! *i else 0*) ∗ *b*) *is*)

lift-definition *eval-tm* :: *interp* ⇒ *int list* ⇒ *int* **is**
λ(-, *I*). *scalar-product I* ⟨*proof*⟩

primrec *satisfies0* **where**
satisfies0 I (*Eq is i d*) = (*eval-tm I is* = *i* − (*2* ^ *Length I*) ∗ *d*)

inductive *lformula0* **where**
lformula0 (*Eq is i 0*)

code-pred *lformula0* ⟨*proof*⟩

fun *lderiv0* :: *bool list* ⇒ *presb* ⇒ *formula* **where**
lderiv0 bs (*Eq is i d*) = (*if d* ≠ *0 then undefined else*
(*let v* = *i* − *scalar-product bs is*
in if v mod 2 = *0 then FBase* (*Eq is* (*v div 2*) *0*) *else FBool False*))

fun *rderiv0* :: *bool list* ⇒ *presb* ⇒ *formula* **where**
rderiv0 bs (*Eq is i d*) =
(*let*
l = − *sum-list* [*i*. *i* ← *is*, *i* < *0*];
h = − *sum-list* [*i*. *i* ← *is*, *i* > *0*];

$d' = scalar\text{-}product\ bs\ is + 2 * d$
$in\ if\ d' \in \{min\ h\ i\ ..\ max\ l\ i\}\ then\ FBase\ (Eq\ is\ i\ d')\ else\ FBool\ False)$

primrec *nullable0* **where**
$nullable0\ (Eq\ is\ i\ off) = (i = off)$

definition $\sigma :: nat \Rightarrow atom\ list$ **where**
$\sigma\ n = List.n\text{-}lists\ n\ [True, False]$

named-theorems *Presb-simps*

lemma *nvars-Extend*[*Presb-simps*]: $\#_V\ (Extend\ ()\ i\ \mathfrak{A}\ P) = Suc\ (\#_V\ \mathfrak{A})$
⟨*proof*⟩

lemma *Length-Extend*[*Presb-simps*]: $Length\ (Extend\ ()\ i\ \mathfrak{A}\ P) = max\ (Length\ \mathfrak{A})\ (len\ P)$
⟨*proof*⟩

lemma *Length0-inj*[*Presb-simps*]: $Length\ \mathfrak{A} = 0 \Longrightarrow Length\ \mathfrak{B} = 0 \Longrightarrow \#_V\ \mathfrak{A} = \#_V\ \mathfrak{B} \Longrightarrow \mathfrak{A} = \mathfrak{B}$
⟨*proof*⟩

lemma *ex-Length0*[*Presb-simps*]: $\exists \mathfrak{A}.\ Length\ \mathfrak{A} = 0 \land \#_V\ \mathfrak{A} = idx$
⟨*proof*⟩

lemma *Extend-commute-safe*[*Presb-simps*]: $[\![j \le i; i < Suc\ (\#_V\ \mathfrak{A})]\!] \Longrightarrow$
$Extend\ k\ j\ (Extend\ k\ i\ \mathfrak{A}\ P)\ Q = Extend\ k\ (Suc\ i)\ (Extend\ k\ j\ \mathfrak{A}\ Q)\ P$
⟨*proof*⟩

lemma *Extend-commute-unsafe*[*Presb-simps*]:
$k \ne k' \Longrightarrow Extend\ k\ j\ (Extend\ k'\ i\ \mathfrak{A}\ P)\ Q = Extend\ k'\ i\ (Extend\ k\ j\ \mathfrak{A}\ Q)\ P$
⟨*proof*⟩

lemma *assigns-Extend*[*Presb-simps*]: $i < Suc\ (\#_V\ \mathfrak{A}) \Longrightarrow$
$m^{Extend\ k\ i\ \mathfrak{A}\ P}k' = (if\ k = k'\ then\ if\ m = i\ then\ P\ else\ dec\ i\ m^{\mathfrak{A}}k\ else\ m^{\mathfrak{A}}k')$
⟨*proof*⟩

lemma *assigns-SNOC-zero*[*Presb-simps*]: $m < \#_V\ \mathfrak{A} \Longrightarrow m^{SNOC\ (zero\ (\#_V\ \mathfrak{A}))\ \mathfrak{A}}k = m^{\mathfrak{A}}k$
⟨*proof*⟩

lemma *Length-CONS*[*Presb-simps*]: $Length\ (CONS\ x\ \mathfrak{A}) = Suc\ (Length\ \mathfrak{A})$
⟨*proof*⟩

lemma *Length-SNOC*[*Presb-simps*]: $Length\ (SNOC\ x\ \mathfrak{A}) = Suc\ (Length\ \mathfrak{A})$
⟨*proof*⟩

lemma *nvars-CONS*[*Presb-simps*]: $\#_V\ (CONS\ x\ \mathfrak{A}) = \#_V\ \mathfrak{A}$
⟨*proof*⟩

lemma *nvars-SNOC*[*Presb-simps*]: $\#_V$ (*SNOC* x $\mathfrak{A}$) = $\#_V$ $\mathfrak{A}$
⟨*proof*⟩

lemma *Extend-CONS*[*Presb-simps*]: $\#_V$ $\mathfrak{A}$ = *length* $x \Longrightarrow$
Extend k *0* (*CONS* x $\mathfrak{A}$) P = *CONS* (*extend* k (*test-bit* P *0*) x) (*Extend* k *0* $\mathfrak{A}$ (*downshift* P))
⟨*proof*⟩

lemma *Extend-SNOC*[*Presb-simps*]: $[\![\#_V$ $\mathfrak{A}$ = *length* x; *len* $P \leq$ *Length* (*SNOC* x $\mathfrak{A}$)$]\!] \Longrightarrow$
Extend k *0* (*SNOC* x $\mathfrak{A}$) P =
SNOC (*extend* k (*test-bit* P (*Length* $\mathfrak{A}$)) x) (*Extend* k *0* $\mathfrak{A}$ (*cut-bits* (*Length* $\mathfrak{A}$) P))
⟨*proof*⟩

lemma *odd-neq-even*:
Suc (*2* $*$ x) = *2* $*$ $y \longleftrightarrow$ *False*
2 $*$ y = *Suc* (*2* $*$ x) $\longleftrightarrow$ *False*
⟨*proof*⟩

