Decision Procedures for MSO on Words Based on Derivatives of Regular Expressions

Dmitriy Traytel and Tobias Nipkow

March 17, 2025

Abstract

Monadic second-order logic on finite words (MSO) is a decidable yet expressive logic into which many decision problems can be encoded. Since MSO formulas correspond to regular languages, equivalence of MSO formulas can be reduced to the equivalence of some regular structures (e.g. automata). We verify an executable decision procedure for MSO formulas that is not based on automata but on regular expressions.

Decision procedures for regular expression equivalence have been formalized before (e.g. in Isabelle/HOL [1]), usually based on Brzozowski derivatives. Yet, for a straightforward embedding of MSO formulas into regular expressions an extension of regular expressions with a projection operation is required. We prove total correctness and completeness of an equivalence checker for regular expressions extended in that way. We also define a language-preserving translation of formulas into regular expressions with respect to two different semantics of MSO.

The formalization is described in the ICFP 2013 functional pearl [2].

Contents

1 Regular Sets **3**
- 1.1 Concatenation of Languages 3
- 1.2 Iteration of Languages 4
- 1.3 Left-Quotients of Languages 6
- 1.4 Right-Quotients of Languages 7
- 1.5 Two-Sided-Quotients of Languages 8
- 1.6 Arden's Lemma 9
- 1.7 Lists of Fixed Length 10

2 Π-Extended Regular Expressions **10**
- 2.1 Syntax of regular expressions 10
- 2.2 ACI normalization 11

2.3 Finality 14
2.4 Wellformedness w.r.t. an alphabet 14
2.5 Language 16

3 Derivatives of Π-Extended Regular Expressions 17
3.1 Syntactic Derivatives 18
3.2 Finiteness of ACI-Equivalent Derivatives 18
3.3 Wellformedness and language of derivatives 20
3.4 Deriving preserves ACI-equivalence 20

4 Some Useful Regular Operators 21
4.1 Quotioning by the same letter 24
4.2 Suffix and Prefix Languages 27

5 Π-Extended Dual Regular Expressions 28
5.1 Syntax of regular expressions 28

6 Deciding Equivalence of Π-Extended Regular Expressions 33

7 Initial Normalization of the Input 41

8 Partial Derivatives-like Normalization 46

9 Monadic Second-Order Logic Formulas 48
9.1 Interpretations and Encodings 48
9.2 Syntax and Semantics of MSO 48
9.3 ENC 50

10 M2L 52
10.1 Encodings 52
10.2 Welldefinedness of enc wrt. Models 55
10.3 From M2L to Regular expressions 56

11 Normalization of M2L Formulas 59

12 Deciding Equivalence of M2L Formulas 60

13 WS1S 64
13.1 Encodings 64
13.2 Welldefinedness of enc wrt. Models 70
13.3 From WS1S to Regular expressions 71

14 Normalization of WS1S Formulas 76

15 Deciding Equivalence of WS1S Formulas 77

1 Regular Sets

type-synonym $'a$ *lang* = $'a$ *list set*

definition *conc* :: $'a$ *lang* ⇒ $'a$ *lang* ⇒ $'a$ *lang* (**infixr** ‹@@› *75*) **where**
A @@ *B* = {*xs*@*ys* | *xs ys*. *xs*:*A* & *ys*:*B*}

lemma [*code*]:
A @@ *B* = (%(*xs*, *ys*). *xs* @ *ys*) ‘ (*A* × *B*)
⟨*proof*⟩

overloading *lang-pow* == *compow* :: *nat* ⇒ $'a$ *lang* ⇒ $'a$ *lang*
begin
primrec *lang-pow* :: *nat* ⇒ $'a$ *lang* ⇒ $'a$ *lang* **where**
lang-pow 0 A = {[]} |
lang-pow (*Suc n*) *A* = *A* @@ (*lang-pow n A*)
end

definition *star* :: $'a$ *lang* ⇒ $'a$ *lang* **where**
star A = (⋃*n*. *A* ^^ *n*)

1.1 Concatenation of Languages

lemma *concI*[*simp*,*intro*]: *u* : *A* ⟹ *v* : *B* ⟹ *u*@*v* : *A* @@ *B*
⟨*proof*⟩

lemma *concE*[*elim*]:
assumes *w* ∈ *A* @@ *B*
obtains *u v* **where** *u* ∈ *A v* ∈ *B w* = *u*@*v*
⟨*proof*⟩

lemma *conc-mono*: *A* ⊆ *C* ⟹ *B* ⊆ *D* ⟹ *A* @@ *B* ⊆ *C* @@ *D*
⟨*proof*⟩

lemma *conc-empty*[*simp*]: **shows** {} @@ *A* = {} **and** *A* @@ {} = {}
⟨*proof*⟩

lemma *conc-epsilon*[*simp*]: **shows** {[]} @@ *A* = *A* **and** *A* @@ {[]} = *A*
⟨*proof*⟩

lemma *conc-assoc*: (*A* @@ *B*) @@ *C* = *A* @@ (*B* @@ *C*)
⟨*proof*⟩

lemma *conc-Un-distrib*:
shows *A* @@ (*B* ∪ *C*) = *A* @@ *B* ∪ *A* @@ *C*
and (*A* ∪ *B*) @@ *C* = *A* @@ *C* ∪ *B* @@ *C*
⟨*proof*⟩

lemma *conc-UNION-distrib*:

shows A @@ $\bigcup(M$ ‘ $I) = \bigcup((\%i.\ A$ @@ $M\ i)$ ‘ $I)$
and $\bigcup(M$ ‘ $I)$ @@ $A = \bigcup((\%i.\ M\ i$ @@ $A)$ ‘ $I)$
⟨*proof*⟩

lemma *hom-image-conc*: $[\![\bigwedge xs\ ys.\ f\ (xs\ @\ ys) = f\ xs\ @\ f\ ys]\!] \Longrightarrow f$ ‘ $(A$ @@ $B) = f$ ‘ A @@ f ‘ B
⟨*proof*⟩

lemma *map-image-conc*[*simp*]: *map f* ‘ $(A$ @@ $B) =$ *map f* ‘ A @@ *map f* ‘ B
⟨*proof*⟩

lemma *conc-subset-lists*: $A \subseteq$ *lists* $S \Longrightarrow B \subseteq$ *lists* $S \Longrightarrow A$ @@ $B \subseteq$ *lists* S
⟨*proof*⟩

1.2 Iteration of Languages

lemma *lang-pow-add*: A ^^ $(n + m) = A$ ^^ n @@ A ^^ m
⟨*proof*⟩

lemma *lang-pow-simps*: $(A$ ^^ *Suc* $n) = (A$ ^^ n @@ $A)$
⟨*proof*⟩

lemma *lang-pow-empty*: $\{\}$ ^^ $n = ($*if* $n = 0$ *then* $\{[]\}$ *else* $\{\})$
⟨*proof*⟩

lemma *lang-pow-empty-Suc*[*simp*]: $(\{\}::'a\ lang)$ ^^ *Suc* $n = \{\}$
⟨*proof*⟩

lemma *conc-pow-comm*:
shows A @@ $(A$ ^^ $n) = (A$ ^^ $n)$ @@ A
⟨*proof*⟩

lemma *length-lang-pow-ub*:
ALL $w : A.$ *length* $w \leq k \Longrightarrow w : A$^^$n \Longrightarrow$ *length* $w \leq k{*}n$
⟨*proof*⟩

lemma *length-lang-pow-lb*:
ALL $w : A.$ *length* $w \geq k \Longrightarrow w : A$^^$n \Longrightarrow$ *length* $w \geq k{*}n$
⟨*proof*⟩

lemma *lang-pow-subset-lists*: $A \subseteq$ *lists* $S \Longrightarrow A$ ^^ $n \subseteq$ *lists* S
⟨*proof*⟩

lemma *star-subset-lists*: $A \subseteq$ *lists* $S \Longrightarrow$ *star* $A \subseteq$ *lists* S
⟨*proof*⟩

lemma *star-if-lang-pow*[*simp*]: $w : A$ ^^ $n \Longrightarrow w :$ *star* A
⟨*proof*⟩

lemma *Nil-in-star*[*iff*]: [] : *star A*
⟨*proof*⟩

lemma *star-if-lang*[*simp*]: **assumes** *w* : *A* **shows** *w* : *star A*
⟨*proof*⟩

lemma *append-in-starI*[*simp*]:
assumes *u* : *star A* **and** *v* : *star A* **shows** *u*@*v* : *star A*
⟨*proof*⟩

lemma *conc-star-star*: *star A* @@ *star A* = *star A*
⟨*proof*⟩

lemma *conc-star-comm*:
shows *A* @@ *star A* = *star A* @@ *A*
⟨*proof*⟩

lemma *star-induct*[*consumes 1*, *case-names Nil append*, *induct set*: *star*]:
assumes *w* : *star A*
and *P* []
and *step*: !!*u v*. *u* : *A* $\Longrightarrow$ *v* : *star A* $\Longrightarrow$ *P v* $\Longrightarrow$ *P* (*u*@*v*)
shows *P w*
⟨*proof*⟩

lemma *star-empty*[*simp*]: *star* {} = {[]}
⟨*proof*⟩

lemma *star-epsilon*[*simp*]: *star* {[]} = {[]}
⟨*proof*⟩

lemma *star-idemp*[*simp*]: *star* (*star A*) = *star A*
⟨*proof*⟩

lemma *star-unfold-left*: *star A* = *A* @@ *star A* ∪ {[]} (**is** *?L* = *?R*)
⟨*proof*⟩

lemma *concat-in-star*: *set ws* ⊆ *A* $\Longrightarrow$ *concat ws* : *star A*
⟨*proof*⟩

lemma *in-star-iff-concat*:
w : *star A* = (*EX ws*. *set ws* ⊆ *A* & *w* = *concat ws* & [] ∉ *set ws*)
(**is** - = (*EX ws*. *?R w ws*))
⟨*proof*⟩

lemma *star-conv-concat*: *star A* = {*concat ws*|*ws*. *set ws* ⊆ *A* & [] ∉ *set ws*}
⟨*proof*⟩

lemma *star-insert-eps*[*simp*]: *star* (*insert* [] *A*) = *star*(*A*)
⟨*proof*⟩

lemma *star-decom*:
  assumes a: $x \in star\ A$ $x \neq []$
  shows $\exists\, a\ b.\ x = a\ @\ b \wedge a \neq [] \wedge a \in A \wedge b \in star\ A$
  ⟨*proof*⟩

lemma *Ball-starI*: $\forall\, a \in set\ as.\ [a] \in A \Longrightarrow as \in star\ A$
  ⟨*proof*⟩

lemma *map-image-star*[*simp*]: $map\ f\ `\ star\ A = star\ (map\ f\ `\ A)$
  ⟨*proof*⟩

1.3 Left-Quotients of Languages

definition *lQuot* :: $'a \Rightarrow 'a\ lang \Rightarrow 'a\ lang$
where $lQuot\ x\ A = \{\ xs.\ x\#xs \in A\ \}$

definition *lQuots* :: $'a\ list \Rightarrow 'a\ lang \Rightarrow 'a\ lang$
where $lQuots\ xs\ A = \{\ ys.\ xs\ @\ ys \in A\ \}$

abbreviation
  lQuotss :: $'a\ list \Rightarrow 'a\ lang\ set \Rightarrow 'a\ lang$
where
  $lQuotss\ s\ As \equiv \bigcup\ (lQuots\ s\ `\ As)$

lemma *lQuot-empty*[*simp*]: $lQuot\ a\ \{\} = \{\}$
  and *lQuot-epsilon*[*simp*]: $lQuot\ a\ \{[]\} = \{\}$
  and *lQuot-char*[*simp*]: $lQuot\ a\ \{[b]\} = (if\ a = b\ then\ \{[]\}\ else\ \{\})$
  and *lQuot-chars*[*simp*]: $lQuot\ a\ \{[b] \mid b.\ P\ b\} = (if\ P\ a\ then\ \{[]\}\ else\ \{\})$
  and *lQuot-union*[*simp*]: $lQuot\ a\ (A \cup B) = lQuot\ a\ A \cup lQuot\ a\ B$
  and *lQuot-inter*[*simp*]: $lQuot\ a\ (A \cap B) = lQuot\ a\ A \cap lQuot\ a\ B$
  and *lQuot-compl*[*simp*]: $lQuot\ a\ (-A) = -\ lQuot\ a\ A$
  ⟨*proof*⟩

lemma *lQuot-conc-subset*: $lQuot\ a\ A\ @@\ B \subseteq lQuot\ a\ (A\ @@\ B)$ (**is** $?L \subseteq ?R$)
⟨*proof*⟩

lemma *lQuot-conc* [*simp*]: $lQuot\ c\ (A\ @@\ B) = (lQuot\ c\ A)\ @@\ B \cup (if\ [] \in A\ then\ lQuot\ c\ B\ else\ \{\})$
  ⟨*proof*⟩

lemma *lQuot-star* [*simp*]: $lQuot\ c\ (star\ A) = (lQuot\ c\ A)\ @@\ star\ A$
⟨*proof*⟩

lemma *lQuot-diff*[*simp*]: $lQuot\ c\ (A - B) = lQuot\ c\ A - lQuot\ c\ B$
  ⟨*proof*⟩

lemma *lQuot-lists*[*simp*]: $c : S \Longrightarrow lQuot\ c\ (lists\ S) = lists\ S$

$\langle proof \rangle$

lemma *lQuots-simps* [*simp*]:
shows $lQuots\ []\ A = A$
and $lQuots\ (c\ \#\ s)\ A = lQuots\ s\ (lQuot\ c\ A)$
and $lQuots\ (s1\ @\ s2)\ A = lQuots\ s2\ (lQuots\ s1\ A)$
$\langle proof \rangle$

lemma *lQuots-append*[*iff*]: $v \in lQuots\ w\ A \longleftrightarrow w\ @\ v \in A$
$\langle proof \rangle$

1.4 Right-Quotients of Languages

definition $rQuot :: 'a \Rightarrow 'a\ lang \Rightarrow 'a\ lang$
where $rQuot\ x\ A = \{\ xs.\ xs\ @\ [x] \in A\ \}$

definition $rQuots :: 'a\ list \Rightarrow 'a\ lang \Rightarrow 'a\ lang$
where $rQuots\ xs\ A = \{\ ys.\ ys\ @\ rev\ xs \in A\ \}$

abbreviation
$rQuotss :: 'a\ list \Rightarrow 'a\ lang\ set \Rightarrow 'a\ lang$
where
$rQuotss\ s\ As \equiv \bigcup\ (rQuots\ s\ `\ As)$

lemma *rQuot-rev-lQuot*: $rQuot\ x\ A = rev\ `\ lQuot\ x\ (rev\ `\ A)$
$\langle proof \rangle$

lemma *rQuots-rev-lQuots*: $rQuots\ x\ A = rev\ `\ lQuots\ x\ (rev\ `\ A)$
$\langle proof \rangle$

lemma *rQuot-empty*[*simp*]: $rQuot\ a\ \{\} = \{\}$
and *rQuot-epsilon*[*simp*]: $rQuot\ a\ \{[]\} = \{\}$
and *rQuot-char*[*simp*]: $rQuot\ a\ \{[b]\} = (if\ a = b\ then\ \{[]\}\ else\ \{\})$
and *rQuot-union*[*simp*]: $rQuot\ a\ (A \cup B) = rQuot\ a\ A \cup rQuot\ a\ B$
and *rQuot-inter*[*simp*]: $rQuot\ a\ (A \cap B) = rQuot\ a\ A \cap rQuot\ a\ B$
and *rQuot-compl*[*simp*]: $rQuot\ a\ (-A) = -\ rQuot\ a\ A$
$\langle proof \rangle$

lemma *lQuot-rQuot*: $lQuot\ a\ (rQuot\ b\ A) = rQuot\ b\ (lQuot\ a\ A)$
$\langle proof \rangle$

lemma *rQuot-lQuot*: $rQuot\ a\ (lQuot\ b\ A) = lQuot\ b\ (rQuot\ a\ A)$
$\langle proof \rangle$

lemma *rev-simp-invert*: $(xs\ @\ [x] = rev\ zs) = (zs = x\ \#\ rev\ xs)$
$\langle proof \rangle$

lemma *rev-append-invert*: $(xs\ @\ ys = rev\ zs) = (zs = rev\ ys\ @\ rev\ xs)$
$\langle proof \rangle$

lemma *image-rev-lists*[*simp*]: rev ‘ $lists\ S = lists\ S$
⟨*proof*⟩

lemma *image-rev-conc*[*simp*]: rev ‘ $(A\ @@\ B) = rev$ ‘ $B\ @@\ rev$ ‘ A
⟨*proof*⟩

lemma *image-rev-star*[*simp*]: rev ‘ $star\ A = star\ (rev$ ‘ $A)$
⟨*proof*⟩

lemma *rQuot-conc* [*simp*]: $rQuot\ c\ (A\ @@\ B) = A\ @@\ (rQuot\ c\ B) \cup (if\ [] \in B\ then\ rQuot\ c\ A\ else\ \{\})$
⟨*proof*⟩

lemma *rQuot-star* [*simp*]: $rQuot\ c\ (star\ A) = star\ A\ @@\ (rQuot\ c\ A)$
⟨*proof*⟩

lemma *rQuot-diff*[*simp*]: $rQuot\ c\ (A - B) = rQuot\ c\ A - rQuot\ c\ B$
⟨*proof*⟩

lemma *rQuot-lists*[*simp*]: $c : S \Longrightarrow rQuot\ c\ (lists\ S) = lists\ S$
⟨*proof*⟩

lemma *rQuots-simps* [*simp*]:
shows $rQuots\ []\ A = A$
and $rQuots\ (c\ \#\ s)\ A = rQuots\ s\ (rQuot\ c\ A)$
and $rQuots\ (s1\ @\ s2)\ A = rQuots\ s2\ (rQuots\ s1\ A)$
⟨*proof*⟩

lemma *rQuots-append*[*iff*]: $v \in rQuots\ w\ A \longleftrightarrow v\ @\ rev\ w \in A$
⟨*proof*⟩