lemma *CONS-inj*[*Presb-simps*]: *size* x = $\#_V$ $\mathfrak{A} \Longrightarrow$ *size* y = $\#_V$ $\mathfrak{A} \Longrightarrow \#_V$ $\mathfrak{A}$ = $\#_V$ $\mathfrak{B} \Longrightarrow$
CONS x $\mathfrak{A}$ = *CONS* y $\mathfrak{B} \longleftrightarrow$ (x = y $\wedge$ $\mathfrak{A}$ = $\mathfrak{B}$)
⟨*proof*⟩

lemma *mod-2-Suc-iff*:
x *mod 2* = *Suc 0* $\longleftrightarrow$ x = *Suc* (*2* $*$ (x *div 2*))
⟨*proof*⟩

lemma *CONS-surj*[*Presb-simps*]: *Length* $\mathfrak{A} \neq$ *0* $\Longrightarrow$
$\exists\, x\ \mathfrak{B}.\ \mathfrak{A}$ = *CONS* x $\mathfrak{B}$ $\wedge$ $\#_V$ $\mathfrak{B}$ = $\#_V$ $\mathfrak{A}$ $\wedge$ *size* x = $\#_V$ $\mathfrak{A}$
⟨*proof*⟩

lemma [*Presb-simps*]:
length (*extend* k b x) = *Suc* (*length* x)
downshift (*upshift* P) = P
downshift (*set-bit 0* P) = *downshift* P
test-bit (*set-bit* n P) n
$\neg$ *test-bit* (*upshift* P) *0*
len $P \leq p \Longrightarrow \neg$ *test-bit* P p
len (*cut-bits* n P) $\leq n$
len $P \leq n \Longrightarrow$ *cut-bits* n P = P
len (*upshift* P) = (*case len* P *of 0* $\Rightarrow$ *0* | *Suc* $n \Rightarrow$ *Suc* (*Suc* n))
len (*downshift* P) = (*case len* P *of 0* $\Rightarrow$ *0* | *Suc* $n \Rightarrow n$)
⟨*proof*⟩

lemma *Suc0-div-pow2-eq*: *Suc 0 div 2* ^ i = (*if* i = *0 then 1 else 0*)
⟨*proof*⟩

lemma *set-unset-bit-preserves-len*:
  assumes $x\ div\ 2\ \widehat{}\ m = 2 * q\ m < len\ x$
  shows $x + 2\ \widehat{}\ m < 2\ \widehat{}\ len\ x$
⟨*proof*⟩

lemma *len-set-bit*[*Presb-simps*]: $len\ (set\text{-}bit\ m\ P) = max\ (Suc\ m)\ (len\ P)$
⟨*proof*⟩

lemma *mod-pow2-div-pow2*:
  fixes $p\ m\ n :: nat$
  shows $m < n \Longrightarrow p\ mod\ 2\ \widehat{}\ n\ div\ 2\ \widehat{}\ m = p\ div\ 2\ \widehat{}\ m\ mod\ 2\ \widehat{}\ (n - m)$
  ⟨*proof*⟩

lemma *irrelevant-set-bit*[*simp*]:
  fixes $p\ m\ n :: nat$
  assumes $n \leq m$
  shows $(p + 2\ \widehat{}\ m)\ mod\ 2\ \widehat{}\ n = p\ mod\ 2\ \widehat{}\ n$
⟨*proof*⟩

lemma *mod-lemma*: $[\![\ (0::nat) < c;\ r < b\ ]\!] \Longrightarrow b * (q\ mod\ c) + r < b * c$
  ⟨*proof*⟩

lemma *relevant-set-bit*[*simp*]:
  fixes $p\ m\ n :: nat$
  assumes $m < n\ p\ div\ 2\ \widehat{}\ m = 2 * q$
  shows $(p + 2\ \widehat{}\ m)\ mod\ 2\ \widehat{}\ n = p\ mod\ 2\ \widehat{}\ n + 2\ \widehat{}\ m$
⟨*proof*⟩

lemma *cut-bits-set-bit*[*Presb-simps*]: $cut\text{-}bits\ n\ (set\text{-}bit\ m\ p) =$
  $(if\ n \leq m\ then\ cut\text{-}bits\ n\ p\ else\ set\text{-}bit\ m\ (cut\text{-}bits\ n\ p))$
  ⟨*proof*⟩

lemma *wf0-decr0*[*Presb-simps*]:
  $wf0\ (Suc\ idx)\ a \Longrightarrow l < Suc\ idx \Longrightarrow \neg\ find0\ k\ l\ a \Longrightarrow wf0\ idx\ (decr0\ k\ l\ a)$
  ⟨*proof*⟩

lemma *lformula0-decr0*[*Presb-simps*]: $lformula0\ a \Longrightarrow lformula0\ (decr0\ k\ l\ a)$
  ⟨*proof*⟩

abbreviation *sat0-syn* (**infix** ‹$\models$*0*› *65*) **where**
 $sat0\text{-}syn \equiv satisfies0$
abbreviation *sat-syn* (**infix** ‹$\models$› *65*) **where**
 $sat\text{-}syn \equiv Formula\text{-}Operations.satisfies\ Extend\ Length\ satisfies0$
abbreviation *sat-bounded-syn* (**infix** ‹$\models_b$› *65*) **where**
 $sat\text{-}bounded\text{-}syn \equiv Formula\text{-}Operations.satisfies\text{-}bounded\ Extend\ Length\ len\ satisfies0$

lemma *scalar-product-Nil*[*simp*]: $scalar\text{-}product\ []\ xs = 0$

$\langle proof \rangle$

lemma *scalar-product-Nil2*[*simp*]: *scalar-product xs* [] = *0*
$\langle proof \rangle$

lemma *scalar-product-Cons*[*simp*]:
scalar-product xs (*y* # *ys*) = (*case xs of x* # *xs* ⇒ *x* ∗ *y* + *scalar-product xs ys* | [] ⇒ *0*)
$\langle proof \rangle$

lemma *scalar-product-append*[*simp*]: *scalar-product ns* (*xs* @ *ys*) =
scalar-product (*take* (*length xs*) *ns*) *xs* + *scalar-product* (*drop* (*length xs*) *ns*) *ys*
$\langle proof \rangle$

lemma *scalar-product-trim*: *scalar-product ns xs* = *scalar-product* (*take* (*length xs*) *ns*) *xs*
$\langle proof \rangle$

lemma *Extend-satisfies0-decr0*[*Presb-simps*]:
assumes ¬ *find0 k i a i* < *Suc* ($\#_V$ $\mathfrak{A}$) *lformula0 a* ∨ *len P* ≤ *Length* $\mathfrak{A}$
shows *Extend k i* $\mathfrak{A}$ *P* ⊨*0 a* = $\mathfrak{A}$ ⊨*0 decr0 k i a*
$\langle proof \rangle$

lemma *scalar-product-eq0*: ∀ *c*∈*set ns*. *c* = *0* ⟹ *scalar-product ns is* = *0*
$\langle proof \rangle$

lemma *nullable0-satisfies0*[*Presb-simps*]: *Length* $\mathfrak{A}$ = *0* ⟹ *nullable0 a* = $\mathfrak{A}$ ⊨*0 a*
$\langle proof \rangle$

lemma *satisfies0-cong*: *wf0* ($\#_V$ $\mathfrak{B}$) *a* ⟹ $\#_V$ $\mathfrak{A}$ = $\#_V$ $\mathfrak{B}$ ⟹ *lformula0 a* ⟹
($\bigwedge m\ k.\ m < \#_V\ \mathfrak{B} \Longrightarrow m^{\mathfrak{A}}k = m^{\mathfrak{B}}k$) ⟹ $\mathfrak{A}$ ⊨*0 a* = $\mathfrak{B}$ ⊨*0 a*
$\langle proof \rangle$

lemma *wf-lderiv0*[*Presb-simps*]:
wf0 idx a ⟹ *lformula0 a* ⟹ *Formula-Operations.wf* (λ-. *Suc*) *wf0 idx* (*lderiv0 x a*)
$\langle proof \rangle$

lemma *lformula-lderiv0*[*Presb-simps*]:
lformula0 a ⟹ *Formula-Operations.lformula lformula0* (*lderiv0 x a*)
$\langle proof \rangle$

lemma *wf-rderiv0*[*Presb-simps*]:
wf0 idx a ⟹ *Formula-Operations.wf* (λ-. *Suc*) *wf0 idx* (*rderiv0 x a*)
$\langle proof \rangle$

lemma *find0-FV0*[*Presb-simps*]: ⟦*wf0 idx a*; *l* < *idx*⟧ ⟹ *find0 k l a* = (*l* ∈ *FV0 k a*)

$\langle proof \rangle$

lemma *FV0-less*[*Presb-simps*]: $wf0\ idx\ a \Longrightarrow v \in FV0\ k\ a \Longrightarrow v < idx$
$\langle proof \rangle$

lemma *finite-FV0*[*Presb-simps*]: $finite\ (FV0\ k\ a)$
$\langle proof \rangle$

lemma *finite-lderiv0*[*Presb-simps*]:
assumes *lformula0 a*
shows $finite\ \{\varphi.\ \exists xs.\ \varphi = fold\ (Formula\text{-}Operations.deriv\ extend\ lderiv0)\ xs\ (FBase\ a)\}$
$\langle proof \rangle$

lemma *finite-rderiv0*[*Presb-simps*]:
$finite\ \{\varphi.\ \exists xs.\ \varphi = fold\ (Formula\text{-}Operations.deriv\ extend\ rderiv0)\ xs\ (FBase\ a)\}$
$\langle proof \rangle$

lemma *scalar-product-CONS*: $length\ xs = length\ (bs :: bool\ list) \Longrightarrow$
$scalar\text{-}product\ (map\text{-}index\ (\lambda i\ n.\ 2 * n + bs\ !\ i)\ xs)\ is =$
$scalar\text{-}product\ bs\ is + 2 * scalar\text{-}product\ xs\ is$
$\langle proof \rangle$

lemma *eval-tm-CONS*[*simp*]:
$[\![length\ is \le \#_V\ \mathfrak{A};\ \#_V\ \mathfrak{A} = length\ x]\!] \Longrightarrow$
$eval\text{-}tm\ (CONS\ x\ \mathfrak{A})\ is = scalar\text{-}product\ x\ is + 2 * eval\text{-}tm\ \mathfrak{A}\ is$
$\langle proof \rangle$

lemma *satisfies-lderiv0*[*Presb-simps*]:
$[\![wf0\ (\#_V\ \mathfrak{A})\ a;\ \#_V\ \mathfrak{A} = length\ x;\ lformula0\ a]\!] \Longrightarrow \mathfrak{A} \models lderiv0\ x\ a \longleftrightarrow CONS\ x\ \mathfrak{A} \models0\ a$
$\langle proof \rangle$

lemma *satisfies-bounded-lderiv0*[*Presb-simps*]:
$[\![wf0\ (\#_V\ \mathfrak{A})\ a;\ \#_V\ \mathfrak{A} = length\ x;\ lformula0\ a]\!] \Longrightarrow \mathfrak{A} \models_b lderiv0\ x\ a \longleftrightarrow CONS\ x\ \mathfrak{A} \models0\ a$
$\langle proof \rangle$

lemma *scalar-product-SNOC*: $length\ xs = length\ (bs :: bool\ list) \Longrightarrow$
$scalar\text{-}product\ (map\text{-}index\ (\lambda i\ m.\ m + 2\ \hat{}\ a * bs\ !\ i)\ xs)\ is =$
$scalar\text{-}product\ xs\ is + 2\ \hat{}\ a * scalar\text{-}product\ bs\ is$
$\langle proof \rangle$

lemma *eval-tm-SNOC*[*simp*]:
$[\![length\ is \le \#_V\ \mathfrak{A};\ \#_V\ \mathfrak{A} = length\ x]\!] \Longrightarrow$
$eval\text{-}tm\ (SNOC\ x\ \mathfrak{A})\ is = eval\text{-}tm\ \mathfrak{A}\ is + 2\ \hat{}\ Length\ \mathfrak{A} * scalar\text{-}product\ x\ is$
$\langle proof \rangle$

lemma *Length-eq0-eval-tm-eq0*[*simp*]: $Length\ \mathfrak{A} = 0 \Longrightarrow eval\text{-}tm\ \mathfrak{A}\ is = 0$