1.5 Two-Sided-Quotients of Languages

definition $biQuot :: 'a \Rightarrow 'a \Rightarrow 'a\ lang \Rightarrow 'a\ lang$
where $biQuot\ x\ y\ A = \{\ xs.\ x\ \#\ xs\ @\ [y] \in A\ \}$

definition $biQuots :: 'a\ list \Rightarrow 'a\ list \Rightarrow 'a\ lang \Rightarrow 'a\ lang$
where $biQuots\ xs\ ys\ A = \{\ zs.\ xs\ @\ zs\ @\ rev\ ys \in A\ \}$

abbreviation
$biQuotss :: 'a\ list \Rightarrow 'a\ list \Rightarrow 'a\ lang\ set \Rightarrow 'a\ lang$
where
$biQuotss\ xs\ ys\ As \equiv \bigcup\ (biQuots\ xs\ ys$ ‘ $As)$

lemma *biQuot-rQuot-lQuot*: $biQuot\ x\ y\ A = rQuot\ y\ (lQuot\ x\ A)$
⟨*proof*⟩

lemma *biQuot-lQuot-rQuot*: $biQuot\ x\ y\ A = lQuot\ x\ (rQuot\ y\ A)$

⟨*proof*⟩

lemma *biQuots-rQuots-lQuots*: $biQuots\ x\ y\ A = rQuots\ y\ (lQuots\ x\ A)$
⟨*proof*⟩

lemma *biQuots-lQuots-rQuots*: $biQuots\ x\ y\ A = lQuots\ x\ (rQuots\ y\ A)$
⟨*proof*⟩

lemma *biQuot-empty*[*simp*]: $biQuot\ a\ b\ \{\} = \{\}$
and *biQuot-epsilon*[*simp*]: $biQuot\ a\ b\ \{[]\} = \{\}$
and *biQuot-char*[*simp*]: $biQuot\ a\ b\ \{[c]\} = \{\}$
and *biQuot-union*[*simp*]: $biQuot\ a\ b\ (A \cup B) = biQuot\ a\ b\ A \cup biQuot\ a\ b\ B$
and *biQuot-inter*[*simp*]: $biQuot\ a\ b\ (A \cap B) = biQuot\ a\ b\ A \cap biQuot\ a\ b\ B$
and *biQuot-compl*[*simp*]: $biQuot\ a\ b\ (-A) = -\ biQuot\ a\ b\ A$
⟨*proof*⟩

lemma *biQuot-conc* [*simp*]: $biQuot\ a\ b\ (A\ @@\ B) =$
$lQuot\ a\ A\ @@\ rQuot\ b\ B \cup$
$(if\ [] \in A \wedge [] \in B\ then\ biQuot\ a\ b\ A \cup biQuot\ a\ b\ B$
$else\ if\ [] \in A\ then\ biQuot\ a\ b\ B$
$else\ if\ [] \in B\ then\ biQuot\ a\ b\ A$
$else\ \{\})$
⟨*proof*⟩

lemma *biQuot-star* [*simp*]: $biQuot\ a\ b\ (star\ A) = biQuot\ a\ b\ A \cup lQuot\ a\ A\ @@\ star\ A\ @@\ rQuot\ b\ A$
⟨*proof*⟩

lemma *biQuot-diff*[*simp*]: $biQuot\ a\ b\ (A - B) = biQuot\ a\ b\ A - biQuot\ a\ b\ B$
⟨*proof*⟩

lemma *biQuot-lists*[*simp*]: $a : S \Longrightarrow b : S \Longrightarrow biQuot\ a\ b\ (lists\ S) = lists\ S$
⟨*proof*⟩

lemma *biQuots-simps* [*simp*]:
shows $biQuots\ []\ []\ A = A$
and $biQuots\ (a\#as)\ (b\#bs)\ A = biQuots\ as\ bs\ (biQuot\ a\ b\ A)$
and $[\![length\ s1 = length\ t1;\ length\ s2 = length\ t2]\!] \Longrightarrow$
$biQuots\ (s1\ @\ s2)\ (t1\ @\ t2)\ A = biQuots\ s2\ t2\ (biQuots\ s1\ t1\ A)$
⟨*proof*⟩

lemma *biQuots-append*[*iff*]: $v \in biQuots\ u\ w\ A \longleftrightarrow u\ @\ v\ @\ rev\ w \in A$
⟨*proof*⟩

1.6 Arden's Lemma

lemma *arden-helper*:
assumes *eq*: $X = A\ @@\ X \cup B$
shows $X = (A\ \widehat{}\widehat{}\ Suc\ n)\ @@\ X \cup (\bigcup m \leq n.\ (A\ \widehat{}\widehat{}\ m)\ @@\ B)$

⟨*proof*⟩

lemma *Arden*:
  assumes $[] \notin A$
  shows $X = A\ @@\ X \cup B \longleftrightarrow X = star\ A\ @@\ B$
⟨*proof*⟩

lemma *reversed-arden-helper*:
  assumes *eq*: $X = X\ @@\ A \cup B$
  shows $X = X\ @@\ (A\ \hat{}\hat{}\ Suc\ n) \cup (\bigcup m \le n.\ B\ @@\ (A\ \hat{}\hat{}\ m))$
⟨*proof*⟩

theorem *reversed-Arden*:
  assumes *nemp*: $[] \notin A$
  shows $X = X\ @@\ A \cup B \longleftrightarrow X = B\ @@\ star\ A$
⟨*proof*⟩

1.7 Lists of Fixed Length

abbreviation *listsN* **where** $listsN\ n\ S \equiv \{xs.\ xs \in lists\ S \wedge length\ xs = n\}$

lemma *tl-listsN*: $A \subseteq listsN\ (n + 1)\ S \Longrightarrow tl\ `\ A \subseteq listsN\ n\ S$
⟨*proof*⟩

lemma *map-tl-listsN*: $A \subseteq lists\ (listsN\ (n + 1)\ S) \Longrightarrow map\ tl\ `\ A \subseteq lists\ (listsN\ n\ S)$
⟨*proof*⟩

2 Π-Extended Regular Expressions

2.1 Syntax of regular expressions

datatype *'a rexp* =
  Zero |
  Full |
  One |
  Atom 'a |
  Plus ('a rexp) ('a rexp) |
  Times ('a rexp) ('a rexp) |
  Star ('a rexp) |
  Not ('a rexp) |
  Inter ('a rexp) ('a rexp) |
  Pr ('a rexp)
derive *linorder rexp*

Lifting constructors to lists

fun *rexp-of-list* **where**
  rexp-of-list OPERATION N [] = *N*

| *rexp-of-list OPERATION N* [*x*] = *x*
| *rexp-of-list OPERATION N* (*x* # *xs*) = *OPERATION x* (*rexp-of-list OPERATION N xs*)

abbreviation *PLUS* ≡ *rexp-of-list Plus Zero*
abbreviation *TIMES* ≡ *rexp-of-list Times One*
abbreviation *INTERSECT* ≡ *rexp-of-list Inter Full*

lemma *list-singleton-induct* [*case-names nil single cons*]:
  assumes *nil*: *P* []
  assumes *single*: $\bigwedge x.\ P\ [x]$
  assumes *cons*: $\bigwedge x\ y\ xs.\ P\ (y\ \#\ xs) \Longrightarrow P\ (x\ \#\ (y\ \#\ xs))$
  shows *P xs*
  ⟨*proof*⟩

2.2 ACI normalization

fun *toplevel-summands* :: *'a rexp* ⇒ *'a rexp set* **where**
  toplevel-summands (*Plus r s*) = *toplevel-summands r* ∪ *toplevel-summands s*
| *toplevel-summands r* = {*r*}

abbreviation (*input*) *flatten LISTOP X* ≡ *LISTOP* (*sorted-list-of-set X*)

lemma *toplevel-summands-nonempty*[*simp*]:
  toplevel-summands r ≠ {}
  ⟨*proof*⟩

lemma *toplevel-summands-finite*[*simp*]:
  finite (*toplevel-summands r*)
  ⟨*proof*⟩

primrec *ACI-norm* :: (*'a*::*linorder*) *rexp* ⇒ *'a rexp* (‹«-»›) **where**
  «*Zero*» = *Zero*
| «*Full*» = *Full*
| «*One*» = *One*
| «*Atom a*» = *Atom a*
| «*Plus r s*» = *flatten PLUS* (*toplevel-summands* (*Plus* «*r*» «*s*»))
| «*Times r s*» = *Times* «*r*» «*s*»
| «*Star r*» = *Star* «*r*»
| «*Not r*» = *Not* «*r*»
| «*Inter r s*» = *Inter* «*r*» «*s*»
| «*Pr r*» = *Pr* «*r*»

lemma *Plus-toplevel-summands*:
  Plus r s ∈ *toplevel-summands t* ⟹ *False*
  ⟨*proof*⟩

lemma *toplevel-summands-not-Plus*[*simp*]:
  (∀ *r s*. *x* ≠ *Plus r s*) ⟹ *toplevel-summands x* = {*x*}

$\langle proof \rangle$

lemma *toplevel-summands-PLUS-strong*:
$[\![xs \neq []; \textit{list-all}\ (\lambda x.\ \neg(\exists\, r\ s.\ x = \textit{Plus}\ r\ s))\ xs]\!] \Longrightarrow \textit{toplevel-summands}\ (\textit{PLUS}\ xs) = \textit{set}\ xs$
$\langle proof \rangle$

lemma *toplevel-summands-flatten*:
$[\![X \neq \{\}; \textit{finite}\ X; \forall\, x \in X.\ \neg(\exists\, r\ s.\ x = \textit{Plus}\ r\ s)]\!] \Longrightarrow \textit{toplevel-summands}\ (\textit{flatten}\ \textit{PLUS}\ X) = X$
$\langle proof \rangle$

lemma *ACI-norm-Plus*:
$«r» = \textit{Plus}\ s\ t \Longrightarrow \exists\, s\ t.\ r = \textit{Plus}\ s\ t$
$\langle proof \rangle$

lemma *toplevel-summands-flatten-ACI-norm-image*:
$\textit{toplevel-summands}\ (\textit{flatten}\ \textit{PLUS}\ (\textit{ACI-norm}\ `\ \textit{toplevel-summands}\ r)) = \textit{ACI-norm}\ `\ \textit{toplevel-summands}\ r$
$\langle proof \rangle$

lemma *toplevel-summands-flatten-ACI-norm-image-Union*:
$\textit{toplevel-summands}\ (\textit{flatten}\ \textit{PLUS}\ (\textit{ACI-norm}\ `\ \textit{toplevel-summands}\ r \cup \textit{ACI-norm}\ `\ \textit{toplevel-summands}\ s)) =$
$\textit{ACI-norm}\ `\ \textit{toplevel-summands}\ r \cup \textit{ACI-norm}\ `\ \textit{toplevel-summands}\ s$
$\langle proof \rangle$

lemma *toplevel-summands-ACI-norm*:
$\textit{toplevel-summands}\ «r» = \textit{ACI-norm}\ `\ \textit{toplevel-summands}\ r$
$\langle proof \rangle$

lemma *ACI-norm-flatten*:
$«r» = \textit{flatten}\ \textit{PLUS}\ (\textit{ACI-norm}\ `\ \textit{toplevel-summands}\ r)$
$\langle proof \rangle$

theorem *ACI-norm-idem*[*simp*]:
$««r»» = «r»$
$\langle proof \rangle$

fun *ACI-nPlus* :: $'a{::}\textit{linorder}\ \textit{rexp} \Rightarrow\ 'a\ \textit{rexp} \Rightarrow\ 'a\ \textit{rexp}$
where
$\textit{ACI-nPlus}\ (\textit{Plus}\ r1\ r2)\ s = \textit{ACI-nPlus}\ r1\ (\textit{ACI-nPlus}\ r2\ s)$
$|\ \textit{ACI-nPlus}\ r\ (\textit{Plus}\ s1\ s2) =$
$(\textit{if}\ r = s1\ \textit{then}\ \textit{Plus}\ s1\ s2$
$\textit{else if}\ r < s1\ \textit{then}\ \textit{Plus}\ r\ (\textit{Plus}\ s1\ s2)$
$\textit{else}\ \textit{Plus}\ s1\ (\textit{ACI-nPlus}\ r\ s2))$
$|\ \textit{ACI-nPlus}\ r\ s =$
$(\textit{if}\ r = s\ \textit{then}\ r$
$\textit{else if}\ r < s\ \textit{then}\ \textit{Plus}\ r\ s$

else Plus s r)

fun *ACI-norm-alt* **where**
ACI-norm-alt Zero = *Zero*
| *ACI-norm-alt Full* = *Full*
| *ACI-norm-alt One* = *One*
| *ACI-norm-alt* (*Atom a*) = *Atom a*
| *ACI-norm-alt* (*Plus r s*) = *ACI-nPlus* (*ACI-norm-alt r*) (*ACI-norm-alt s*)
| *ACI-norm-alt* (*Times r s*) = *Times* (*ACI-norm-alt r*) (*ACI-norm-alt s*)
| *ACI-norm-alt* (*Star r*) = *Star* (*ACI-norm-alt r*)
| *ACI-norm-alt* (*Not r*) = *Not* (*ACI-norm-alt r*)
| *ACI-norm-alt* (*Inter r s*) = *Inter* (*ACI-norm-alt r*) (*ACI-norm-alt s*)
| *ACI-norm-alt* (*Pr r*) = *Pr* (*ACI-norm-alt r*)

lemma *toplevel-summands-ACI-nPlus*:
toplevel-summands (*ACI-nPlus r s*) = *toplevel-summands* (*Plus r s*)
⟨*proof*⟩

lemma *toplevel-summands-ACI-norm-alt*:
toplevel-summands (*ACI-norm-alt r*) = *ACI-norm-alt* ‘ *toplevel-summands r*
⟨*proof*⟩

lemma *ACI-norm-alt-Plus*:
ACI-norm-alt r = *Plus s t* $\Longrightarrow$ $\exists$ *s t*. *r* = *Plus s t*
⟨*proof*⟩

lemma *toplevel-summands-flatten-ACI-norm-alt-image*:
toplevel-summands (*flatten PLUS* (*ACI-norm-alt* ‘ *toplevel-summands r*)) = *ACI-norm-alt* ‘ *toplevel-summands r*
⟨*proof*⟩

lemma *ACI-norm-ACI-norm-alt*: «*ACI-norm-alt r*» = «*r*»
⟨*proof*⟩

lemma *ACI-nPlus-singleton-PLUS*:
$[\![$*xs* $\neq$ []; *sorted xs*; *distinct xs*; $\forall$ *x* $\in$ {*x*} $\cup$ *set xs*. $\neg$($\exists$ *r s*. *x* = *Plus r s*)$]\!]$ $\Longrightarrow$
ACI-nPlus x (*PLUS xs*) = (*if x* $\in$ *set xs then PLUS xs else PLUS* (*insort x xs*))
⟨*proof*⟩

lemma *ACI-nPlus-PLUS*:
$[\![$*xs1* $\neq$ []; *xs2* $\neq$ []; $\forall$ *x* $\in$ *set* (*xs1* @ *xs2*). $\neg$($\exists$ *r s*. *x* = *Plus r s*); *sorted xs2*; *distinct xs2*$]\!]$ $\Longrightarrow$
ACI-nPlus (*PLUS xs1*) (*PLUS xs2*) = *flatten PLUS* (*set* (*xs1* @ *xs2*))
⟨*proof*⟩

lemma *ACI-nPlus-flatten-PLUS*:
$[\![$*X1* $\neq$ {}; *X2* $\neq$ {}; *finite X1*; *finite X2*; $\forall$ *x* $\in$ *X1* $\cup$ *X2*. $\neg$($\exists$ *r s*. *x* = *Plus r s*)$]\!]$ $\Longrightarrow$
ACI-nPlus (*flatten PLUS X1*) (*flatten PLUS X2*) = *flatten PLUS* (*X1* $\cup$ *X2*)

⟨*proof*⟩

lemma *ACI-nPlus-ACI-norm*[*simp*]: *ACI-nPlus* «*r*» «*s*» = «*Plus r s*»
⟨*proof*⟩

lemma *ACI-norm-alt*:
ACI-norm-alt r = «*r*»
⟨*proof*⟩

declare *ACI-norm-alt*[*symmetric*, *code*]