⟨proof⟩

lemma *less-pow2*: $x < 2 \hat{} a \Longrightarrow int\ x < 2 \hat{} a$
⟨proof⟩

lemma *scalar-product-upper-bound*: $\forall x{\in}set\ b.\ len\ x \leq a \Longrightarrow$
$scalar\text{-}product\ b\ is \leq (2 \hat{} a - 1) * sum\text{-}list\ [i.\ i \leftarrow is,\ i > 0]$
⟨proof⟩

lemma *scalar-product-lower-bound*: $\forall x{\in}set\ b.\ len\ x \leq a \Longrightarrow$
$scalar\text{-}product\ b\ is \geq (2 \hat{} a - 1) * sum\text{-}list\ [i.\ i \leftarrow is,\ i < 0]$
⟨proof⟩

lemma *eval-tm-upper-bound*: $eval\text{-}tm\ \mathfrak{A}\ is \leq (2 \hat{} Length\ \mathfrak{A} - 1) * sum\text{-}list\ [i.\ i \leftarrow is,\ i > 0]$
⟨proof⟩

lemma *eval-tm-lower-bound*: $eval\text{-}tm\ \mathfrak{A}\ is \geq (2 \hat{} Length\ \mathfrak{A} - 1) * sum\text{-}list\ [i.\ i \leftarrow is,\ i < 0]$
⟨proof⟩

lemma *satisfies-bounded-rderiv0*[*Presb-simps*]:
$[\![wf0\ (\#_V\ \mathfrak{A})\ a;\ \#_V\ \mathfrak{A} = length\ x]\!] \Longrightarrow \mathfrak{A} \models_b rderiv0\ x\ a \longleftrightarrow SNOC\ x\ \mathfrak{A} \models 0\ a$
⟨proof⟩

declare [[*goals-limit* = *100*]]

global-interpretation *Presb*: *Formula*
where *SUC* = *SUC* **and** *LESS* = λ-. (<) **and** *Length* = *Length*
and *assigns* = *assigns* **and** *nvars* = *nvars* **and** *Extend* = *Extend* **and** *CONS* = *CONS* **and** *SNOC* = *SNOC*
and *extend* = *extend* **and** *size* = *size-atom* **and** *zero* = *zero* **and** *alphabet* = σ **and** *eval* = *test-bit*
and *downshift* = *downshift* **and** *upshift* = *upshift* **and** *add* = *set-bit* **and** *cut* = *cut-bits* **and** *len* = *len*
and *restrict* = λ- -. *True* **and** *Restrict* = λ- -. *FBool True* **and** *lformula0* = *lformula0*
and *FV0* = *FV0* **and** *find0* = *find0* **and** *wf0* = *wf0* **and** *decr0* = *decr0* **and** *satisfies0* = *satisfies0*
and *nullable0* = *nullable0* **and** *lderiv0* = *lderiv0* **and** *rderiv0* = *rderiv0*
and *TYPEVARS* = *undefined*
defines *norm* = *Formula-Operations.norm find0 decr0*
and *nFOr* = *Formula-Operations.nFOr* :: *formula* ⇒ -
and *nFAnd* = *Formula-Operations.nFAnd* :: *formula* ⇒ -
and *nFNot* = *Formula-Operations.nFNot find0 decr0* :: *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* (λ- -. *FBool True*) :: - ⇒ *formula*
and *RESTRICT = Formula-Operations.RESTRICT* (λ- -. *FBool True*) *FV0*
and *deriv* = λ*d0* (*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 :: *nat* ⇒ *formula* ⇒ -
and *presb-wf = Formula-Operations.wf SUC* (*wf0* :: *nat* ⇒ *presb* ⇒ -)
and *presb-lformula = Formula-Operations.lformula* (*lformula0* :: *presb* ⇒ -) :: *formula* ⇒ -
and *check-eqv* = λ*idx*. *DAs.check-eqv*
(σ *idx*) (λφ. *norm* (*RESTRICT* φ) :: *formula*)
(λ*a* φ. *norm* (*deriv* (*lderiv0* :: - ⇒ - ⇒ *formula*) (*a* :: *atom*) φ))
(*final idx*) (λφ :: *formula*. *presb-wf idx* φ ∧ *presb-lformula* φ)
(σ *idx*) (λφ. *norm* (*RESTRICT* φ) :: *formula*)
(λ*a* φ. *norm* (*deriv* (*lderiv0* :: - ⇒ - ⇒ *formula*) (*a* :: *atom*) φ))
(*final idx*) (λφ :: *formula*. *presb-wf idx* φ ∧ *presb-lformula* φ)
(=)
and *bounded-check-eqv* = λ*idx*. *DAs.check-eqv*
(σ *idx*) (λφ. *norm* (*RESTRICT* φ) :: *formula*)
(λ*a* φ. *norm* (*deriv* (*lderiv0* :: - ⇒ - ⇒ *formula*) (*a* :: *atom*) φ))
nullable (λφ :: *formula*. *presb-wf idx* φ ∧ *presb-lformula* φ)
(σ *idx*) (λφ. *norm* (*RESTRICT* φ) :: *formula*)
(λ*a* φ. *norm* (*deriv* (*lderiv0* :: - ⇒ - ⇒ *formula*) (*a* :: *atom*) φ))
nullable (λφ :: *formula*. *presb-wf idx* φ ∧ *presb-lformula* φ)
(=)
and *automaton = DA.automaton*
(λ*a* φ. *norm* (*deriv lderiv0* (*a* :: *atom*) φ :: *formula*))
⟨*proof*⟩

lemma *check-eqv-code*[*code*]: *check-eqv idx r s* =
((*presb-wf idx r* ∧ *presb-lformula r*) ∧ (*presb-wf idx s* ∧ *presb-lformula s*) ∧
(*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 ⇒ *False*
| *Some* ([], *x*) ⇒ *True*
| *Some* (*a* # *list*, *x*) ⇒ *False*))
⟨*proof*⟩

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*]

lemma *check-eqv-sound*:
$[\![\#_V\ \mathfrak{A} = idx;\ check\text{-}eqv\ idx\ \varphi\ \psi]\!] \Longrightarrow (Presb.sat\ \mathfrak{A}\ \varphi \longleftrightarrow Presb.sat\ \mathfrak{A}\ \psi)$
⟨*proof*⟩

lemma *bounded-check-eqv-sound*:
$[\![\#_V\ \mathfrak{A} = idx;\ bounded\text{-}check\text{-}eqv\ idx\ \varphi\ \psi]\!] \Longrightarrow (Presb.sat_b\ \mathfrak{A}\ \varphi \longleftrightarrow Presb.sat_b\ \mathfrak{A}\ \psi)$
⟨*proof*⟩