2.3 Finality

primrec *final* :: *'a rexp* ⇒ *bool*
where
final Zero = *False*
| *final Full* = *True*
| *final One* = *True*
| *final* (*Atom* -) = *False*
| *final* (*Plus r s*) = (*final r* ∨ *final s*)
| *final* (*Times r s*) = (*final r* ∧ *final s*)
| *final* (*Star* -) = *True*
| *final* (*Not r*) = (~ *final r*)
| *final* (*Inter r1 r2*) = (*final r1* ∧ *final r2*)
| *final* (*Pr r*) = *final r*

lemma *toplevel-summands-final*:
final s = (∃ *r*∈*toplevel-summands s*. *final r*)
⟨*proof*⟩

lemma *final-PLUS*:
final (*PLUS xs*) = (∃ *r* ∈ *set xs*. *final r*)
⟨*proof*⟩

theorem *ACI-norm-final*[*simp*]:
final «*r*» = *final r*
⟨*proof*⟩

2.4 Wellformedness w.r.t. an alphabet

locale *alphabet* =
fixes Σ :: *nat* ⇒ *'a set* (‹Σ -›)
and *wf-atom* :: *nat* ⇒ *'b* :: *linorder* ⇒ *bool*
begin

primrec *wf* :: *nat* ⇒ *'b rexp* ⇒ *bool*
where
wf n Zero = *True* |
wf n Full = *True* |
wf n One = *True* |

wf n (*Atom a*) = (*wf-atom n a*) |
wf n (*Plus r s*) = (*wf n r* ∧ *wf n s*) |
wf n (*Times r s*) = (*wf n r* ∧ *wf n s*) |
wf n (*Star r*) = *wf n r* |
wf n (*Not r*) = *wf n r* |
wf n (*Inter r s*) = (*wf n r* ∧ *wf n s*) |
wf n (*Pr r*) = *wf* (*n* + *1*) *r*

primrec *wf-word* **where**
wf-word n [] = *True*
| *wf-word n* (*w* # *ws*) = ((*w* ∈ Σ *n*) ∧ *wf-word n ws*)

lemma *wf-word-snoc*[*simp*]: *wf-word n* (*ws* @ [*w*]) = ((*w* ∈ Σ *n*) ∧ *wf-word n ws*)
⟨*proof*⟩

lemma *wf-word-append*[*simp*]: *wf-word n* (*ws* @ *vs*) = (*wf-word n ws* ∧ *wf-word n vs*)
⟨*proof*⟩

lemma *wf-word*: *wf-word n w* = (*w* ∈ *lists* (Σ *n*))
⟨*proof*⟩

lemma *toplevel-summands-wf*:
wf n s = (∀ *r*∈*toplevel-summands s*. *wf n r*)
⟨*proof*⟩

lemma *wf-PLUS*[*simp*]:
wf n (*PLUS xs*) = (∀ *r* ∈ *set xs*. *wf n r*)
⟨*proof*⟩

lemma *wf-TIMES*[*simp*]:
wf n (*TIMES xs*) = (∀ *r* ∈ *set xs*. *wf n r*)
⟨*proof*⟩

lemma *wf-flatten-PLUS*[*simp*]:
finite X ⟹ *wf n* (*flatten PLUS X*) = (∀ *r* ∈ *X*. *wf n r*)
⟨*proof*⟩

theorem *ACI-norm-wf*[*simp*]:
wf n «*r*» = *wf n r*
⟨*proof*⟩

lemma *wf-INTERSECT*[*simp*]:
wf n (*INTERSECT xs*) = (∀ *r* ∈ *set xs*. *wf n r*)
⟨*proof*⟩

lemma *wf-flatten-INTERSECT*[*simp*]:
finite X ⟹ *wf n* (*flatten INTERSECT X*) = (∀ *r* ∈ *X*. *wf n r*)
⟨*proof*⟩

end

2.5 Language

locale *project* =
  alphabet Σ *wf-atom* **for** Σ :: *nat* $\Rightarrow$ *'a set* **and** *wf-atom* :: *nat* $\Rightarrow$ *'b* :: *linorder* $\Rightarrow$ *bool* +
fixes *project* :: *'a* $\Rightarrow$ *'a*
and *lookup* :: *'b* $\Rightarrow$ *'a* $\Rightarrow$ *bool*
assumes *project*: $\bigwedge a.\ a \in \Sigma\ (Suc\ n) \Longrightarrow project\ a \in \Sigma\ n$
begin

primrec *lang* :: *nat* $\Rightarrow$ *'b rexp* => *'a lang* **where**
$lang\ n\ Zero = \{\}$ |
$lang\ n\ Full = lists\ (\Sigma\ n)$ |
$lang\ n\ One = \{[]\}$ |
$lang\ n\ (Atom\ b) = \{[x] \mid x.\ lookup\ b\ x \wedge x \in \Sigma\ n\}$ |
$lang\ n\ (Plus\ r\ s) = (lang\ n\ r) \cup (lang\ n\ s)$ |
$lang\ n\ (Times\ r\ s) = conc\ (lang\ n\ r)\ (lang\ n\ s)$ |
$lang\ n\ (Star\ r) = star\ (lang\ n\ r)$ |
$lang\ n\ (Not\ r) = lists\ (\Sigma\ n) - lang\ n\ r$ |
$lang\ n\ (Inter\ r\ s) = (lang\ n\ r \cap lang\ n\ s)$ |
$lang\ n\ (Pr\ r) = map\ project\ `\ lang\ (n + 1)\ r$

lemma *wf-word-map-project*[*simp*]: *wf-word* (*Suc n*) *ws* $\Longrightarrow$ *wf-word n* (*map project ws*)
  ⟨*proof*⟩

lemma *wf-lang-wf-word*: $wf\ n\ r \Longrightarrow \forall w \in lang\ n\ r.\ wf\text{-}word\ n\ w$
  ⟨*proof*⟩

lemma *lang-subset-lists*: $wf\ n\ r \Longrightarrow lang\ n\ r \subseteq lists\ (\Sigma\ n)$
⟨*proof*⟩

lemma *toplevel-summands-lang*:
  $r \in toplevel\text{-}summands\ s \Longrightarrow lang\ n\ r \subseteq lang\ n\ s$
  ⟨*proof*⟩

lemma *toplevel-summands-lang-UN*:
  $lang\ n\ s = (\bigcup r{\in}toplevel\text{-}summands\ s.\ lang\ n\ r)$
  ⟨*proof*⟩

lemma *toplevel-summands-in-lang*:
  $w \in lang\ n\ s = (\exists r{\in}toplevel\text{-}summands\ s.\ w \in lang\ n\ r)$
  ⟨*proof*⟩

lemma *lang-PLUS*[*simp*]:
  $lang\ n\ (PLUS\ xs) = (\bigcup r \in set\ xs.\ lang\ n\ r)$

$\langle proof \rangle$

lemma *lang-TIMES*[*simp*]:
$lang\ n\ (TIMES\ xs) = foldr\ (@@)\ (map\ (lang\ n)\ xs)\ \{[]\}$
$\langle proof \rangle$

lemma *lang-flatten-PLUS*:
$finite\ X \Longrightarrow lang\ n\ (flatten\ PLUS\ X) = (\bigcup r \in X.\ lang\ n\ r)$
$\langle proof \rangle$

theorem *ACI-norm-lang*[*simp*]:
$lang\ n\ «r» = lang\ n\ r$
$\langle proof \rangle$

lemma *lang-final*: $final\ r = ([] \in lang\ n\ r)$
$\langle proof \rangle$

lemma *in-lang-INTERSECT*:
$wf\text{-}word\ n\ w \Longrightarrow w \in lang\ n\ (INTERSECT\ xs) = (\forall r \in set\ xs.\ w \in lang\ n\ r)$
$\langle proof \rangle$

lemma *lang-INTERSECT*:
$lang\ n\ (INTERSECT\ xs) = (if\ xs = []\ then\ lists\ (\Sigma\ n)\ else\ \bigcap r \in set\ xs.\ lang\ n\ r)$
$\langle proof \rangle$

lemma *lang-flatten-INTERSECT*[*simp*]:
assumes $finite\ X\ X \neq \{\}\ \forall r{\in}X.\ wf\ n\ r$
shows $w \in lang\ n\ (flatten\ INTERSECT\ X) = (\forall r \in X.\ w \in lang\ n\ r)$ (**is** $?L = ?R$)
$\langle proof \rangle$

end

3 Derivatives of Π-Extended Regular Expressions

locale *embed* = *project* Σ *wf-atom project lookup*
for $\Sigma :: nat \Rightarrow 'a\ set$
and $wf\text{-}atom :: nat \Rightarrow 'b :: linorder \Rightarrow bool$
and $project :: 'a \Rightarrow 'a$
and $lookup :: 'b \Rightarrow 'a \Rightarrow bool$ +
fixes $embed :: 'a \Rightarrow 'a\ list$
assumes *embed*: $\bigwedge a.\ a \in \Sigma\ n \Longrightarrow b \in set\ (embed\ a) = (b \in \Sigma\ (Suc\ n) \wedge project\ b = a)$
begin

3.1 Syntactic Derivatives

primrec $lderiv :: 'a \Rightarrow 'b\ rexp \Rightarrow 'b\ rexp$ **where**
$lderiv\ \text{-}\ Zero = Zero$
$|\ lderiv\ \text{-}\ Full = Full$
$|\ lderiv\ \text{-}\ One = Zero$
$|\ lderiv\ a\ (Atom\ b) = (if\ lookup\ b\ a\ then\ One\ else\ Zero)$
$|\ lderiv\ a\ (Plus\ r\ s) = Plus\ (lderiv\ a\ r)\ (lderiv\ a\ s)$
$|\ lderiv\ a\ (Times\ r\ s) =$
$(let\ r's = Times\ (lderiv\ a\ r)\ s$
$in\ if\ final\ r\ then\ Plus\ r's\ (lderiv\ a\ s)\ else\ r's)$
$|\ lderiv\ a\ (Star\ r) = Times\ (lderiv\ a\ r)\ (Star\ r)$
$|\ lderiv\ a\ (Not\ r) = Not\ (lderiv\ a\ r)$
$|\ lderiv\ a\ (Inter\ r\ s) = Inter\ (lderiv\ a\ r)\ (lderiv\ a\ s)$
$|\ lderiv\ a\ (Pr\ r) = Pr\ (PLUS\ (map\ (\lambda a'.\ lderiv\ a'\ r)\ (embed\ a)))$

primrec *lderivs* **where**
$lderivs\ []\ r = r$
$|\ lderivs\ (w\#ws)\ r = lderivs\ ws\ (lderiv\ w\ r)$

3.2 Finiteness of ACI-Equivalent Derivatives

lemma *toplevel-summands-lderiv*:
$toplevel\text{-}summands\ (lderiv\ as\ r) = (\bigcup s{\in}toplevel\text{-}summands\ r.\ toplevel\text{-}summands\ (lderiv\ as\ s))$
⟨*proof*⟩

lemma *lderivs-Zero*[*simp*]: $lderivs\ xs\ Zero = Zero$
⟨*proof*⟩

lemma *lderivs-Full*[*simp*]: $lderivs\ xs\ Full = Full$
⟨*proof*⟩

lemma *lderivs-One*: $lderivs\ xs\ One \in \{Zero,\ One\}$
⟨*proof*⟩

lemma *lderivs-Atom*: $lderivs\ xs\ (Atom\ as) \in \{Zero,\ One,\ Atom\ as\}$
⟨*proof*⟩

lemma *lderivs-Plus*: $lderivs\ xs\ (Plus\ r\ s) = Plus\ (lderivs\ xs\ r)\ (lderivs\ xs\ s)$
⟨*proof*⟩

lemma *lderivs-PLUS*: $lderivs\ xs\ (PLUS\ ys) = PLUS\ (map\ (lderivs\ xs)\ ys)$
⟨*proof*⟩

lemma *toplevel-summands-lderivs-Times*: $toplevel\text{-}summands\ (lderivs\ xs\ (Times\ r\ s)) \subseteq$
$\{Times\ (lderivs\ xs\ r)\ s\} \cup$
$\{r'.\ \exists ys\ zs.\ r' \in toplevel\text{-}summands\ (lderivs\ ys\ s) \wedge ys \neq [] \wedge zs\ @\ ys = xs\}$
⟨*proof*⟩

lemma *toplevel-summands-lderivs-Star-nonempty*:
$xs \neq [] \Longrightarrow$ *toplevel-summands* (*lderivs xs* (*Star r*)) $\subseteq$
{*Times* (*lderivs ys r*) (*Star r*) | *ys*. $\exists$ *zs*. *ys* $\neq []$ $\wedge$ *zs* @ *ys* = *xs*}
⟨*proof*⟩

lemma *toplevel-summands-lderivs-Star*:
toplevel-summands (*lderivs xs* (*Star r*)) $\subseteq$
{*Star r*} $\cup$ {*Times* (*lderivs ys r*) (*Star r*) | *ys*. $\exists$ *zs*. *ys* $\neq []$ $\wedge$ *zs* @ *ys* = *xs*}
⟨*proof*⟩

lemma *ex-lderivs-Pr*: $\exists$ *s*. *lderivs ass* (*Pr r*) = *Pr s*
⟨*proof*⟩

lemma *toplevel-summands-PLUS*:
$xs \neq [] \Longrightarrow$ *toplevel-summands* (*PLUS* (*map f xs*)) = ($\bigcup$ *r* $\in$ *set xs*. *toplevel-summands* (*f r*))
⟨*proof*⟩

lemma *lderiv-toplevel-summands-Zero*:
⟦*lderivs xs* (*Pr r*) = *Pr s*; *toplevel-summands r* = {*Zero*}⟧ $\Longrightarrow$ *toplevel-summands s* = {*Zero*}
⟨*proof*⟩

lemma *toplevel-summands-lderivs-Pr*:
⟦*xs* $\neq []$; *lderivs xs* (*Pr r*) = *Pr s*⟧ $\Longrightarrow$
toplevel-summands s $\subseteq$ {*Zero*} $\vee$ *toplevel-summands s* $\subseteq$ ($\bigcup$ *xs*. *toplevel-summands* (*lderivs xs r*))
⟨*proof*⟩

lemma *lderivs-Pr*:
{*lderivs xs* (*Pr r*) | *xs*. *True*} $\subseteq$
{*Pr s* | *s*. *toplevel-summands s* $\subseteq$ {*Zero*} $\vee$
toplevel-summands s $\subseteq$ ($\bigcup$ *xs*. *toplevel-summands* (*lderivs xs r*))}
(**is** *?L* $\subseteq$ *?R*)
⟨*proof*⟩

lemma *ACI-norm-toplevel-summands-Zero*: *toplevel-summands r* $\subseteq$ {*Zero*} $\Longrightarrow$ «*r*» = *Zero*
⟨*proof*⟩

lemma *ACI-norm-lderivs-Pr*:
ACI-norm ‘ {*lderivs xs* (*Pr r*) | *xs*. *True*} $\subseteq$
{*Pr Zero*} $\cup$ {*Pr* «*s*» | *s*. *toplevel-summands s* $\subseteq$ ($\bigcup$ *xs*. *toplevel-summands* «*lderivs xs r*»)}
⟨*proof*⟩

lemma *finite-ACI-norm-toplevel-summands*: *finite B* $\Longrightarrow$ *finite* {*f* «*s*» |*s*. *toplevel-summands s* $\subseteq$ *B*}

$\langle proof \rangle$

lemma *lderivs-Not*: *lderivs xs* (*Not r*) = *Not* (*lderivs xs r*)
$\langle proof \rangle$

lemma *lderivs-Inter*: *lderivs xs* (*Inter r s*) = *Inter* (*lderivs xs r*) (*lderivs xs s*)
$\langle proof \rangle$

theorem *finite-lderivs*: *finite* {«*lderivs xs r*» | *xs* . *True*}
$\langle proof \rangle$

3.3 Wellformedness and language of derivatives

lemma *wf-lderiv*[*simp*]: *wf n r* $\Longrightarrow$ *wf n* (*lderiv w r*)
$\langle proof \rangle$

lemma *wf-lderivs*[*simp*]: *wf n r* $\Longrightarrow$ *wf n* (*lderivs ws r*)
$\langle proof \rangle$

lemma *lQuot-map-project*:
assumes *as* $\in \Sigma$ *n A* $\subseteq$ *lists* (Σ (*Suc n*))
shows *lQuot as* (*map project* ‘ *A*) = *map project* ‘ ($\bigcup$ *a* $\in$ *set* (*embed as*). *lQuot a A*) (**is** *?L* = *?R*)
$\langle proof \rangle$

lemma *lang-lderiv*: $[\![$*wf n r*; *w* $\in \Sigma$ *n*$]\!]$ $\Longrightarrow$ *lang n* (*lderiv w r*) = *lQuot w* (*lang n r*)
$\langle proof \rangle$

lemma *lang-lderivs*: $[\![$*wf n r*; *wf-word n ws*$]\!]$ $\Longrightarrow$ *lang n* (*lderivs ws r*) = *lQuots ws* (*lang n r*)
$\langle proof \rangle$

corollary *lderivs-final*:
assumes *wf n r wf-word n ws*
shows *final* (*lderivs ws r*) $\longleftrightarrow$ *ws* $\in$ *lang n r*
$\langle proof \rangle$

abbreviation *lderivs-set n r s* $\equiv$ {(«*lderivs w r*», «*lderivs w s*») | *w*. *wf-word n w*}

3.4 Deriving preserves ACI-equivalence

lemma *ACI-norm-PLUS*:
list-all2 (λ*r s*. «*r*» = «*s*») *xs ys* $\Longrightarrow$ «*PLUS xs*» = «*PLUS ys*»
$\langle proof \rangle$

lemma *toplevel-summands-ACI-norm-lderiv*:
($\bigcup$ *a*$\in$*toplevel-summands r*. *toplevel-summands* «*lderiv as* «*a*»») = *toplevel-summands* «*lderiv as* «*r*»»