⟨*ML*⟩

end

7 Comparing WS1S Formulas with Presburger Formulas

lift-definition *letter-eq* :: $idx \Rightarrow nat \Rightarrow bool\ list \times bool\ list \Rightarrow bool\ list \Rightarrow bool$ **is**
$\lambda(m1,m2)\ n\ (bs1,\ bs2)\ bs.\ m1 = 0 \wedge m2 = n \wedge bs1 = [] \wedge bs2 = bs$ ⟨*proof*⟩

lemma *letter-eq*[*dest*]:
$letter\text{-}eq\ idx\ n\ a\ b \Longrightarrow (a \in set\ (WS1S\text{-}Prelim.\sigma\ idx)) = (b \in set\ (Presburger\text{-}Formula.\sigma\ n))$
⟨*proof*⟩

global-interpretation *WS1S-Presb*: *DAs*
$WS1S\text{-}Prelim.\sigma\ idx$
$(\lambda\varphi.\ norm\ (RESTRICT\ \varphi) :: (ws1s,\ order)\ aformula)$
$(\lambda a\ \varphi.\ norm\ (deriv\ lderiv0\ (a :: atom)\ \varphi))$
$(WS1S.final\ idx)$
$(\lambda\varphi :: formula.\ ws1s\text{-}wf\ idx\ \varphi \wedge ws1s\text{-}lformula\ \varphi)$
$\lambda\varphi.\ Formula.lang\ WS1S\text{-}Prelim.nvars$
$WS1S\text{-}Prelim.Extend\ WS1S\text{-}Prelim.CONS\ WS1S\text{-}Prelim.Length\ WS1S\text{-}Prelim.size\text{-}atom$
$WS1S\text{-}Formula.satisfies0\ idx\ \varphi$
$(\lambda\varphi :: formula.\ ws1s\text{-}wf\ idx\ \varphi \wedge ws1s\text{-}lformula\ \varphi)$
$\lambda\varphi.\ Formula.language\ WS1S\text{-}Prelim.assigns$
$WS1S\text{-}Prelim.nvars\ WS1S\text{-}Prelim.Extend\ WS1S\text{-}Prelim.CONS$
$WS1S\text{-}Prelim.Length\ WS1S\text{-}Prelim.size\text{-}atom\ restrict\ WS1S\text{-}Formula.FV0$
$WS1S\text{-}Formula.satisfies0\ idx\ \varphi$
$(Presburger\text{-}Formula.\sigma\ n)$
$(\lambda\varphi.\ Presburger\text{-}Formula.norm\ (Presburger\text{-}Formula.RESTRICT\ \varphi))$
$(\lambda a\ \varphi.\ Presburger\text{-}Formula.norm\ (Presburger\text{-}Formula.deriv\ Presburger\text{-}Formula.lderiv0\ a\ \varphi))$
$(Presburger\text{-}Formula.final\ n)$
$(\lambda\varphi.\ presb\text{-}wf\ n\ \varphi \wedge presb\text{-}lformula\ \varphi)$
$(\lambda\varphi.\ Formula.lang\ Presburger\text{-}Formula.nvars$
$Presburger\text{-}Formula.Extend\ Presburger\text{-}Formula.CONS\ Presburger\text{-}Formula.Length$
$Presburger\text{-}Formula.size\text{-}atom\ (\models 0)\ n\ \varphi)$

($\lambda\varphi$. *presb-wf n* φ $\wedge$ *presb-lformula* φ)
($\lambda\varphi$. *Formula.language Presburger-Formula.assigns*
Presburger-Formula.nvars Presburger-Formula.Extend Presburger-Formula.CONS
Presburger-Formula.Length Presburger-Formula.size-atom (λ- -. *True*)
Presburger-Formula.FV0 ($\models$*0*)
n φ)
letter-eq idx n
defines *check-eqv* = λ*idx n*. *DAs.check-eqv*
(σ *idx*) ($\lambda\varphi$. *norm* (*RESTRICT* φ) :: (*ws1s*, *order*) *aformula*)
(λa φ. *norm* (*deriv* (*lderiv0* :: - $\Rightarrow$ - $\Rightarrow$ *formula*) (*a* :: *atom*) φ))
(*final idx*) ($\lambda\varphi$ :: *formula*. *ws1s-wf idx* φ $\wedge$ *ws1s-lformula* φ)
(*Presburger-Formula.*σ *n*) ($\lambda\varphi$. *Presburger-Formula.norm* (*Presburger-Formula.RESTRICT* φ))
(λa φ. *Presburger-Formula.norm* (*Presburger-Formula.deriv Presburger-Formula.lderiv0* *a* φ))
(*Presburger-Formula.final n*) ($\lambda\varphi$. *presb-wf n* φ $\wedge$ *presb-lformula* φ) (*letter-eq* *idx n*)
⟨*proof*⟩

lemma *check-eqv-code*[*code*]: *check-eqv idx n r s* $\longleftrightarrow$
((*ws1s-wf idx r* $\wedge$ *ws1s-lformula r*) $\wedge$ (*presb-wf n s* $\wedge$ *presb-lformula s*) $\wedge$
(*case rtrancl-while* (λ(*p*, *q*). *final idx p* = *Presburger-Formula.final n q*)
(λ(*p*, *q*).
map (λ(*a*, *b*). (*norm* (*deriv lderiv0 a p*),
Presburger-Formula.norm (*Presburger-Formula.deriv Presburger-Formula.lderiv0* *b q*)))
[(*x*, *y*)←*List.product* (σ *idx*) (*Presburger-Formula.*σ *n*). *letter-eq idx n x y*])
(*norm* (*RESTRICT r*), *Presburger-Formula.norm* (*Presburger-Formula.RESTRICT* *s*)) *of*
None $\Rightarrow$ *False*
| *Some* ([], *x*) $\Rightarrow$ *True*
| *Some* (*a* # *list*, *x*) $\Rightarrow$ *False*))
⟨*proof*⟩

⟨*ML*⟩

8 Nameful WS1S Formulas

declare [[*coercion of-char* :: *char* $\Rightarrow$ *nat*, *coercion-enabled*]]

definition *is-upper* :: *char* $\Rightarrow$ *bool* **where** [*simp*]: *is-upper c* = (*c* $\in$ {*65..90* :: *nat*})
definition *is-lower* :: *char* $\Rightarrow$ *bool* **where** [*simp*]: *is-lower c* = (*c* $\in$ {*97..122* :: *nat*})

typedef *fo* = {*s*. *s* $\neq$ [] $\wedge$ *is-lower* (*hd s*)} ⟨*proof*⟩

typedef $so = \{s.\ s \neq [] \wedge \textit{is-upper}\ (hd\ s)\}$ $\langle proof \rangle$

datatype *ws1s* =
  T | *F* | *Or ws1s ws1s* | *And ws1s ws1s* | *Not ws1s*
  | *Ex1 fo ws1s* | *Ex2 so ws1s* | *All1 fo ws1s* | *All2 so ws1s*
  | *Lt fo fo*
  | *In fo so*
  | *Eq-Const fo nat*
  | *Eq-Presb so nat*
  | *Eq-FO fo fo*
  | *Eq-FO-Offset fo fo nat*
  | *Eq-SO so so*
  | *Eq-SO-Inc so so*
  | *Eq-Max fo so*
  | *Eq-Min fo so*
  | *Empty so*
  | *Singleton so*
  | *Subset so so*
  | *Disjoint so so*
  | *Eq-Union so so so*
  | *Eq-Inter so so so*
  | *Eq-Diff so so so*