$\langle proof \rangle$

theorem *ACI-norm-lderiv*:
«*lderiv as* «*r*»» = «*lderiv as r*»
$\langle proof \rangle$

corollary *lderiv-preserves*: «*r*» = «*s*» $\Longrightarrow$ «*lderiv as r*» = «*lderiv as s*»
$\langle proof \rangle$

lemma *lderivs-snoc*[*simp*]: *lderivs* (*ws* @ [*w*]) *r* = (*lderiv w* (*lderivs ws r*))
$\langle proof \rangle$

theorem *ACI-norm-lderivs*:
«*lderivs ws* «*r*»» = «*lderivs ws r*»
$\langle proof \rangle$

lemma *lderivs-alt*: «*lderivs w r*» = *fold* (λ*a r*. «*lderiv a r*») *w* «*r*»
$\langle proof \rangle$

lemma *finite-fold-lderiv*: *finite* {*fold* (λ*a r*. «*lderiv a r*») *w* «*s*» |*w*. *True*}
$\langle proof \rangle$

end

4 Some Useful Regular Operators

primrec *REV* :: *'a rexp* $\Rightarrow$ *'a rexp* **where**
REV Zero = *Zero*
| *REV Full* = *Full*
| *REV One* = *One*
| *REV* (*Atom a*) = *Atom a*
| *REV* (*Plus r s*) = *Plus* (*REV r*) (*REV s*)
| *REV* (*Times r s*) = *Times* (*REV s*) (*REV r*)
| *REV* (*Star r*) = *Star* (*REV r*)
| *REV* (*Not r*) = *Not* (*REV r*)
| *REV* (*Inter r s*) = *Inter* (*REV r*) (*REV s*)
| *REV* (*Pr r*) = *Pr* (*REV r*)

lemma *REV-REV*[*simp*]: *REV* (*REV r*) = *r*
$\langle proof \rangle$

lemma *final-REV*[*simp*]: *final* (*REV r*) = *final r*
$\langle proof \rangle$

lemma *REV-PLUS*: *REV* (*PLUS xs*) = *PLUS* (*map REV xs*)
$\langle proof \rangle$

lemma (**in** *alphabet*) *wf-REV*[*simp*]: $wf\ n\ r \Longrightarrow wf\ n\ (REV\ r)$
⟨*proof*⟩

lemma (**in** *project*) *lang-REV*[*simp*]: $lang\ n\ (REV\ r) = rev\ `\ lang\ n\ r$
⟨*proof*⟩

context *embed*
begin

primrec *rderiv* :: $'a \Rightarrow 'b\ rexp \Rightarrow 'b\ rexp$ **where**
$rderiv\ \text{-}\ Zero = Zero$
$|\ rderiv\ \text{-}\ Full = Full$
$|\ rderiv\ \text{-}\ One = Zero$
$|\ rderiv\ a\ (Atom\ b) = (if\ lookup\ b\ a\ then\ One\ else\ Zero)$
$|\ rderiv\ a\ (Plus\ r\ s) = Plus\ (rderiv\ a\ r)\ (rderiv\ a\ s)$
$|\ rderiv\ a\ (Times\ r\ s) =$
$(let\ rs' = Times\ r\ (rderiv\ a\ s)$
$in\ if\ final\ s\ then\ Plus\ rs'\ (rderiv\ a\ r)\ else\ rs')$
$|\ rderiv\ a\ (Star\ r) = Times\ (Star\ r)\ (rderiv\ a\ r)$
$|\ rderiv\ a\ (Not\ r) = Not\ (rderiv\ a\ r)$
$|\ rderiv\ a\ (Inter\ r\ s) = Inter\ (rderiv\ a\ r)\ (rderiv\ a\ s)$
$|\ rderiv\ a\ (Pr\ r) = Pr\ (PLUS\ (map\ (\lambda a'.\ rderiv\ a'\ r)\ (embed\ a)))$

primrec *rderivs* **where**
$rderivs\ []\ r = r$
$|\ rderivs\ (w\#ws)\ r = rderivs\ ws\ (rderiv\ w\ r)$

lemma *rderivs-snoc*: $rderivs\ (ws\ @\ [w])\ r = rderiv\ w\ (rderivs\ ws\ r)$
⟨*proof*⟩

lemma *rderivs-append*: $rderivs\ (ws\ @\ ws')\ r = rderivs\ ws'\ (rderivs\ ws\ r)$
⟨*proof*⟩

lemma *rderiv-lderiv*: $rderiv\ as\ r = REV\ (lderiv\ as\ (REV\ r))$
⟨*proof*⟩

lemma *rderivs-lderivs*: $rderivs\ w\ r = REV\ (lderivs\ w\ (REV\ r))$
⟨*proof*⟩

lemma *wf-rderiv*[*simp*]: $wf\ n\ r \Longrightarrow wf\ n\ (rderiv\ w\ r)$
⟨*proof*⟩

lemma *wf-rderivs*[*simp*]: $wf\ n\ r \Longrightarrow wf\ n\ (rderivs\ ws\ r)$
⟨*proof*⟩

lemma *lang-rderiv*: $[\![wf\ n\ r;\ as \in \Sigma\ n]\!] \Longrightarrow lang\ n\ (rderiv\ as\ r) = rQuot\ as\ (lang\ n\ r)$
⟨*proof*⟩

lemma *lang-rderivs*: $[\![wf\ n\ r;\ wf\text{-}word\ n\ w]\!] \Longrightarrow lang\ n\ (rderivs\ w\ r) = rQuots\ w\ (lang\ n\ r)$
⟨*proof*⟩

corollary *rderivs-final*:
assumes *wf n r wf-word n w*
shows $final\ (rderivs\ w\ r) \longleftrightarrow rev\ w \in lang\ n\ r$
⟨*proof*⟩

lemma *toplevel-summands-REV*[*simp*]: $toplevel\text{-}summands\ (REV\ r) = REV\ `\ toplevel\text{-}summands\ r$
⟨*proof*⟩

lemma *ACI-norm-REV*: «*REV* «*r*»» = «*REV r*»
⟨*proof*⟩

lemma *ACI-norm-rderiv*: «*rderiv as* «*r*»» = «*rderiv as r*»
⟨*proof*⟩

lemma *ACI-norm-rderivs*: «*rderivs w* «*r*»» = «*rderivs w r*»
⟨*proof*⟩

theorem *finite-rderivs*: *finite* {«*rderivs xs r*» | *xs* . *True*}
⟨*proof*⟩

lemma *lderiv-PLUS*[*simp*]: $lderiv\ a\ (PLUS\ xs) = PLUS\ (map\ (lderiv\ a)\ xs)$
⟨*proof*⟩

lemma *rderiv-PLUS*[*simp*]: $rderiv\ a\ (PLUS\ xs) = PLUS\ (map\ (rderiv\ a)\ xs)$
⟨*proof*⟩

lemma *lang-rderiv-lderiv*: $lang\ n\ (rderiv\ a\ (lderiv\ b\ r)) = lang\ n\ (lderiv\ b\ (rderiv\ a\ r))$
⟨*proof*⟩

lemma *lang-lderiv-rderiv*: $lang\ n\ (lderiv\ a\ (rderiv\ b\ r)) = lang\ n\ (rderiv\ b\ (lderiv\ a\ r))$
⟨*proof*⟩

lemma *lang-rderiv-lderivs*[*simp*]: $[\![wf\ n\ r;\ wf\text{-}word\ n\ w;\ a \in \Sigma\ n]\!] \Longrightarrow$
$lang\ n\ (rderiv\ a\ (lderivs\ w\ r)) = lang\ n\ (lderivs\ w\ (rderiv\ a\ r))$
⟨*proof*⟩

lemma *lang-lderiv-rderivs*[*simp*]: $[\![wf\ n\ r;\ wf\text{-}word\ n\ w;\ a \in \Sigma\ n]\!] \Longrightarrow$
$lang\ n\ (lderiv\ a\ (rderivs\ w\ r)) = lang\ n\ (rderivs\ w\ (lderiv\ a\ r))$
⟨*proof*⟩

definition $biderivs\ w1\ w2 = rderivs\ w2\ o\ lderivs\ w1$

lemma *lang-biderivs*: $\llbracket wf\ n\ r;\ wf\text{-}word\ n\ w1;\ wf\text{-}word\ n\ w2\rrbracket \Longrightarrow$
$lang\ n\ (biderivs\ w1\ w2\ r) = biQuots\ w1\ w2\ (lang\ n\ r)$
⟨*proof*⟩

lemma *wf-biderivs*[*simp*]: $wf\ n\ r \Longrightarrow wf\ n\ (biderivs\ w1\ w2\ r)$
⟨*proof*⟩

corollary *biderivs-final*:
assumes $wf\ n\ r\ wf\text{-}word\ n\ w1\ wf\text{-}word\ n\ w2$
shows $final\ (biderivs\ w1\ w2\ r) \longleftrightarrow w1\ @\ rev\ w2 \in lang\ n\ r$
⟨*proof*⟩

lemma *ACI-norm-biderivs*: «*biderivs w1 w2* «*r*»» = «*biderivs w1 w2 r*»
⟨*proof*⟩

lemma *finite* {«*biderivs w1 w2 r*» | *w1 w2* . *True*}
⟨*proof*⟩

end

4.1 Quotioning by the same letter

definition $fin\text{-}cut\text{-}same\ x\ xs = take\ (LEAST\ n.\ drop\ n\ xs = replicate\ (length\ xs - n)\ x)\ xs$

lemma *fin-cut-same-Nil*[*simp*]: $fin\text{-}cut\text{-}same\ x\ [] = []$
⟨*proof*⟩

lemma *Least-fin-cut-same*: $(LEAST\ n.\ drop\ n\ xs = replicate\ (length\ xs - n)\ y) =$
$length\ xs - length\ (takeWhile\ (\lambda x.\ x = y)\ (rev\ xs))$
(**is** $Least\ ?P = ?min$)
⟨*proof*⟩

lemma *takeWhile-takes-all*: $length\ xs = m \Longrightarrow m \le length\ (takeWhile\ P\ xs) \longleftrightarrow Ball\ (set\ xs)\ P$
⟨*proof*⟩

lemma *fin-cut-same-Cons*[*simp*]: $fin\text{-}cut\text{-}same\ x\ (y\ \#\ xs) =$
$(if\ fin\text{-}cut\text{-}same\ x\ xs = []\ then\ if\ x = y\ then\ []\ else\ [y]\ else\ y\ \#\ fin\text{-}cut\text{-}same\ x\ xs)$
⟨*proof*⟩

lemma *fin-cut-same-singleton*[*simp*]: $fin\text{-}cut\text{-}same\ x\ (xs\ @\ [x]) = fin\text{-}cut\text{-}same\ x\ xs$
⟨*proof*⟩

lemma *fin-cut-same-replicate*[*simp*]: $fin\text{-}cut\text{-}same\ x\ (xs\ @\ replicate\ n\ x) = fin\text{-}cut\text{-}same\ x\ xs$
⟨*proof*⟩

lemma *fin-cut-sameE*: *fin-cut-same x xs* = *ys* $\Longrightarrow \exists\, m.$ *xs* = *ys* @ *replicate m x*
⟨*proof*⟩

definition *SAMEQUOT a A* = {*fin-cut-same a x* @ *replicate m a*| *x m*. *x* ∈ *A*}

lemma *SAMEQUOT-mono*: *A* ⊆ *B* $\Longrightarrow$ *SAMEQUOT a A* ⊆ *SAMEQUOT a B*
⟨*proof*⟩

locale *embed2* = *embed* Σ *wf-atom project lookup embed*
for Σ :: *nat* ⇒ *'a set*
and *wf-atom* :: *nat* ⇒ *'b* :: *linorder* ⇒ *bool*
and *project* :: *'a* ⇒ *'a*
and *lookup* :: *'b* ⇒ *'a* ⇒ *bool*
and *embed* :: *'a* ⇒ *'a list* +
fixes *singleton* :: *'a* ⇒ *'b*
assumes *wf-singleton*[*simp*]: *a* ∈ Σ *n* $\Longrightarrow$ *wf-atom n* (*singleton a*)
assumes *lookup-singleton*[*simp*]: *lookup* (*singleton a*) *a'* = (*a* = *a'*)
begin

lemma *finite-rderivs-same*: *finite* {«*rderivs* (*replicate m a*) *r*» | *m* . *True*}
⟨*proof*⟩

lemma *wf-word-replicate*[*simp*]: *a* ∈ Σ *n* $\Longrightarrow$ *wf-word n* (*replicate m a*)
⟨*proof*⟩

lemma *star-singleton*[*simp*]: *star* {[*x*]} = {*replicate m x* | *m* . *True*}
⟨*proof*⟩

definition *samequot a r* = *Times* (*flatten PLUS* {«*rderivs* (*replicate m a*) *r*» | *m* . *True*}) (*Star* (*Atom* (*singleton a*)))

lemma *wf-samequot*: ⟦*wf n r*; *a* ∈ Σ *n*⟧ $\Longrightarrow$ *wf n* (*samequot a r*)
⟨*proof*⟩

lemma *lang-samequot*: ⟦*wf n r*; *a* ∈ Σ *n*⟧ $\Longrightarrow$
lang n (*samequot a r*) = *SAMEQUOT a* (*lang n r*)
⟨*proof*⟩

fun *rderiv-and-add* **where**
rderiv-and-add as (- :: *bool*, *rs*) =
(*let*
r = «*rderiv as* (*hd rs*)»
in if r ∈ *set rs then* (*False*, *rs*) *else* (*True*, *r* # *rs*))

definition *invar-rderiv-and-add as r brs* ≡
(*if fst brs then True else* «*rderiv as* (*hd* (*snd brs*))» ∈ *set* (*snd brs*)) ∧
snd brs ≠ [] ∧ *distinct* (*snd brs*) ∧

$(\forall\, i < \mathit{length}\ (\mathit{snd}\ \mathit{brs}).\ \mathit{snd}\ \mathit{brs}\ !\ i = «\mathit{rderivs}\ (\mathit{replicate}\ (\mathit{length}\ (\mathit{snd}\ \mathit{brs}) - 1 - i)\ \mathit{as})\ r»)$

lemma *invar-rderiv-and-add-init*: *invar-rderiv-and-add as r* (*True*, [«*r*»])
⟨*proof*⟩

lemma *invar-rderiv-and-add-step*: *invar-rderiv-and-add as r brs* ⟹ *fst brs* ⟹ *invar-rderiv-and-add as r* (*rderiv-and-add as brs*)
⟨*proof*⟩

lemma *rderivs-replicate-mult*: ⟦«*rderivs* (*replicate i as*) *r*» = «*r*»; $i > 0$⟧ ⟹ «*rderivs* (*replicate* ($m * i$) *as*) *r*» = «*r*»
⟨*proof*⟩

lemma *rderivs-replicate-mult-rest*:
assumes «*rderivs* (*replicate i as*) *r*» = «*r*» $k < i$
shows «*rderivs* (*replicate* ($m * i + k$) *as*) *r*» = «*rderivs* (*replicate k as*) *r*» (**is** *?L* = *?R*)
⟨*proof*⟩

lemma *rderivs-replicate-mod*:
assumes «*rderivs* (*replicate i as*) *r*» = «*r*» $i > 0$
shows «*rderivs* (*replicate m as*) *r*» = «*rderivs* (*replicate* (*m mod i*) *as*) *r*» (**is** *?L* = *?R*)
⟨*proof*⟩

lemma *rderivs-replicate-diff*: ⟦«*rderivs* (*replicate i as*) *r*» = «*rderivs* (*replicate j as*) *r*»; $i > j$⟧ ⟹
«*rderivs* (*replicate* ($i - j$) *as*) (*rderivs* (*replicate j as*) *r*)» = «*rderivs* (*replicate j as*) *r*»
⟨*proof*⟩

lemma *samequot-wf*:
assumes *wf n r while-option fst* (*rderiv-and-add as*) (*True*, [«*r*»]) = *Some* (*b*, *rs*)
shows *wf n* (*PLUS rs*)
⟨*proof*⟩

lemma *samequot-soundness*:
assumes *while-option fst* (*rderiv-and-add as*) (*True*, [«*r*»]) = *Some* (*b*, *rs*)
shows *lang n* (*PLUS rs*) = $\bigcup$ (*lang n* ‘ {«*rderivs* (*replicate m as*) *r*» | *m*. *True*})
⟨*proof*⟩

lemma *length-subset-card*: ⟦*finite X*; *distinct* (*x* # *xs*); *set* (*x* # *xs*) ⊆ *X*⟧ ⟹ *length xs* < *card X*
⟨*proof*⟩

lemma *samequot-termination*:
assumes *while-option fst* (*rderiv-and-add as*) (*True*, [«*r*»]) = *None* (**is** *?cl* =

None)
  shows *False*
⟨*proof*⟩

definition *samequot-exec a r* =
  Times (*PLUS* (*snd* (*the* (*while-option fst* (*rderiv-and-add a*) (*True*, [«*r*»]))))) (*Star* (*Atom* (*singleton a*)))

lemma *wf-samequot-exec*: ⟦*wf n r*; *as* ∈ Σ *n*⟧ ⟹ *wf n* (*samequot-exec as r*)
  ⟨*proof*⟩

lemma *samequot-exec-samequot*: *lang n* (*samequot-exec as r*) = *lang n* (*samequot as r*)
  ⟨*proof*⟩

lemma *lang-samequot-exec*:
  ⟦*wf n r*; *as* ∈ Σ *n*⟧ ⟹ *lang n* (*samequot-exec as r*) = *SAMEQUOT as* (*lang n r*)
⟨*proof*⟩

end

4.2 Suffix and Prefix Languages

definition *Suffix* :: ′*a lang* ⇒ ′*a lang* **where**
  Suffix L = {*w*. ∃ *u*. *u* @ *w* ∈ *L*}

definition *Prefix* :: ′*a lang* ⇒ ′*a lang* **where**
  Prefix L = {*w*. ∃ *u*. *w* @ *u* ∈ *L*}

lemma *Prefix-Suffix*: *Prefix L* = *rev* ‘ *Suffix* (*rev* ‘ *L*)
  ⟨*proof*⟩

definition *Root* :: ′*a lang* ⇒ ′*a lang* **where**
  Root L = {*x* . ∃ *n* > *0*. *x* ^^ *n* ∈ *L*}

definition *Cycle* :: ′*a lang* ⇒ ′*a lang* **where**
  Cycle L = {*u* @ *w* | *u w*. *w* @ *u* ∈ *L*}

context *embed*
begin

context
fixes *n* :: *nat*
begin

definition *SUFFIX* :: ′*b rexp* ⇒ ′*b rexp* **where**
  SUFFIX r = *flatten PLUS* {«*lderivs w r*»| *w*. *wf-word n w*}

lemma *finite-lderivs-wf*: *finite* {«*lderivs w r*»| *w*. *wf-word n w*}
⟨*proof*⟩

definition *PREFIX* :: *'b rexp* ⇒ *'b rexp* **where**
PREFIX r = *REV* (*SUFFIX* (*REV r*))

lemma *wf-SUFFIX*[*simp*]: *wf n r* ⟹ *wf n* (*SUFFIX r*)
⟨*proof*⟩

lemma *lang-SUFFIX*[*simp*]: *wf n r* ⟹ *lang n* (*SUFFIX r*) = *Suffix* (*lang n r*)
⟨*proof*⟩

lemma *wf-PREFIX*[*simp*]: *wf n r* ⟹ *wf n* (*PREFIX r*)
⟨*proof*⟩

lemma *lang-PREFIX*[*simp*]: *wf n r* ⟹ *lang n* (*PREFIX r*) = *Prefix* (*lang n r*)
⟨*proof*⟩

end

lemma *take-drop-CycleI*[*intro*!]: *x* ∈ *L* ⟹ *drop i x* @ *take i x* ∈ *Cycle L*
⟨*proof*⟩

lemma *take-drop-CycleI'*[*intro*!]: *drop i x* @ *take i x* ∈ *L* ⟹ *x* ∈ *Cycle L*
⟨*proof*⟩

end

5 Π-Extended Dual Regular Expressions

5.1 Syntax of regular expressions

datatype *'a rexp-dual* =
CoZero (*co*: *bool*) |
CoOne (*co*: *bool*) |
CoAtom (*co*: *bool*) *'a* |
CoPlus (*co*: *bool*) *'a rexp-dual 'a rexp-dual* |
CoTimes (*co*: *bool*) *'a rexp-dual 'a rexp-dual* |
CoStar (*co*: *bool*) *'a rexp-dual* |
CoPr (*co*: *bool*) *'a rexp-dual*
derive *linorder rexp-dual*

abbreviation *CoPLUS-dual b* ≡ *rexp-of-list* (*CoPlus b*) (*CoZero b*)
abbreviation *bool-unop-dual b* ≡ (*if b then id else HOL.Not*)
abbreviation *bool-binop-dual b* ≡ (*if b then* (∨) *else* (∧))
abbreviation *set-binop-dual b* ≡ (*if b then* (∪) *else* (∩))

primrec *final-dual* :: *'a rexp-dual* ⇒ *bool*

where
  final-dual (*CoZero b*) = (¬ *b*)
| *final-dual* (*CoOne b*) = *b*
| *final-dual* (*CoAtom b* -) = (¬ *b*)
| *final-dual* (*CoPlus b r s*) = *bool-binop-dual b* (*final-dual r*) (*final-dual s*)
| *final-dual* (*CoTimes b r s*) = *bool-binop-dual* (¬ *b*) (*final-dual r*) (*final-dual s*)
| *final-dual* (*CoStar b* -) = *b*
| *final-dual* (*CoPr* - *r*) = *final-dual r*

context *alphabet*
begin

primrec *wf-dual* :: *nat* ⇒ *'b rexp-dual* ⇒ *bool*
where
wf-dual n (*CoZero* -) = *True* |
wf-dual n (*CoOne* -) = *True* |
wf-dual n (*CoAtom* - *a*) = (*wf-atom n a*) |
wf-dual n (*CoPlus* - *r s*) = (*wf-dual n r* ∧ *wf-dual n s*) |
wf-dual n (*CoTimes* - *r s*) = (*wf-dual n r* ∧ *wf-dual n s*) |
wf-dual n (*CoStar* - *r*) = *wf-dual n r* |
wf-dual n (*CoPr* - *r*) = *wf-dual* (*n* + *1*) *r*

lemma *wf-dual-PLUS-dual*[*simp*]:
  wf-dual n (*CoPLUS-dual b xs*) = (∀ *r* ∈ *set xs*. *wf-dual n r*)
  ⟨*proof*⟩

abbreviation *set-unop-dual n b A* ≡ *if b then A else lists* (Σ *n*) − *A*

end

context *project*
begin

primrec *lang-dual* :: *nat* ⇒ *'b rexp-dual* => *'a lang* **where**
lang-dual n (*CoZero b*) = *set-unop-dual n b* {} |
lang-dual n (*CoOne b*) = *set-unop-dual n b* {[]} |
lang-dual n (*CoAtom b a*) = *set-unop-dual n b* {[*x*] | *x*. *lookup a x* ∧ *x* ∈ Σ *n*} |
lang-dual n (*CoPlus b r s*) = *set-binop-dual b* (*lang-dual n r*) (*lang-dual n s*) |
lang-dual n (*CoTimes b r s*) = *set-unop-dual n b*
  (*set-unop-dual n b* (*lang-dual n r*) @@ *set-unop-dual n b* (*lang-dual n s*)) |
lang-dual n (*CoStar b r*) = *set-unop-dual n b* (*star* (*set-unop-dual n b* (*lang-dual n r*))) |
lang-dual n (*CoPr b r*) = *set-unop-dual n b* (*map project* ‘ (*set-unop-dual* (*n* + *1*) *b* (*lang-dual* (*n* + *1*) *r*)))

lemma *wf-dual-lang-dual-wf-word*: *wf-dual n r* ⟹ ∀ *w* ∈ *lang-dual n r*. *wf-word n w*
  ⟨*proof*⟩

lemma *lang-dual-subset-lists*: *wf-dual n r* $\Longrightarrow$ *lang-dual n r* $\subseteq$ *lists* (Σ *n*)
⟨*proof*⟩

lemma *lang-dual-final-dual*: *final-dual r* = ([] $\in$ *lang-dual n r*)
⟨*proof*⟩

lemma *lang-dual-PLUS-dual*[*simp*]:
lang-dual n (*CoPLUS-dual True xs*) = ($\bigcup$ *r* $\in$ *set xs*. *lang-dual n r*)
⟨*proof*⟩

lemma *lang-dual-CoPLUS-dual*[*simp*]:
lang-dual n (*CoPLUS-dual False xs*) = (*if xs* = [] *then lists* (Σ *n*) *else* $\bigcap$ *r* $\in$ *set xs*. *lang-dual n r*)
⟨*proof*⟩

end

context *embed*
begin

primrec *lderiv-dual* :: *'a* $\Rightarrow$ *'b rexp-dual* $\Rightarrow$ *'b rexp-dual* **where**
lderiv-dual - (*CoZero b*) = (*CoZero b*)
| *lderiv-dual* - (*CoOne b*) = (*CoZero b*)
| *lderiv-dual a* (*CoAtom b c*) = (*if lookup c a then CoOne b else CoZero b*)
| *lderiv-dual a* (*CoPlus b r s*) = *CoPlus b* (*lderiv-dual a r*) (*lderiv-dual a s*)
| *lderiv-dual a* (*CoTimes b r s*) =
(*let r's* = *CoTimes b* (*lderiv-dual a r*) *s*
in if bool-unop-dual b (*final-dual r*) *then CoPlus b r's* (*lderiv-dual a s*) *else r's*)
| *lderiv-dual a* (*CoStar b r*) = *CoTimes b* (*lderiv-dual a r*) (*CoStar b r*)
| *lderiv-dual a* (*CoPr b r*) = *CoPr b* (*CoPLUS-dual b* (*map* ($\lambda a'$. *lderiv-dual a' r*) (*embed a*)))

primrec *lderivs-dual* **where**
lderivs-dual [] *r* = *r*
| *lderivs-dual* (*w#ws*) *r* = *lderivs-dual ws* (*lderiv-dual w r*)

lemma *wf-dual-lderiv-dual*[*simp*]: *wf-dual n r* $\Longrightarrow$ *wf-dual n* (*lderiv-dual w r*)
⟨*proof*⟩

lemma *wf-dual-lderivs-dual*[*simp*]: *wf-dual n r* $\Longrightarrow$ *wf-dual n* (*lderivs-dual ws r*)
⟨*proof*⟩

lemma *lang-dual-lderiv-dual*: $[\![$*wf-dual n r*; *w* $\in$ Σ *n*$]\!]$ $\Longrightarrow$
lang-dual n (*lderiv-dual w r*) = *lQuot w* (*lang-dual n r*)
⟨*proof*⟩

lemma *lang-dual-lderivs-dual*: $[\![$*wf-dual n r*; *wf-word n ws*$]\!]$ $\Longrightarrow$
lang-dual n (*lderivs-dual ws r*) = *lQuots ws* (*lang-dual n r*)
⟨*proof*⟩

corollary *lderivs-dual-final-dual*:
assumes *wf-dual n r wf-word n ws*
shows *final-dual* (*lderivs-dual ws r*) $\longleftrightarrow$ *ws* $\in$ *lang-dual n r*
⟨*proof*⟩

end

fun *pnCoPlus* :: *bool* $\Rightarrow$ *'a::linorder rexp-dual* $\Rightarrow$ *'a rexp-dual* $\Rightarrow$ *'a rexp-dual* **where**
pnCoPlus b1 (*CoZero b2*) *r* = (*if b1* = *b2 then r else CoZero b2*)
| *pnCoPlus b1 r* (*CoZero b2*) = (*if b1* = *b2 then r else CoZero b2*)
| *pnCoPlus b1* (*CoPlus b2 r s*) *t* =
(*if b1* = *b2 then pnCoPlus b2 r* (*pnCoPlus b2 s t*) *else CoPlus b1* (*CoPlus b2 r s*) *t*)
| *pnCoPlus b1 r* (*CoPlus b2 s t*) =
(*if b1* = *b2 then*
(*if r* = *s then* (*CoPlus b2 s t*)
else if r $\leq$ *s then CoPlus b2 r* (*CoPlus b2 s t*)
else CoPlus b2 s (*pnCoPlus b2 r t*))
else CoPlus b1 r (*CoPlus b2 s t*))
| *pnCoPlus b r s* =
(*if r* = *s then r*
else if r $\leq$ *s then CoPlus b r s*
else CoPlus b s r)

lemma (**in** *alphabet*) *wf-dual-pnCoPlus*[*simp*]: ⟦*wf-dual n r*; *wf-dual n s*⟧ $\Longrightarrow$ *wf-dual n* (*pnCoPlus b r s*)
⟨*proof*⟩

lemma (**in** *project*) *lang-dual-pnCoPlus*[*simp*]: ⟦*wf-dual n r*; *wf-dual n s*⟧ $\Longrightarrow$
lang-dual n (*pnCoPlus b r s*) = *lang-dual n* (*CoPlus b r s*)
⟨*proof*⟩

fun *pnCoTimes* :: *bool* $\Rightarrow$ *'a::linorder rexp-dual* $\Rightarrow$ *'a rexp-dual* $\Rightarrow$ *'a rexp-dual* **where**
pnCoTimes b1 (*CoZero b2*) *r* = (*if b1* = *b2 then CoZero b1 else CoTimes b1* (*CoZero b2*) *r*)
| *pnCoTimes b1* (*CoOne b2*) *r* = (*if b1* = *b2 then r else CoTimes b1* (*CoOne b2*) *r*)
| *pnCoTimes b1* (*CoPlus b2 r s*) *t* = (*if b1* = *b2 then pnCoPlus b2* (*pnCoTimes b2 r t*) (*pnCoTimes b2 s t*)
else CoTimes b1 (*CoPlus b2 r s*) *t*)
| *pnCoTimes b r s* = *CoTimes b r s*

lemma (**in** *alphabet*) *wf-dual-pnCoTimes*[*simp*]: ⟦*wf-dual n r*; *wf-dual n s*⟧ $\Longrightarrow$ *wf-dual n* (*pnCoTimes b r s*)
⟨*proof*⟩

lemma (**in** *project*) *lang-dual-pnCoTimes*[*simp*]: ⟦*wf-dual n r*; *wf-dual n s*⟧ $\Longrightarrow$

lang-dual n (*pnCoTimes b r s*) = *lang-dual n* (*CoTimes b r s*)
⟨*proof*⟩

fun *pnCoPr* :: *bool* ⇒ *'a*::*linorder rexp-dual* ⇒ *'a rexp-dual* **where**
pnCoPr b1 (*CoZero b2*) = (*if b1* = *b2 then CoZero b2 else CoPr b1* (*CoZero b2*))
| *pnCoPr b1* (*CoOne b2*) = (*if b1* = *b2 then CoOne b2 else CoPr b1* (*CoOne b2*))
| *pnCoPr b1* (*CoPlus b2 r s*) = (*if b1* = *b2 then pnCoPlus b2* (*pnCoPr b2 r*) (*pnCoPr b2 s*)
else CoPr b1 (*CoPlus b2 r s*))
| *pnCoPr b r* = *CoPr b r*

lemma (**in** *alphabet*) *wf-dual-pnCoPr*[*simp*]: *wf-dual* (*Suc n*) *r* ⟹ *wf-dual n* (*pnCoPr b r*)
⟨*proof*⟩

lemma (**in** *project*) *lang-dual-pnCoPr*[*simp*]: *wf-dual* (*Suc n*) *r* ⟹ *lang-dual n* (*pnCoPr b r*) = *lang-dual n* (*CoPr b r*)
⟨*proof*⟩

primrec *pnorm-dual* :: *'a*::*linorder rexp-dual* ⇒ *'a rexp-dual* **where**
pnorm-dual (*CoZero b*) = (*CoZero b*)
| *pnorm-dual* (*CoOne b*) = (*CoOne b*)
| *pnorm-dual* (*CoAtom b a*) = (*CoAtom b a*)
| *pnorm-dual* (*CoPlus b r s*) = *pnCoPlus b* (*pnorm-dual r*) (*pnorm-dual s*)
| *pnorm-dual* (*CoTimes b r s*) = *pnCoTimes b* (*pnorm-dual r*) *s*
| *pnorm-dual* (*CoStar b r*) = *CoStar b r*
| *pnorm-dual* (*CoPr b r*) = *pnCoPr b* (*pnorm-dual r*)

lemma (**in** *alphabet*) *wf-dual-pnorm-dual*[*simp*]: *wf-dual n r* ⟹ *wf-dual n* (*pnorm-dual r*)
⟨*proof*⟩

lemma (**in** *project*) *lang-dual-pnorm-dual*[*simp*]: *wf-dual n r* ⟹ *lang-dual n* (*pnorm-dual r*) = *lang-dual n r*
⟨*proof*⟩

primrec *CoNot* **where**
CoNot (*CoZero b*) = *CoZero* (¬ *b*)
| *CoNot* (*CoOne b*) = *CoOne* (¬ *b*)
| *CoNot* (*CoAtom b a*) = *CoAtom* (¬ *b*) *a*
| *CoNot* (*CoPlus b r s*) = *CoPlus* (¬ *b*) (*CoNot r*) (*CoNot s*)
| *CoNot* (*CoTimes b r s*) = *CoTimes* (¬ *b*) (*CoNot r*) (*CoNot s*)
| *CoNot* (*CoStar b r*) = *CoStar* (¬ *b*) (*CoNot r*)
| *CoNot* (*CoPr b r*) = *CoPr* (¬ *b*) (*CoNot r*)

primrec *rexp-dual-of* **where**
rexp-dual-of Zero = *CoZero True*
| *rexp-dual-of Full* = *CoZero False*

| $\textit{rexp-dual-of One} = \textit{CoOne True}$
| $\textit{rexp-dual-of}\ (\textit{Atom}\ a) = \textit{CoAtom True}\ a$
| $\textit{rexp-dual-of}\ (\textit{Plus}\ r\ s) = \textit{CoPlus True}\ (\textit{rexp-dual-of}\ r)\ (\textit{rexp-dual-of}\ s)$
| $\textit{rexp-dual-of}\ (\textit{Times}\ r\ s) = \textit{CoTimes True}\ (\textit{rexp-dual-of}\ r)\ (\textit{rexp-dual-of}\ s)$
| $\textit{rexp-dual-of}\ (\textit{Star}\ r) = \textit{CoStar True}\ (\textit{rexp-dual-of}\ r)$
| $\textit{rexp-dual-of}\ (\textit{Not}\ r) = \textit{CoNot}\ (\textit{rexp-dual-of}\ r)$
| $\textit{rexp-dual-of}\ (\textit{Inter}\ r\ s) = \textit{CoPlus False}\ (\textit{rexp-dual-of}\ r)\ (\textit{rexp-dual-of}\ s)$
| $\textit{rexp-dual-of}\ (\textit{Pr}\ r) = \textit{CoPr True}\ (\textit{rexp-dual-of}\ r)$

lemma (**in** *alphabet*) *wf-dual-CoNot*[*simp*]: $\textit{wf-dual}\ n\ r \Longrightarrow \textit{wf-dual}\ n\ (\textit{CoNot}\ r)$
⟨*proof*⟩

lemma (**in** *project*) *lang-dual-CoNot*[*simp*]: $\textit{wf-dual}\ n\ r \Longrightarrow \textit{lang-dual}\ n\ (\textit{CoNot}\ r) = \textit{lists}\ (\Sigma\ n) - \textit{lang-dual}\ n\ r$
⟨*proof*⟩

lemma (**in** *alphabet*) *wf-dual-rexp-dual-of*[*simp*]: $\textit{wf}\ n\ r \Longrightarrow \textit{wf-dual}\ n\ (\textit{rexp-dual-of}\ r)$
⟨*proof*⟩

lemma (**in** *project*) *lang-dual-rexp-dual-of*[*simp*]: $\textit{wf}\ n\ r \Longrightarrow \textit{lang-dual}\ n\ (\textit{rexp-dual-of}\ r) = \textit{lang}\ n\ r$
⟨*proof*⟩

end

6 Deciding Equivalence of Π-Extended Regular Expressions

lemma *image2p-in-rel*: $\textit{BNF-Greatest-Fixpoint.image2p}\ f\ g\ (\textit{in-rel}\ R) = \textit{in-rel}\ (\textit{map-prod}\ f\ g\ \text{`}\ R)$
⟨*proof*⟩

lemma *image2p-apply*: $\textit{BNF-Greatest-Fixpoint.image2p}\ f\ g\ R\ x\ y = (\exists\, x'\ y'.\ R\ x'\ y' \wedge f\ x' = x \wedge g\ y' = y)$
⟨*proof*⟩

lemma *rtrancl-fold-product*:
shows $\{((r, s), (f\ a\ r, f\ a\ s))|\ r\ s\ a.\ a \in A\}\hat{}* =$
$\{((r, s), (\textit{fold}\ f\ w\ r, \textit{fold}\ f\ w\ s))|\ r\ s\ w.\ w \in \textit{lists}\ A\}$ (**is** *?L* = *?R*)
⟨*proof*⟩

lemma *in-fold-lQuot*: $v \in \textit{fold lQuot}\ w\ L \longleftrightarrow w\ @\ v \in L$
⟨*proof*⟩

lemma (**in** *project*) *lang-eq-ext*: $[\![\textit{wf}\ n\ r;\ \textit{wf}\ n\ s]\!] \Longrightarrow (\textit{lang}\ n\ r = \textit{lang}\ n\ s) =$
$(\forall\, w \in \textit{lists}(\Sigma\ n).\ w \in \textit{lang}\ n\ r \longleftrightarrow w \in \textit{lang}\ n\ s)$