primrec *satisfies* :: $(fo \Rightarrow nat) \Rightarrow (so \Rightarrow nat\ fset) \Rightarrow ws1s \Rightarrow bool$ **where**
  $satisfies\ I1\ I2\ T = True$
| $satisfies\ I1\ I2\ F = False$
| $satisfies\ I1\ I2\ (Or\ \varphi\ \psi) = (satisfies\ I1\ I2\ \varphi \vee satisfies\ I1\ I2\ \psi)$
| $satisfies\ I1\ I2\ (And\ \varphi\ \psi) = (satisfies\ I1\ I2\ \varphi \wedge satisfies\ I1\ I2\ \psi)$
| $satisfies\ I1\ I2\ (Not\ \varphi) = (\neg\ satisfies\ I1\ I2\ \varphi)$
| $satisfies\ I1\ I2\ (Ex1\ x\ \varphi) = (\exists\, n.\ satisfies\ (I1(x := n))\ I2\ \varphi)$
| $satisfies\ I1\ I2\ (Ex2\ X\ \varphi) = (\exists\, N.\ satisfies\ I1\ (I2(X := N))\ \varphi)$
| $satisfies\ I1\ I2\ (All1\ x\ \varphi) = (\forall\, n.\ satisfies\ (I1(x := n))\ I2\ \varphi)$
| $satisfies\ I1\ I2\ (All2\ X\ \varphi) = (\forall\, N.\ satisfies\ I1\ (I2(X := N))\ \varphi)$

| $satisfies\ I1\ I2\ (Lt\ x\ y) = (I1\ x < I1\ y)$
| $satisfies\ I1\ I2\ (In\ x\ X) = (I1\ x\ |{\in}|\ I2\ X)$
| $satisfies\ I1\ I2\ (Eq\text{-}Const\ x\ n) = (I1\ x = n)$
| $satisfies\ I1\ I2\ (Eq\text{-}Presb\ X\ n) = ((\sum x \in fset\ (I2\ X).\ 2\ \hat{}\ x) = n)$
| $satisfies\ I1\ I2\ (Eq\text{-}FO\ x\ y) = (I1\ x = I1\ y)$
| $satisfies\ I1\ I2\ (Eq\text{-}FO\text{-}Offset\ x\ y\ n) = (I1\ x = I1\ y + n)$
| $satisfies\ I1\ I2\ (Eq\text{-}SO\ X\ Y) = (I2\ X = I2\ Y)$
| $satisfies\ I1\ I2\ (Eq\text{-}SO\text{-}Inc\ X\ Y) = (I2\ X = Suc\ |`|\ I2\ Y)$
| $satisfies\ I1\ I2\ (Eq\text{-}Max\ x\ X) = (let\ Z = I2\ X\ in\ Z \neq \{||\} \wedge I1\ x = fMax\ Z)$
| $satisfies\ I1\ I2\ (Eq\text{-}Min\ x\ X) = (let\ Z = I2\ X\ in\ Z \neq \{||\} \wedge I1\ x = fMin\ Z)$
| $satisfies\ I1\ I2\ (Empty\ X) = (I2\ X = \{||\})$
| $satisfies\ I1\ I2\ (Singleton\ X) = (\exists\, x.\ I2\ X = \{|x|\})$
| $satisfies\ I1\ I2\ (Subset\ X\ Y) = (I2\ X\ |{\subseteq}|\ I2\ Y)$

| $satisfies\ I1\ I2\ (Disjoint\ X\ Y) = (I2\ X\ |\cap|\ I2\ Y = \{||\})$
| $satisfies\ I1\ I2\ (Eq\text{-}Union\ X\ Y\ Z) = (I2\ X = I2\ Y\ |\cup|\ I2\ Z)$
| $satisfies\ I1\ I2\ (Eq\text{-}Inter\ X\ Y\ Z) = (I2\ X = I2\ Y\ |\cap|\ I2\ Z)$
| $satisfies\ I1\ I2\ (Eq\text{-}Diff\ X\ Y\ Z) = (I2\ X = I2\ Y\ |-|\ I2\ Z)$

primrec *fos* **where**
$fos\ T = []$
| $fos\ F = []$
| $fos\ (Or\ \varphi\ \psi) = List.union\ (fos\ \varphi)\ (fos\ \psi)$
| $fos\ (And\ \varphi\ \psi) = List.union\ (fos\ \varphi)\ (fos\ \psi)$
| $fos\ (Not\ \varphi) = fos\ \varphi$
| $fos\ (Ex1\ x\ \varphi) = List.remove1\ x\ (fos\ \varphi)$
| $fos\ (Ex2\ X\ \varphi) = fos\ \varphi$
| $fos\ (All1\ x\ \varphi) = List.remove1\ x\ (fos\ \varphi)$
| $fos\ (All2\ X\ \varphi) = fos\ \varphi$
| $fos\ (Lt\ x\ y) = remdups\ [x,\ y]$
| $fos\ (In\ x\ X) = [x]$
| $fos\ (Eq\text{-}Const\ x\ n) = [x]$
| $fos\ (Eq\text{-}Presb\ X\ n) = []$
| $fos\ (Eq\text{-}FO\ x\ y) = remdups\ [x,\ y]$
| $fos\ (Eq\text{-}FO\text{-}Offset\ x\ y\ n) = remdups\ [x,\ y]$
| $fos\ (Eq\text{-}SO\ X\ Y) = []$
| $fos\ (Eq\text{-}SO\text{-}Inc\ X\ Y) = []$
| $fos\ (Eq\text{-}Max\ x\ X) = [x]$
| $fos\ (Eq\text{-}Min\ x\ X) = [x]$
| $fos\ (Empty\ X) = []$
| $fos\ (Singleton\ X) = []$
| $fos\ (Subset\ X\ Y) = []$
| $fos\ (Disjoint\ X\ Y) = []$
| $fos\ (Eq\text{-}Union\ X\ Y\ Z) = []$
| $fos\ (Eq\text{-}Inter\ X\ Y\ Z) = []$
| $fos\ (Eq\text{-}Diff\ X\ Y\ Z) = []$