$\langle proof \rangle$

lemma (**in** *project*) *lang-eq-ext-Nil-fold-Deriv*:
  fixes $r\ s\ n$
  assumes *WF*: *wf n r wf n s*
  defines $\mathfrak{B} \equiv \{(fold\ lQuot\ w\ (lang\ n\ r),\ fold\ lQuot\ w\ (lang\ n\ s))|\ w.\ w{\in}lists\ (\Sigma\ n)\}$
  shows $lang\ n\ r = lang\ n\ s \longleftrightarrow (\forall (K,\ L) \in \mathfrak{B}.\ [] \in K \longleftrightarrow [] \in L)$
  $\langle proof \rangle$

locale *rexp-DA* = *project set o* σ *wf-atom project lookup*
  for σ :: $nat \Rightarrow 'a\ list$
  and *wf-atom* :: $nat \Rightarrow 'b :: linorder \Rightarrow bool$
  and *project* :: $'a \Rightarrow 'a$
  and *lookup* :: $'b \Rightarrow 'a \Rightarrow bool$ +
  fixes *init* :: $'b\ rexp \Rightarrow 's$
  fixes *delta* :: $'a \Rightarrow 's \Rightarrow 's$
  fixes *final* :: $'s \Rightarrow bool$
  fixes *wf-state* :: $'s \Rightarrow bool$
  fixes *post* :: $'s \Rightarrow 's$
  fixes L :: $'s \Rightarrow 'a\ lang$
  fixes n :: nat
  assumes *L-init*[*simp*]: $wf\ n\ r \Longrightarrow L\ (init\ r) = lang\ n\ r$
  assumes *L-delta*[*simp*]: $[\![a \in set\ (\sigma\ n);\ wf\text{-}state\ s]\!] \Longrightarrow L\ (delta\ a\ s) = lQuot\ a\ (L\ s)$
  assumes *final-iff-Nil*[*simp*]: $final\ s \longleftrightarrow [] \in L\ s$
  assumes *L-wf-state*[*dest*]: $wf\text{-}state\ s \Longrightarrow L\ s \subseteq lists\ (set\ (\sigma\ n))$
  assumes *init-wf-state*[*simp*]: $wf\ n\ r \Longrightarrow wf\text{-}state\ (init\ r)$
  assumes *delta-wf-state*[*simp*]: $[\![a \in set\ (\sigma\ n);\ wf\text{-}state\ s]\!] \Longrightarrow wf\text{-}state\ (delta\ a\ s)$
  assumes *L-post*[*simp*]: $wf\text{-}state\ s \Longrightarrow L\ (post\ s) = L\ s$
  assumes *wf-state-post*[*simp*]: $wf\text{-}state\ s \Longrightarrow wf\text{-}state\ (post\ s)$
begin

lemma *L-deltas*[*simp*]: $[\![wf\text{-}word\ n\ w;\ wf\text{-}state\ s]\!] \Longrightarrow L\ (fold\ delta\ w\ s) = fold\ lQuot\ w\ (L\ s)$
  $\langle proof \rangle$

definition *progression* (**infix** ‹→› *60*) **where**
  $R \rightarrow S = (\forall\, s1\ s2.\ R\ s1\ s2 \longrightarrow wf\text{-}state\ s1 \wedge wf\text{-}state\ s2 \wedge final\ s1 = final\ s2 \wedge$
    $(\forall\, x \in set\ (\sigma\ n).\ BNF\text{-}Greatest\text{-}Fixpoint.image2p\ post\ post\ S\ (post\ (delta\ x\ s1))\ (post\ (delta\ x\ s2))))$

lemma *SUPR-progression*[*intro*!]: $\forall\, n.\ \exists\, m.\ X\ n \rightarrow Y\ m \Longrightarrow (SUP\ n.\ X\ n) \rightarrow (SUP\ n.\ Y\ n)$
  $\langle proof \rangle$

definition *bisimulation* **where**
  $bisimulation\ R = R \rightarrow R$

definition *bisimulation-upto* **where**
bisimulation-upto $R\ f = R \to f\ R$

declare *image2pI*[*intro*!] *image2pE*[*elim*!]
lemmas *bisim-def* = *bisimulation-def progression-def*
lemmas *bisim-upto-def* = *bisimulation-upto-def progression-def*

definition *compatible* **where**
compatible $f = (mono\ f \wedge (\forall R\ S.\ R \to S \longrightarrow f\ R \to f\ S))$

lemmas *compat-def* = *compatible-def progression-def*

lemma *bisimulation-upto-bisimulation*:
assumes *compatible f bisimulation-upto R f*
obtains *S* **where** *bisimulation* $S\ R \le S$
⟨*proof*⟩

lemma *bisimulation-eqL*: *bisimulation* (λ*s1 s2*. *wf-state s1* ∧ *wf-state s2* ∧ *L s1* = *L s2*)
⟨*proof*⟩

lemma *coinduction*:
assumes *bisim*[*unfolded bisim-def*]: *bisimulation R* **and**
WF: *wf-state s1 wf-state s2* **and** *R*: *R s1 s2*
shows *L s1* = *L s2*
⟨*proof*⟩

lemma *coinduction-upto*:
assumes *bisimulation-upto R f* **and** *WF*: *wf-state s1 wf-state s2* **and** *R s1 s2* *compatible f*
shows *L s1* = *L s2*
⟨*proof*⟩

fun *test-invariant* **where**
test-invariant (*ws*, - :: ('*s* × '*s*) *list* , - :: '*s rel*) = (*case ws of* [] ⇒ *False* | (*w*::'*a list*,*p*,*q*)#- ⇒ *final p* = *final q*)
fun *test* **where** *test* (*ws*, - :: '*s rel*) = (*case ws of* [] ⇒ *False* | (*p*,*q*)#- ⇒ *final p* = *final q*)

fun *step-invariant* **where** *step-invariant* (*ws*, *ps*, *N*) =
(*let*
(*w*, *r*, *s*) = *hd ws*;
ps′ = (*r*, *s*) # *ps*;
succs = *map* (λ*a*.
let r′ = *delta a r*; *s*′ = *delta a s*
in ((*a* # *w*, *r*′, *s*′), (*post r*′, *post s*′))) (σ *n*);
new = *remdups*′ *snd* (*filter* (λ(-, *rs*). *rs* ∉ *N*) *succs*);
ws′ = *tl ws* @ *map fst new*;
N′ = *set* (*map snd new*) ∪ *N*

$in\ (ws',\ ps',\ N'))$

fun *step* **where** $step\ (ws,\ N) =$
$(let$
$(r,\ s) = hd\ ws;$
$succs = map\ (\lambda a.$
$let\ r' = delta\ a\ r;\ s' = delta\ a\ s$
$in\ ((r',\ s'),\ (post\ r',\ post\ s')))\ (\sigma\ n);$
$new = remdups'\ snd\ (filter\ (\lambda(\text{-},\ rs).\ rs \notin N)\ succs)$
$in\ (tl\ ws\ @\ map\ fst\ new,\ set\ (map\ snd\ new) \cup N))$

definition *closure-invariant* **where** *closure-invariant* = *while-option test-invariant step-invariant*
definition *closure* **where** *closure* = *while-option test step*

definition *invariant* **where**
$invariant\ r\ s = (\lambda(ws,\ ps,\ N).$
$(r,\ s) \in snd\ `\ set\ ws \cup set\ ps\ \wedge$
$distinct\ (map\ snd\ ws\ @\ ps)\ \wedge$
$bij\text{-}betw\ (map\text{-}prod\ post\ post)\ (set\ (map\ snd\ ws\ @\ ps))\ N\ \wedge$
$(\forall (w,\ r',\ s') \in set\ ws.\ fold\ delta\ (rev\ w)\ r = r' \wedge fold\ delta\ (rev\ w)\ s = s'\ \wedge$
$wf\text{-}word\ n\ (rev\ w) \wedge wf\text{-}state\ r' \wedge wf\text{-}state\ s') \wedge$
$(\forall (r',\ s') \in set\ ps.\ (\exists w.\ fold\ delta\ w\ r = r' \wedge fold\ delta\ w\ s = s')\ \wedge$
$wf\text{-}state\ r' \wedge wf\text{-}state\ s' \wedge (final\ r' \longleftrightarrow final\ s') \wedge$
$(\forall a \in set\ (\sigma\ n).\ (post\ (delta\ a\ r'),\ post\ (delta\ a\ s')) \in N)))$

lemma *invariant-start*:
$[\![wf\text{-}state\ r;\ wf\text{-}state\ s]\!] \Longrightarrow invariant\ r\ s\ ([([],\ r,\ s)],\ [],\ \{(post\ r,\ post\ s)\})$
$\langle proof \rangle$

lemma *step-invariant-mono*:
assumes $step\text{-}invariant\ (ws,\ ps,\ N) = (ws',\ ps',\ N')$
shows $snd\ `\ set\ ws \cup set\ ps \subseteq snd\ `\ set\ ws' \cup set\ ps'$
$\langle proof \rangle$

lemma *step-invatiant-unfold*: $step\text{-}invariant\ (w\ \#\ ws,\ ps,\ N) = (ws',\ ps',\ N') \Longrightarrow$
$(\exists xs\ r\ s.$
$w = (xs,\ r,\ s) \wedge ps' = (r,\ s)\ \#\ ps\ \wedge$
$ws' = ws\ @\ remdups'\ (map\text{-}prod\ post\ post\ o\ snd)\ (filter\ (\lambda(\text{-},\ p).\ map\text{-}prod\ post\ post\ p \notin N)$
$(map\ (\lambda a.\ (a\#xs,\ delta\ a\ r,\ delta\ a\ s))\ (\sigma\ n)))\ \wedge$
$N' = set\ (map\ (\lambda a.\ (post\ (delta\ a\ r),\ post\ (delta\ a\ s)))\ (\sigma\ n)) \cup N)$
$\langle proof \rangle$

lemma *invariant*: $invariant\ r\ s\ st \Longrightarrow test\text{-}invariant\ st \Longrightarrow invariant\ r\ s\ (step\text{-}invariant\ st)$
$\langle proof \rangle$

lemma *step-commute*: $ws \neq [] \Longrightarrow$

$(case\ step\text{-}invariant\ (ws,\ ps,\ N)\ of\ (ws',\ ps',\ N') \Rightarrow (map\ snd\ ws',\ N')) = step\ (map\ snd\ ws,\ N)$
⟨proof⟩

lemma *closure-invariant-closure*:
$map\text{-}option\ (\lambda(ws,\ ps,\ N).\ (map\ snd\ ws,\ N))\ (closure\text{-}invariant\ (ws,\ ps,\ N)) = closure\ (map\ snd\ ws,\ N)$
⟨proof⟩

lemma
assumes *result*: $closure\text{-}invariant\ ([([],\ init\ r,\ init\ s)],\ [],\ \{(post\ (init\ r),\ post\ (init\ s))\}) =$
$Some(ws,\ ps,\ N)$ (**is** $closure\text{-}invariant\ ([([],\ ?r,\ ?s)],\ \text{-}) = \text{-}$)
and *WF*: $wf\ n\ r\ wf\ n\ s$
shows *closure-invariant-sound*: $ws = [] \Longrightarrow lang\ n\ r = lang\ n\ s$ **and**
counterexample: $ws \neq [] \Longrightarrow rev\ (fst\ (hd\ ws)) \in lang\ n\ r \longleftrightarrow rev\ (fst\ (hd\ ws)) \notin lang\ n\ s$
⟨proof⟩

lemma *closure-sound*:
assumes *result*: $closure\ ([(init\ r,\ init\ s)],\ \{(post\ (init\ r),\ post\ (init\ s))\}) = Some\ ([],\ N)$
and *WF*: $wf\ n\ r\ wf\ n\ s$
shows $lang\ n\ r = lang\ n\ s$
⟨proof⟩

definition *check-eqv* **where**
$check\text{-}eqv\ r\ s =$
$(let\ r' = init\ r;\ s' = init\ s\ in\ (case\ closure\ ([(r',\ s')],\ \{(post\ r',\ post\ s')\})\ of$
$Some\ ([],\ \text{-}) \Rightarrow True \mid \text{-} \Rightarrow False))$

lemma *check-eqv-sound*:
assumes $check\text{-}eqv\ r\ s$ **and** *WF*: $wf\ n\ r\ wf\ n\ s$
shows $lang\ n\ r = lang\ n\ s$
⟨proof⟩

definition *counterexample* **where**
$counterexample\ r\ s =$
$(let\ r' = init\ r;\ s' = init\ s\ in\ (case\ closure\text{-}invariant\ ([([],\ r',\ s')],\ [],\ \{(post\ r',\ post\ s')\})\ of$
$Some((w,\text{-},\text{-})\ \#\ \text{-},\ \text{-}) \Rightarrow Some\ (rev\ w) \mid \text{-} => None))$

lemma *counterexample-sound*:
assumes *result*: $counterexample\ r\ s = Some\ w$ **and** *WF*: $wf\ n\ r\ wf\ n\ s$
shows $w \in lang\ n\ r \longleftrightarrow w \notin lang\ n\ s$
⟨proof⟩

Auxiliary exacutable functions:

definition *reachable* :: $'b\ rexp \Rightarrow 's\ set$ **where**

reachable s = *snd* (*the* (*rtrancl-while* (λ-. *True*) (λ*s*. *map* (λ*a*. *post* (*delta a s*)) (σ *n*)) (*init s*)))

definition *automaton* :: ′*b rexp* ⇒ ((′*s* ∗ ′*a*) ∗ ′*s*) *set* **where**
automaton s =
snd (*the*
(*let i* = *init s*;
start = (([*i*], {*post i*}), {});
test-invariant = λ((*ws*, *Z*), *A*). *ws* ≠ [];
step-invariant = λ((*ws*, *Z*), *A*).
(*let s* = *hd ws*;
new-edges = *map* (λ*a*. ((*s*, *a*), *delta a s*)) (σ *n*);
new = *remdups* (*filter* (λ*ss*. *post ss* ∉ *Z*) (*map snd new-edges*))
in ((*new* @ *tl ws*, *post* ‘ *set new* ∪ *Z*), *set new-edges* ∪ *A*))
in while-option test-invariant step-invariant start))

definition *match* :: ′*b rexp* ⇒ ′*a list* ⇒ *bool* **where**
match s w = *final* (*fold delta w* (*init s*))

lemma *match-correct*: ⟦*wf-word n w*; *wf n s*⟧ ⟹ *match s w* ⟷ *w* ∈ *lang n s*
⟨*proof*⟩

end

locale *rexp-DFA* = *rexp-DA* σ *wf-atom project lookup init delta final wf-state post L n*
for σ :: *nat* ⇒ ′*a list*
and *wf-atom* :: *nat* ⇒ ′*b* :: *linorder* ⇒ *bool*
and *project* :: ′*a* ⇒ ′*a*
and *lookup* :: ′*b* ⇒ ′*a* ⇒ *bool*
and *init* :: ′*b rexp* ⇒ ′*s*
and *delta* :: ′*a* ⇒ ′*s* ⇒ ′*s*
and *final* :: ′*s* ⇒ *bool*
and *wf-state* :: ′*s* ⇒ *bool*
and *post* :: ′*s* ⇒ ′*s*
and *L* :: ′*s* ⇒ ′*a lang*
and *n* :: *nat* +
assumes *fin*: *finite* {*fold delta w* (*init s*) | *w*. *True*}
begin

abbreviation *Reachable s* ≡ {*fold delta w* (*init s*) | *w*. *True*}

lemma *closure-invariant-termination*:
assumes *WF*: *wf n r wf n s*
and *result*: *closure-invariant* ([([], *init r*, *init s*)], [], {(*post* (*init r*), *post* (*init s*))}) = *None*
(**is** *closure-invariant* ([([], *?r*, *?s*)], -) = *None* **is** *?cl* = *None*)
shows *False*
⟨*proof*⟩

lemma *closure-termination*:
  assumes *WF*: *wf n r wf n s*
  and *result*: *closure* $([(init\ r,\ init\ s)],\ \{(post\ (init\ r),\ post\ (init\ s))\}) = None$
  shows *False*
  ⟨*proof*⟩

lemma *closure-invariant-complete*:
  assumes *eq*: *lang n r* = *lang n s*
  and *WF*: *wf n r wf n s*
  shows ∃ *ps N*. *closure-invariant* $([([],\ init\ r,\ init\ s)],\ [],\ \{(post\ (init\ r),\ post\ (init\ s))\}) =$
    Some([], *ps*, *N*) (**is** ∃ - -. *closure-invariant* ([([], *?r*, *?s*)], -) = - **is** ∃ - -. *?cl* = -)
⟨*proof*⟩

lemma *closure-complete*:
  assumes *lang n r* = *lang n s wf n r wf n s*
  shows ∃ *N*. *closure* $([(init\ r,\ init\ s)],\ \{(post\ (init\ r),\ post\ (init\ s))\}) = Some\ ([],\ N)$
  ⟨*proof*⟩

lemma *check-eqv-complete*:
  assumes *lang n r* = *lang n s wf n r wf n s*
  shows *check-eqv r s*
  ⟨*proof*⟩

lemma *counterexample-complete*:
  assumes *lang n r* ≠ *lang n s* **and** *WF*: *wf n r wf n s*
  shows ∃ *w*. *counterexample r s* = *Some w*
  ⟨*proof*⟩

end

locale *rexp-DA-no-post* = *rexp-DA σ wf-atom project lookup init delta final wf-state id L n*
  for σ :: *nat* ⇒ *'a list*
  and *wf-atom* :: *nat* ⇒ *'b* :: *linorder* ⇒ *bool*
  and *project* :: *'a* ⇒ *'a*
  and *lookup* :: *'b* ⇒ *'a* ⇒ *bool*
  and *init* :: *'b rexp* ⇒ *'s*
  and *delta* :: *'a* ⇒ *'s* ⇒ *'s*
  and *final* :: *'s* ⇒ *bool*
  and *wf-state* :: *'s* ⇒ *bool*
  and *L* :: *'s* ⇒ *'a lang*
  and *n* :: *nat*
begin

lemma *step-efficient*[*code*]: *step* (*ws*, *N*) =
  (*let*

$(r,\ s) = hd\ ws$;
$new = remdups\ (filter\ (\lambda(r,s).\ (r,s) \notin N)\ (map\ (\lambda a.\ (delta\ a\ r,\ delta\ a\ s))\ (\sigma\ n)))$
$in\ (tl\ ws\ @\ new,\ set\ new \cup N))$
$\langle proof \rangle$

end

locale *rexp-DFA-no-post* = *rexp-DFA* σ *wf-atom project lookup init delta final wf-state id L*
for σ :: *nat* $\Rightarrow$ *'a list*
and *wf-atom* :: *nat* $\Rightarrow$ *'b* :: *linorder* $\Rightarrow$ *bool*
and *project* :: *'a* $\Rightarrow$ *'a*
and *lookup* :: *'b* $\Rightarrow$ *'a* $\Rightarrow$ *bool*
and *init* :: *'b rexp* $\Rightarrow$ *'s*
and *delta* :: *'a* $\Rightarrow$ *'s* $\Rightarrow$ *'s*
and *final* :: *'s* $\Rightarrow$ *bool*
and *wf-state* :: *'s* $\Rightarrow$ *bool*
and *L* :: *'s* $\Rightarrow$ *'a lang*
begin

sublocale *rexp-DA-no-post* $\langle proof \rangle$

end

locale *rexp-DA-sim* = *project set o* σ *wf-atom project lookup*
for σ :: *nat* $\Rightarrow$ *'a list*
and *wf-atom* :: *nat* $\Rightarrow$ *'b* :: *linorder* $\Rightarrow$ *bool*
and *project* :: *'a* $\Rightarrow$ *'a*
and *lookup* :: *'b* $\Rightarrow$ *'a* $\Rightarrow$ *bool* +
fixes *init* :: *'b rexp* $\Rightarrow$ *'s*
fixes *sim-delta* :: *'s* $\Rightarrow$ *'s list*
fixes *final* :: *'s* $\Rightarrow$ *bool*
fixes *wf-state* :: *'s* $\Rightarrow$ *bool*
fixes *L* :: *'s* $\Rightarrow$ *'a lang*
fixes *post* :: *'s* $\Rightarrow$ *'s*
fixes *n* :: *nat*
assumes *L-init*[*simp*]: $wf\ n\ r \Longrightarrow L\ (init\ r) = lang\ n\ r$
assumes *final-iff-Nil*[*simp*]: $final\ s \longleftrightarrow [] \in L\ s$
assumes *L-wf-state*[*dest*]: $wf\text{-}state\ s \Longrightarrow L\ s \subseteq lists\ (set\ (\sigma\ n))$
assumes *init-wf-state*[*simp*]: $wf\ n\ r \Longrightarrow wf\text{-}state\ (init\ r)$
assumes *L-post*[*simp*]: $wf\text{-}state\ s \Longrightarrow L\ (post\ s) = L\ s$
assumes *wf-state-post*[*simp*]: $wf\text{-}state\ s \Longrightarrow wf\text{-}state\ (post\ s)$
assumes *L-sim-delta*[*simp*]: $wf\text{-}state\ s \Longrightarrow map\ L\ (sim\text{-}delta\ s) = map\ (\lambda a.\ lQuot\ a\ (L\ s))\ (\sigma\ n)$
assumes *sim-delta-wf-state*[*simp*]: $wf\text{-}state\ s \Longrightarrow \forall\, s' \in set\ (sim\text{-}delta\ s).\ wf\text{-}state\ s'$
begin

definition *delta a s* = *sim-delta s* ! *index* (σ *n*) *a*

lemma *length-sim-delta*[*simp*]: *wf-state s* $\Longrightarrow$ *length* (*sim-delta s*) = *length* (σ *n*)
⟨*proof*⟩

lemma *L-delta*[*simp*]: ⟦*a* $\in$ *set* (σ *n*); *wf-state s*⟧ $\Longrightarrow$ *L* (*delta a s*) = *lQuot a* (*L s*)
⟨*proof*⟩

lemma *delta-wf-state*[*simp*]: ⟦*a* $\in$ *set* (σ *n*); *wf-state s*⟧ $\Longrightarrow$ *wf-state* (*delta a s*)
⟨*proof*⟩

sublocale *rexp-DA* σ *wf-atom project lookup init delta final wf-state post L*
⟨*proof*⟩

sublocale *rexp-DA-sim-no-post*: *rexp-DA-no-post* σ *wf-atom project lookup init delta final wf-state L*
⟨*proof*⟩

end

7 Initial Normalization of the Input

fun *toplevel-inters* **where**
toplevel-inters (*Inter r s*) = *toplevel-inters r* $\cup$ *toplevel-inters s*
| *toplevel-inters r* = {*r*}

lemma *toplevel-inters-nonempty*[*simp*]:
toplevel-inters r $\neq$ {}
⟨*proof*⟩

lemma *toplevel-inters-finite*[*simp*]:
finite (*toplevel-inters r*)
⟨*proof*⟩

context *alphabet*
begin

lemma *toplevel-inters-wf*:
wf n s = ($\forall$ *r*$\in$*toplevel-inters s*. *wf n r*)
⟨*proof*⟩

end

context *project*
begin

lemma *toplevel-inters-lang*:
  $r \in$ *toplevel-inters* $s \Longrightarrow$ *lang* n $s \subseteq$ *lang* n r
  ⟨*proof*⟩

lemma *toplevel-inters-lang-INT*:
  lang n $s = (\bigcap r{\in}$*toplevel-inters* $s.$ *lang* n $r)$
  ⟨*proof*⟩

lemma *toplevel-inters-in-lang*:
  $w \in$ *lang* n $s = (\forall\, r{\in}$*toplevel-inters* $s.$ $w \in$ *lang* n $r)$
  ⟨*proof*⟩

lemma *lang-flatten-INTERSECT-finite*[*simp*]:
  finite $X \Longrightarrow w \in$ *lang* n (*flatten INTERSECT* X) $=$
    (*if* $X = \{\}$ *then* $w \in$ *lists* $(\Sigma\ n)$ *else* $(\forall\, r \in X.\ w \in$ *lang* n $r))$
  ⟨*proof*⟩

end

fun *merge-distinct* **where**
  merge-distinct [] *xs* = *xs*
| *merge-distinct xs* [] = *xs*
| *merge-distinct* (*a* # *xs*) (*b* # *ys*) =
    (*if* $a = b$ *then merge-distinct xs* (*b* # *ys*)
    else if $a < b$ *then a* # *merge-distinct xs* (*b* # *ys*)
    else b # *merge-distinct* (*a* # *xs*) *ys*)

lemma *set-merge-distinct*[*simp*]: *set* (*merge-distinct xs ys*) = *set xs* $\cup$ *set ys*
  ⟨*proof*⟩

lemma *sorted-merge-distinct*[*simp*]: ⟦*sorted xs*; *sorted ys*⟧ $\Longrightarrow$ *sorted* (*merge-distinct xs ys*)
  ⟨*proof*⟩

lemma *distinct-merge-distinct*[*simp*]: ⟦*sorted xs*; *distinct xs*; *sorted ys*; *distinct ys*⟧ $\Longrightarrow$
  distinct (*merge-distinct xs ys*)
  ⟨*proof*⟩

lemma *sorted-list-of-set-merge-distinct*[*simp*]: ⟦*sorted xs*; *distinct xs*; *sorted ys*; *distinct ys*⟧ $\Longrightarrow$
  merge-distinct xs ys = *sorted-list-of-set* (*set xs* $\cup$ *set ys*)
  ⟨*proof*⟩

fun *zip-with-option* **where**
  zip-with-option f (*Some a*) (*Some b*) = *Some* (*f a b*)
| *zip-with-option* - - - = *None*

lemma *zip-with-option-eq-Some*[*simp*]:
  $zip\text{-}with\text{-}option\ f\ x\ y = Some\ z \longleftrightarrow (\exists\, a\ b.\ z = f\ a\ b \wedge x = Some\ a \wedge y = Some\ b)$
  ⟨*proof*⟩

fun *Pluss* **where**
  $Pluss\ (Plus\ r\ s) = zip\text{-}with\text{-}option\ merge\text{-}distinct\ (Pluss\ r)\ (Pluss\ s)$
$|\ Pluss\ Zero = Some\ []$
$|\ Pluss\ Full = None$
$|\ Pluss\ r = Some\ [r]$

lemma *Pluss-None*[*symmetric*]: $Pluss\ r = None \longleftrightarrow Full \in toplevel\text{-}summands\ r$
  ⟨*proof*⟩

lemma *Pluss-Some*: $Pluss\ r = Some\ xs \longleftrightarrow$
  $(Full \notin set\ xs \wedge xs = sorted\text{-}list\text{-}of\text{-}set\ (toplevel\text{-}summands\ r - \{Zero\}))$
⟨*proof*⟩

fun *Inters* **where**
  $Inters\ (Inter\ r\ s) = zip\text{-}with\text{-}option\ merge\text{-}distinct\ (Inters\ r)\ (Inters\ s)$
$|\ Inters\ Zero = None$
$|\ Inters\ Full = Some\ []$
$|\ Inters\ r = Some\ [r]$

lemma *Inters-None*[*symmetric*]: $Inters\ r = None \longleftrightarrow Zero \in toplevel\text{-}inters\ r$
  ⟨*proof*⟩

lemma *Inters-Some*: $Inters\ r = Some\ xs \longleftrightarrow$
  $(Zero \notin set\ xs \wedge xs = sorted\text{-}list\text{-}of\text{-}set\ (toplevel\text{-}inters\ r - \{Full\}))$
⟨*proof*⟩

definition *inPlus* **where**
  $inPlus\ r\ s = (case\ Pluss\ (Plus\ r\ s)\ of\ None \Rightarrow Full\ |\ Some\ rs \Rightarrow PLUS\ rs)$

lemma *inPlus-alt*: $inPlus\ r\ s = (let\ X = toplevel\text{-}summands\ (Plus\ r\ s) - \{Zero\}\ in$
  $flatten\ PLUS\ (if\ Full \in X\ then\ \{Full\}\ else\ X))$
⟨*proof*⟩

fun *inTimes* **where**
  $inTimes\ Zero\ \text{-} = Zero$
$|\ inTimes\ \text{-}\ Zero = Zero$
$|\ inTimes\ One\ r = r$
$|\ inTimes\ r\ One = r$
$|\ inTimes\ (Times\ r\ s)\ t = Times\ r\ (inTimes\ s\ t)$
$|\ inTimes\ r\ s = Times\ r\ s$

fun *inStar* **where**
  $inStar\ Zero = One$

| *inStar Full* = *Full*
| *inStar One* = *One*
| *inStar* (*Star r*) = *Star r*
| *inStar r* = *Star r*

definition *inInter* **where**
inInter r s = (*case Inters* (*Inter r s*) *of None* ⇒ *Zero* | *Some rs* ⇒ *INTERSECT rs*)

lemma *inInter-alt*: *inInter r s* = (*let X* = *toplevel-inters* (*Inter r s*) − {*Full*} *in flatten INTERSECT* (*if Zero* ∈ *X then* {*Zero*} *else X*))
⟨*proof*⟩

fun *inNot* **where**
inNot Zero = *Full*
| *inNot Full* = *Zero*
| *inNot* (*Not r*) = *r*
| *inNot* (*Plus r s*) = *Inter* (*inNot r*) (*inNot s*)
| *inNot* (*Inter r s*) = *Plus* (*inNot r*) (*inNot s*)
| *inNot r* = *Not r*

fun *inPr* **where**
inPr Zero = *Zero*
| *inPr One* = *One*
| *inPr* (*Plus r s*) = *Plus* (*inPr r*) (*inPr s*)
| *inPr r* = *Pr r*

primrec *inorm* **where**
inorm Zero = *Zero*
| *inorm Full* = *Full*
| *inorm One* = *One*
| *inorm* (*Atom a*) = *Atom a*
| *inorm* (*Plus r s*) = *Plus* (*inorm r*) (*inorm s*)
| *inorm* (*Times r s*) = *Times* (*inorm r*) (*inorm s*)
| *inorm* (*Star r*) = *inStar* (*inorm r*)
| *inorm* (*Not r*) = *inNot* (*inorm r*)
| *inorm* (*Inter r s*) = *inInter* (*inorm r*) (*inorm s*)
| *inorm* (*Pr r*) = *inPr* (*inorm r*)

context *alphabet* **begin**

lemma *wf-inPlus*[*simp*]: ⟦*wf n r*; *wf n s*⟧ ⟹ *wf n* (*inPlus r s*)
⟨*proof*⟩

lemma *wf-inTimes*[*simp*]: ⟦*wf n r*; *wf n s*⟧ ⟹ *wf n* (*inTimes r s*)
⟨*proof*⟩

lemma *wf-inStar*[*simp*]: *wf n r* ⟹ *wf n* (*inStar r*)
⟨*proof*⟩

lemma *wf-inInter*[*simp*]: $\llbracket wf\ n\ r;\ wf\ n\ s\rrbracket \Longrightarrow wf\ n\ (inInter\ r\ s)$
⟨*proof*⟩

lemma *wf-inNot*[*simp*]: $wf\ n\ r \Longrightarrow wf\ n\ (inNot\ r)$
⟨*proof*⟩

lemma *wf-inPr*[*simp*]: $wf\ (Suc\ n)\ r \Longrightarrow wf\ n\ (inPr\ r)$
⟨*proof*⟩

lemma *wf-inorm*[*simp*]: $wf\ n\ r \Longrightarrow wf\ n\ (inorm\ r)$
⟨*proof*⟩

end

context *project* **begin**

lemma *lang-inPlus*[*simp*]: $\llbracket wf\ n\ r;\ wf\ n\ s\rrbracket \Longrightarrow lang\ n\ (inPlus\ r\ s) = lang\ n\ (Plus\ r\ s)$
⟨*proof*⟩

lemma *lang-inTimes*[*simp*]: $\llbracket wf\ n\ r;\ wf\ n\ s\rrbracket \Longrightarrow lang\ n\ (inTimes\ r\ s) = lang\ n\ (Times\ r\ s)$
⟨*proof*⟩

lemma *lang-inStar*[*simp*]: $wf\ n\ r \Longrightarrow lang\ n\ (inStar\ r) = lang\ n\ (Star\ r)$
⟨*proof*⟩

lemma *Zero-toplevel-inters*[*dest*]: $Zero \in toplevel\text{-}inters\ r \Longrightarrow lang\ n\ r = \{\}$
⟨*proof*⟩

lemma *toplevel-inters-Full*: $\llbracket toplevel\text{-}inters\ r = \{Full\};\ wf\ n\ r\rrbracket \Longrightarrow lang\ n\ r = lists\ (\Sigma\ n)$
⟨*proof*⟩

lemma *toplevel-inters-subset-singleton*[*simp*]: $toplevel\text{-}inters\ r \subseteq \{s\} \longleftrightarrow toplevel\text{-}inters\ r = \{s\}$
⟨*proof*⟩

lemma *lang-inInter*[*simp*]: $\llbracket wf\ n\ r;\ wf\ n\ s\rrbracket \Longrightarrow lang\ n\ (inInter\ r\ s) = lang\ n\ (Inter\ r\ s)$
⟨*proof*⟩

lemma *lang-inNot*[*simp*]: $wf\ n\ r \Longrightarrow lang\ n\ (inNot\ r) = lang\ n\ (Not\ r)$
⟨*proof*⟩

lemma *lang-inPr*[*simp*]: $wf\ (Suc\ n)\ r \Longrightarrow lang\ n\ (inPr\ r) = lang\ n\ (Pr\ r)$
⟨*proof*⟩