primrec *sos* **where**
$sos\ T = []$
| $sos\ F = []$
| $sos\ (Or\ \varphi\ \psi) = List.union\ (sos\ \varphi)\ (sos\ \psi)$
| $sos\ (And\ \varphi\ \psi) = List.union\ (sos\ \varphi)\ (sos\ \psi)$
| $sos\ (Not\ \varphi) = sos\ \varphi$
| $sos\ (Ex1\ x\ \varphi) = sos\ \varphi$
| $sos\ (Ex2\ X\ \varphi) = List.remove1\ X\ (sos\ \varphi)$
| $sos\ (All1\ x\ \varphi) = sos\ \varphi$
| $sos\ (All2\ X\ \varphi) = List.remove1\ X\ (sos\ \varphi)$
| $sos\ (Lt\ x\ y) = []$
| $sos\ (In\ x\ X) = [X]$
| $sos\ (Eq\text{-}Const\ x\ n) = []$
| $sos\ (Eq\text{-}Presb\ X\ n) = [X]$
| $sos\ (Eq\text{-}FO\ x\ y) = []$
| $sos\ (Eq\text{-}FO\text{-}Offset\ x\ y\ n) = []$

| sos ($Eq\text{-}SO$ X Y) = $remdups$ [X, Y]
| sos ($Eq\text{-}SO\text{-}Inc$ X Y) = $remdups$ [X, Y]
| sos ($Eq\text{-}Max$ x X) = [X]
| sos ($Eq\text{-}Min$ x X) = [X]
| sos ($Empty$ X) = [X]
| sos ($Singleton$ X) = [X]
| sos ($Subset$ X Y) = $remdups$ [X, Y]
| sos ($Disjoint$ X Y) = $remdups$ [X, Y]
| sos ($Eq\text{-}Union$ X Y Z) = $remdups$ [X, Y, Z]
| sos ($Eq\text{-}Inter$ X Y Z) = $remdups$ [X, Y, Z]
| sos ($Eq\text{-}Diff$ X Y Z) = $remdups$ [X, Y, Z]

lemma $distinct\text{-}fos$[$simp$]: $distinct$ (fos φ) $\langle proof \rangle$
lemma $distinct\text{-}sos$[$simp$]: $distinct$ (sos φ) $\langle proof \rangle$

primrec ε **where**
ε $bs1$ $bs2$ T = $FBool$ $True$
| ε $bs1$ $bs2$ F = $FBool$ $False$
| ε $bs1$ $bs2$ (Or φ ψ) = FOr (ε $bs1$ $bs2$ φ) (ε $bs1$ $bs2$ ψ)
| ε $bs1$ $bs2$ (And φ ψ) = $FAnd$ (ε $bs1$ $bs2$ φ) (ε $bs1$ $bs2$ ψ)
| ε $bs1$ $bs2$ (Not φ) = $FNot$ (ε $bs1$ $bs2$ φ)
| ε $bs1$ $bs2$ ($Ex1$ x φ) = FEx FO (ε (x # $bs1$) $bs2$ φ)
| ε $bs1$ $bs2$ ($Ex2$ X φ) = FEx SO (ε $bs1$ (X # $bs2$) φ)
| ε $bs1$ $bs2$ ($All1$ x φ) = $FAll$ FO (ε (x # $bs1$) $bs2$ φ)
| ε $bs1$ $bs2$ ($All2$ X φ) = $FAll$ SO (ε $bs1$ (X # $bs2$) φ)
| ε $bs1$ $bs2$ (Lt x y) = $FBase$ ($Less$ $None$ ($index$ $bs1$ x) ($index$ $bs1$ y))
| ε $bs1$ $bs2$ (In x X) = $FBase$ ($WS1S\text{-}Formula.In$ $False$ ($index$ $bs1$ x) ($index$ $bs2$ X))
| ε $bs1$ $bs2$ ($Eq\text{-}Const$ x n) = $FBase$ ($WS1S\text{-}Formula.Eq\text{-}Const$ $None$ ($index$ $bs1$ x) n)
| ε $bs1$ $bs2$ ($Eq\text{-}Presb$ X n) = $FBase$ ($WS1S\text{-}Formula.Eq\text{-}Presb$ $None$ ($index$ $bs2$ X) n)
| ε $bs1$ $bs2$ ($Eq\text{-}FO$ x y) = $FBase$ ($WS1S\text{-}Formula.Eq\text{-}FO$ $False$ ($index$ $bs1$ x) ($index$ $bs1$ y))
| ε $bs1$ $bs2$ ($Eq\text{-}FO\text{-}Offset$ x y n) = $FBase$ ($WS1S\text{-}Formula.Plus\text{-}FO$ $None$ ($index$ $bs1$ x) ($index$ $bs1$ y) n)
| ε $bs1$ $bs2$ ($Eq\text{-}SO$ X Y) = $FBase$ ($WS1S\text{-}Formula.Eq\text{-}SO$ ($index$ $bs2$ X) ($index$ $bs2$ Y))
| ε $bs1$ $bs2$ ($Eq\text{-}SO\text{-}Inc$ X Y) = $FBase$ ($WS1S\text{-}Formula.Suc\text{-}SO$ $False$ $False$ ($index$ $bs2$ X) ($index$ $bs2$ Y))
| ε $bs1$ $bs2$ ($Eq\text{-}Max$ x X) = $FBase$ ($WS1S\text{-}Formula.Eq\text{-}Max$ $False$ ($index$ $bs1$ x) ($index$ $bs2$ X))
| ε $bs1$ $bs2$ ($Eq\text{-}Min$ x X) = $FBase$ ($WS1S\text{-}Formula.Eq\text{-}Min$ $False$ ($index$ $bs1$ x) ($index$ $bs2$ X))
| ε $bs1$ $bs2$ ($Empty$ X) = $FBase$ ($WS1S\text{-}Formula.Empty$ ($index$ $bs2$ X))
| ε $bs1$ $bs2$ ($Singleton$ X) = $FBase$ ($WS1S\text{-}Formula.Singleton$ ($index$ $bs2$ X))
| ε $bs1$ $bs2$ ($Subset$ X Y) = $FBase$ ($WS1S\text{-}Formula.Subset$ ($index$ $bs2$ X) ($index$ $bs2$ Y))
| ε $bs1$ $bs2$ ($Disjoint$ X Y) = $FBase$ ($WS1S\text{-}Formula.Disjoint$ ($index$ $bs2$ X) ($index$

bs2 Y))
| ε *bs1 bs2* (*Eq-Union X Y Z*) = *FBase* (*WS1S-Formula.Eq-Union* (*index bs2 X*) (*index bs2 Y*) (*index bs2 Z*))
| ε *bs1 bs2* (*Eq-Inter X Y Z*) = *FBase* (*WS1S-Formula.Eq-Inter* (*index bs2 X*) (*index bs2 Y*) (*index bs2 Z*))
| ε *bs1 bs2* (*Eq-Diff X Y Z*) = *FBase* (*WS1S-Formula.Eq-Diff* (*index bs2 X*) (*index bs2 Y*) (*index bs2 Z*))

lift-definition *mk-I* ::
(*fo* ⇒ *nat*) ⇒ (*so* ⇒ *nat fset*) ⇒ *fo list* ⇒ *so list* ⇒ *interp* **is**
λ*I1 I2 fs ss. let I1s* = *map* (λ*x.* {|*I1 x*|}) *fs*; *I2s* = *map I2 ss in* (*MSB* (*I1s* @ *I2s*), *I1s*, *I2s*)
⟨*proof*⟩

definition *dec-I1* :: *interp* ⇒ *fo list* ⇒ *fo* ⇒ *nat* **where** *dec-I1* $\mathfrak{A}$ *fs x* = *fMin* (*index fs* $x^{\mathfrak{A}}$*FO*)
definition *dec-I2* :: *interp* ⇒ *so list* ⇒ *so* ⇒ *nat fset* **where** *dec-I2* $\mathfrak{A}$ *ss X* = *index ss* $X^{\mathfrak{A}}$*SO*

lemma *nvars-mk-I*[*simp*]: $\#_V$ (*mk-I I1 I2 fs ss*) = *Abs-idx* (*length fs*, *length ss*)
⟨*proof*⟩

lemma *assigns-mk-I-FO*[*simp*]:
$m^{mk\text{-}I\ I1\ I2\ bs1\ bs2}$*FO* = (*if m* < *length bs1 then* {|*I1* (*bs1* ! *m*)|} *else* {||})
⟨*proof*⟩

lemma *assigns-mk-I-SO*[*simp*]:
$m^{mk\text{-}I\ I1\ I2\ bs1\ bs2}$*SO* = (*if m* < *length bs2 then I2* (*bs2* ! *m*) *else* {||})
⟨*proof*⟩

lemma *satisfies-cong*:
⟦∀ *x* ∈ *set* (*fos* φ). *I1 x* = *J1 x*; ∀ *X* ∈ *set* (*sos* φ). *I2 X* = *J2 X*⟧ ⟹
satisfies I1 I2 φ ⟷ *satisfies J1 J2* φ
⟨*proof*⟩

lemma *dec-I-mk-I-satisfies-cong*:
⟦*set* (*fos* φ) ⊆ *set bs1*; *set* (*sos* φ) ⊆ *set bs2*; $\mathfrak{A}$ = *mk-I I1 I2 bs1 bs2*⟧ ⟹
satisfies (*dec-I1* $\mathfrak{A}$ *bs1*) (*dec-I2* $\mathfrak{A}$ *bs2*) φ ⟷ *satisfies I1 I2* φ
⟨*proof*⟩

definition *ok-I* $\mathfrak{A}$ *fs* = (∀ *x* ∈ *set fs. index fs* $x^{\mathfrak{A}}$*FO* ≠ {||})

lemma *ok-I-mk-I*[*simp*]: *ok-I* (*mk-I I1 I2 bs1 bs2*) *bs1*
⟨*proof*⟩

lemma *in-FV-*ε*D*[*dest*]: ⟦*v* ∈ *FV* (ε *bs1 bs2* φ) *FO*;
set (*fos* φ) ⊆ *set bs1*; *set* (*sos* φ) ⊆ *set bs2*⟧ ⟹
(∃ *y* ∈ *set bs1*. *v* = *index bs1 y*)
⟨*proof*⟩

lemma *dec-I1-Extend-FO*[*simp*]:
dec-I1 (*Extend FO 0* $\mathfrak{A}$ *P*) (*x* # *bs1*) = (*dec-I1* $\mathfrak{A}$ *bs1*)(*x* := *fMin P*)
⟨*proof*⟩

lemma *dec-I1-Extend-SO*[*simp*]: *dec-I1* (*Extend SO i* $\mathfrak{A}$ *P*) *bs1* = *dec-I1* $\mathfrak{A}$ *bs1*
⟨*proof*⟩

lemma *dec-I2-Extend-FO*[*simp*]: *dec-I2* (*Extend FO i* $\mathfrak{A}$ *P*) *bs2* = *dec-I2* $\mathfrak{A}$ *bs2*
⟨*proof*⟩

lemma *dec-I2-Extend-SO*[*simp*]:
dec-I2 (*Extend SO 0* $\mathfrak{A}$ *P*) (*X* # *bs2*) = (*dec-I2* $\mathfrak{A}$ *bs2*)(*X* := *P*)
⟨*proof*⟩

lemma *sat-ε*: ⟦*set* (*fos* φ) ⊆ *set bs1*; *set* (*sos* φ) ⊆ *set bs2*; *ok-I* $\mathfrak{A}$ *bs1*⟧ ⟹
WS1S.sat $\mathfrak{A}$ (ε *bs1 bs2* φ) ⟷
satisfies (*dec-I1* $\mathfrak{A}$ *bs1*) (*dec-I2* $\mathfrak{A}$ *bs2*) φ
⟨*proof*⟩

definition *eqv* φ ψ =
(*let fs* = *List.union* (*fos* φ) (*fos* ψ); *ss* = *List.union* (*sos* φ) (*sos* ψ)
in check-eqv (*Abs-idx* (*length fs*, *length ss*)) (ε *fs ss* φ) (ε *fs ss* ψ))

lemma *eqv-sound*: *eqv* φ ψ ⟹ *satisfies I1 I2* φ ⟷ *satisfies I1 I2* ψ
⟨*proof*⟩

definition *Thm* φ = *eqv* φ *T*

lemma *Thm* Φ ⟹ *satisfies I1 I2* Φ
⟨*proof*⟩

setup-lifting *type-definition-fo*
setup-lifting *type-definition-so*

instantiation *fo* :: *equal*
begin
lift-definition *equal-fo* :: *fo* ⇒ *fo* ⇒ *bool* **is** (=) ⟨*proof*⟩
instance ⟨*proof*⟩
end

instantiation *so* :: *equal*
begin
lift-definition *equal-so* :: *so* ⇒ *so* ⇒ *bool* **is** (=) ⟨*proof*⟩
instance ⟨*proof*⟩
end