lemma *lang-inorm*[*simp*]: $wf\ n\ r \Longrightarrow lang\ n\ (inorm\ r) = lang\ n\ r$
⟨*proof*⟩

end

8 Partial Derivatives-like Normalization

fun *pnPlus* :: $'a::linorder\ rexp \Rightarrow 'a\ rexp \Rightarrow 'a\ rexp$ **where**
$pnPlus\ Zero\ r = r$
$|\ pnPlus\ r\ Zero = r|\ pnPlus\ (Plus\ r\ s)\ t = pnPlus\ r\ (pnPlus\ s\ t)$
$|\ pnPlus\ r\ (Plus\ s\ t) =$
$(if\ r = s\ then\ (Plus\ s\ t)$
$else\ if\ r \leq s\ then\ Plus\ r\ (Plus\ s\ t)$
$else\ Plus\ s\ (pnPlus\ r\ t))$
$|\ pnPlus\ r\ s =$
$(if\ r = s\ then\ r$
$else\ if\ r \leq s\ then\ Plus\ r\ s$
$else\ Plus\ s\ r)$

lemma (**in** *alphabet*) *wf-pnPlus*[*simp*]: $[\![wf\ n\ r;\ wf\ n\ s]\!] \Longrightarrow wf\ n\ (pnPlus\ r\ s)$
⟨*proof*⟩

lemma (**in** *project*) *lang-pnPlus*[*simp*]: $[\![wf\ n\ r;\ wf\ n\ s]\!] \Longrightarrow lang\ n\ (pnPlus\ r\ s) = lang\ n\ (Plus\ r\ s)$
⟨*proof*⟩

fun *pnTimes* :: $'a::linorder\ rexp \Rightarrow 'a\ rexp \Rightarrow 'a\ rexp$ **where**
$pnTimes\ Zero\ r = Zero$
$|\ pnTimes\ One\ r = r$
$|\ pnTimes\ (Plus\ r\ s)\ t = pnPlus\ (pnTimes\ r\ t)\ (pnTimes\ s\ t)$
$|\ pnTimes\ r\ s = Times\ r\ s$

lemma (**in** *alphabet*) *wf-pnTimes*[*simp*]: $[\![wf\ n\ r;\ wf\ n\ s]\!] \Longrightarrow wf\ n\ (pnTimes\ r\ s)$
⟨*proof*⟩

lemma (**in** *project*) *lang-pnTimes*[*simp*]: $[\![wf\ n\ r;\ wf\ n\ s]\!] \Longrightarrow lang\ n\ (pnTimes\ r\ s) = lang\ n\ (Times\ r\ s)$
⟨*proof*⟩

fun *pnInter* :: $'a::linorder\ rexp \Rightarrow 'a\ rexp \Rightarrow 'a\ rexp$ **where**
$pnInter\ Zero\ r = Zero$
$|\ pnInter\ r\ Zero = Zero$
$|\ pnInter\ Full\ r = r$
$|\ pnInter\ r\ Full = r$
$|\ pnInter\ (Plus\ r\ s)\ t = pnPlus\ (pnInter\ r\ t)\ (pnInter\ s\ t)$
$|\ pnInter\ r\ (Plus\ s\ t) = pnPlus\ (pnInter\ r\ s)\ (pnInter\ r\ t)$
$|\ pnInter\ (Inter\ r\ s)\ t = pnInter\ r\ (pnInter\ s\ t)$

| *pnInter r* (*Inter s t*) =
  (*if r* = *s then Inter s t*
  else if r ≤ *s then Inter r* (*Inter s t*)
  else Inter s (*pnInter r t*))
| *pnInter r s* =
  (*if r* = *s then s*
  else if r ≤ *s then Inter r s*
  else Inter s r)

lemma (**in** *alphabet*) *wf-pnInter*[*simp*]: ⟦*wf n r*; *wf n s*⟧ ⟹ *wf n* (*pnInter r s*)
  ⟨*proof*⟩

lemma (**in** *project*) *lang-pnInter*[*simp*]: ⟦*wf n r*; *wf n s*⟧ ⟹ *lang n* (*pnInter r s*) = *lang n* (*Inter r s*)
  ⟨*proof*⟩

fun *pnNot* :: ′*a*::*linorder rexp* ⇒ ′*a rexp* **where**
  pnNot (*Plus r s*) = *pnInter* (*pnNot r*) (*pnNot s*)
| *pnNot* (*Inter r s*) = *pnPlus* (*pnNot r*) (*pnNot s*)
| *pnNot Full* = *Zero*
| *pnNot Zero* = *Full*
| *pnNot* (*Not r*) = *r*
| *pnNot r* = *Not r*

lemma (**in** *alphabet*) *wf-pnNot*[*simp*]: *wf n r* ⟹ *wf n* (*pnNot r*)
  ⟨*proof*⟩

lemma (**in** *project*) *lang-pnNot*[*simp*]: *wf n r* ⟹ *lang n* (*pnNot r*) = *lang n* (*Not r*)
  ⟨*proof*⟩

fun *pnPr* :: ′*a*::*linorder rexp* ⇒ ′*a rexp* **where**
  pnPr Zero = *Zero*
| *pnPr One* = *One*
| *pnPr* (*Plus r s*) = *pnPlus* (*pnPr r*) (*pnPr s*)
| *pnPr r* = *Pr r*

lemma (**in** *alphabet*) *wf-pnPr*[*simp*]: *wf* (*Suc n*) *r* ⟹ *wf n* (*pnPr r*)
  ⟨*proof*⟩

lemma (**in** *project*) *lang-pnPr*[*simp*]: *wf* (*Suc n*) *r* ⟹ *lang n* (*pnPr r*) = *lang n* (*Pr r*)
  ⟨*proof*⟩

primrec *pnorm* :: ′*a*::*linorder rexp* ⇒ ′*a rexp* **where**
  pnorm Zero = *Zero*
| *pnorm Full* = *Full*
| *pnorm One* = *One*
| *pnorm* (*Atom a*) = *Atom a*

| $pnorm\ (Plus\ r\ s) = pnPlus\ (pnorm\ r)\ (pnorm\ s)$
| $pnorm\ (Times\ r\ s) = pnTimes\ (pnorm\ r)\ s$
| $pnorm\ (Star\ r) = Star\ r$
| $pnorm\ (Inter\ r\ s) = pnInter\ (pnorm\ r)\ (pnorm\ s)$
| $pnorm\ (Not\ r) = pnNot\ (pnorm\ r)$
| $pnorm\ (Pr\ r) = pnPr\ (pnorm\ r)$

lemma (**in** *alphabet*) *wf-pnorm*[*simp*]: $wf\ n\ r \Longrightarrow wf\ n\ (pnorm\ r)$
⟨*proof*⟩

lemma (**in** *project*) *lang-pnorm*[*simp*]: $wf\ n\ r \Longrightarrow lang\ n\ (pnorm\ r) = lang\ n\ r$
⟨*proof*⟩

9 Monadic Second-Order Logic Formulas

9.1 Interpretations and Encodings

type-synonym $'a\ interp = 'a\ list \times (nat + nat\ set)\ list$

abbreviation $enc\text{-}atom\text{-}bool\ I\ n \equiv map\ (\lambda x.\ case\ x\ of\ Inl\ p \Rightarrow n = p \mid Inr\ P \Rightarrow n \in P)\ I$

abbreviation $enc\text{-}atom\ I\ n\ a \equiv (a,\ enc\text{-}atom\text{-}bool\ I\ n)$

9.2 Syntax and Semantics of MSO

datatype *'a formula* =
FQ 'a nat
| *FLess nat nat*
| *FIn nat nat*
| *FNot 'a formula*
| *FOr 'a formula 'a formula*
| *FAnd 'a formula 'a formula*
| *FExists 'a formula*
| *FEXISTS 'a formula*

primrec *FOV* :: *'a formula* ⇒ *nat set* **where**
$FOV\ (FQ\ a\ m) = \{m\}$
| $FOV\ (FLess\ m1\ m2) = \{m1,\ m2\}$
| $FOV\ (FIn\ m\ M) = \{m\}$
| $FOV\ (FNot\ \varphi) = FOV\ \varphi$
| $FOV\ (FOr\ \varphi_1\ \varphi_2) = FOV\ \varphi_1 \cup FOV\ \varphi_2$
| $FOV\ (FAnd\ \varphi_1\ \varphi_2) = FOV\ \varphi_1 \cup FOV\ \varphi_2$
| $FOV\ (FExists\ \varphi) = (\lambda x.\ x - 1)\ `\ (FOV\ \varphi - \{0\})$
| $FOV\ (FEXISTS\ \varphi) = (\lambda x.\ x - 1)\ `\ FOV\ \varphi$

primrec *SOV* :: *'a formula* ⇒ *nat set* **where**
$SOV\ (FQ\ a\ m) = \{\}$
| $SOV\ (FLess\ m1\ m2) = \{\}$

| SOV $(FIn\ m\ M) = \{M\}$
| SOV $(FNot\ \varphi) = SOV\ \varphi$
| SOV $(FOr\ \varphi_1\ \varphi_2) = SOV\ \varphi_1 \cup SOV\ \varphi_2$
| SOV $(FAnd\ \varphi_1\ \varphi_2) = SOV\ \varphi_1 \cup SOV\ \varphi_2$
| SOV $(FExists\ \varphi) = (\lambda x.\ x - 1)$ ‘ $SOV\ \varphi$
| SOV $(FEXISTS\ \varphi) = (\lambda x.\ x - 1)$ ‘ $(SOV\ \varphi - \{0\})$

definition $\sigma = (\lambda\Sigma\ n.\ concat\ (map\ (\lambda bs.\ map\ (\lambda a.\ (a,\ bs))\ \Sigma)\ (List.n\text{-}lists\ n\ [True,\ False])))$
definition $\pi = (\lambda(a,\ bs).\ (a,\ tl\ bs))$
definition $\varepsilon = (\lambda\Sigma\ (a{::}'a,\ bs).\ if\ a \in set\ \Sigma\ then\ [(a,\ True\ \#\ bs),\ (a,\ False\ \#\ bs)]\ else\ [])$

datatype *'a atom* =
  Singleton 'a bool list
  | *AQ nat 'a*
  | *Arbitrary-Except nat bool*
  | *Arbitrary-Except2 nat nat*
derive *linorder atom*

fun *wf-atom* **where**
  $wf\text{-}atom\ \Sigma\ n\ (Singleton\ a\ bs) = (a \in set\ \Sigma \wedge length\ bs = n)$
| $wf\text{-}atom\ \Sigma\ n\ (AQ\ m\ a) = (a \in set\ \Sigma \wedge m < n)$
| $wf\text{-}atom\ \Sigma\ n\ (Arbitrary\text{-}Except\ m\ \text{-}) = (m < n)$
| $wf\text{-}atom\ \Sigma\ n\ (Arbitrary\text{-}Except2\ m1\ m2) = (m1 < n \wedge m2 < n)$

fun *lookup* **where**
  $lookup\ (Singleton\ a'\ bs')\ (a,\ bs) = (a = a' \wedge bs = bs')$
| $lookup\ (AQ\ m\ a')\ (a,\ bs) = (a = a' \wedge bs\ !\ m)$
| $lookup\ (Arbitrary\text{-}Except\ m\ b)\ (\text{-},\ bs) = (bs\ !\ m = b)$
| $lookup\ (Arbitrary\text{-}Except2\ m1\ m2)\ (\text{-},\ bs) = (bs\ !\ m1 \wedge bs\ !\ m2)$

lemma $\pi\text{-}\sigma$: π ‘ $(set\ o\ \sigma\ \Sigma)\ (n + 1) = (set\ o\ \sigma\ \Sigma)\ n$
  $\langle proof \rangle$

locale $formula = embed2\ set\ o\ (\sigma\ \Sigma)\ wf\text{-}atom\ \Sigma\ \pi\ lookup\ \varepsilon\ \Sigma\ case\text{-}prod\ Singleton$
for $\Sigma :: 'a :: linorder\ list +$
assumes *nonempty*: $\Sigma \neq []$
begin

abbreviation $\Sigma\text{-}product\text{-}lists\ n \equiv$
  $List.maps\ (\lambda bools.\ map\ (\lambda a.\ (a,\ bools))\ \Sigma)\ (bool\text{-}product\text{-}lists\ n)$

primrec $pre\text{-}wf\text{-}formula :: nat \Rightarrow 'a\ formula \Rightarrow bool$ **where**
  $pre\text{-}wf\text{-}formula\ n\ (FQ\ a\ m) = (a \in set\ \Sigma \wedge m < n)$
| $pre\text{-}wf\text{-}formula\ n\ (FLess\ m1\ m2) = (m1 < n \wedge m2 < n)$
| $pre\text{-}wf\text{-}formula\ n\ (FIn\ m\ M) = (m < n \wedge M < n)$
| $pre\text{-}wf\text{-}formula\ n\ (FNot\ \varphi) = pre\text{-}wf\text{-}formula\ n\ \varphi$

| *pre-wf-formula n* ($FOr\ \varphi_1\ \varphi_2$) = (*pre-wf-formula n* φ_1 ∧ *pre-wf-formula n* φ_2)
| *pre-wf-formula n* ($FAnd\ \varphi_1\ \varphi_2$) = (*pre-wf-formula n* φ_1 ∧ *pre-wf-formula n* φ_2)
| *pre-wf-formula n* (*FExists* φ) = (*pre-wf-formula* (*n* + *1*) φ ∧ *0* ∈ *FOV* φ ∧ *0* ∉ *SOV* φ)
| *pre-wf-formula n* (*FEXISTS* φ) = (*pre-wf-formula* (*n* + *1*) φ ∧ *0* ∉ *FOV* φ ∧ *0* ∈ *SOV* φ)

abbreviation *closed* ≡ *pre-wf-formula 0*

definition [*simp*]: *wf-formula n* φ ≡ *pre-wf-formula n* φ ∧ *FOV* φ ∩ *SOV* φ = {}

lemma *max-idx-vars*: *pre-wf-formula n* φ ⟹ ∀ *p* ∈ *FOV* φ ∪ *SOV* φ. *p* < *n*
⟨*proof*⟩

lemma *finite-FOV*: *finite* (*FOV* φ)
⟨*proof*⟩

9.3 ENC

definition *valid-ENC* :: *nat* ⇒ *nat* ⇒ (*'a atom*) *rexp* **where**
valid-ENC n p = (*if n* = *0 then Full else*
TIMES [
Star (*Atom* (*Arbitrary-Except p False*)),
Atom (*Arbitrary-Except p True*),
Star (*Atom* (*Arbitrary-Except p False*))])

lemma *wf-rexp-valid-ENC*: *n* = *0* ∨ *p* < *n* ⟹ *wf n* (*valid-ENC n p*)
⟨*proof*⟩

definition *ENC* :: *nat* ⇒ *nat set* ⇒ (*'a atom*) *rexp* **where**
ENC n V = *flatten INTERSECT* (*valid-ENC n* ‘ *V*)

lemma *wf-rexp-ENC*: ⟦*finite V*; *n* = *0* ∨ (∀ *v* ∈ *V*. *v* < *n*)⟧ ⟹ *wf n* (*ENC n V*)
⟨*proof*⟩

lemma *enc-atom-σ-eq*: *i* < *length w* ⟹
(*length I* = *n* ∧ *p* ∈ *set* Σ) ⟷ *enc-atom I i p* ∈ *set* (σ Σ *n*)
⟨*proof*⟩

lemmas *enc-atom-σ* = *iffD1*[*OF enc-atom-σ-eq*, *OF - conjI*]

lemma *enc-atom-bool-take-drop-True*:
⟦*r* < *length I*; *case I* ! *r of Inl p'* ⇒ *p* = *p'* | *Inr P* ⇒ *p* ∈ *P*⟧ ⟹
enc-atom-bool I p = *take r* (*enc-atom-bool I p*) @ *True* # *drop* (*Suc r*) (*enc-atom-bool I p*)
⟨*proof*⟩

lemma *enc-atom-bool-take-drop-True2*:
⟦*r* < *length I*; *case I* ! *r of Inl p'* ⇒ *p* = *p'* | *Inr P* ⇒ *p* ∈ *P*;

$s < length\ I$; $case\ I\ !\ s\ of\ Inl\ p' \Rightarrow p = p' \mid Inr\ P \Rightarrow p \in P$; $r < s]\!] \Longrightarrow$
$enc\text{-}atom\text{-}bool\ I\ p = take\ r\ (enc\text{-}atom\text{-}bool\ I\ p)\ @\ True\ \#$
$take\ (s - Suc\ r)\ (drop\ (Suc\ r)\ (enc\text{-}atom\text{-}bool\ I\ p))\ @\ True\ \#$
$drop\ (Suc\ s)\ (enc\text{-}atom\text{-}bool\ I\ p)$
⟨*proof*⟩

lemma *enc-atom-bool-take-drop-False*:
$[\![r < length\ I;\ case\ I\ !\ r\ of\ Inl\ p' \Rightarrow p \neq p' \mid Inr\ P \Rightarrow p \notin P]\!] \Longrightarrow$
$enc\text{-}atom\text{-}bool\ I\ p = take\ r\ (enc\text{-}atom\text{-}bool\ I\ p)\ @\ False\ \#\ drop\ (Suc\ r)$
$(enc\text{-}atom\text{-}bool\ I\ p)$
⟨*proof*⟩

lemma *enc-atom-lang-AQ*: $[\![r < length\ I;$
$case\ I\ !\ r\ of\ Inl\ p' \Rightarrow p = p' \mid Inr\ P \Rightarrow p \in P;\ length\ I = n;\ a \in set\ \Sigma]\!] \Longrightarrow$
$[enc\text{-}atom\ I\ p\ a] \in lang\ n\ (Atom\ (AQ\ r\ a))$
⟨*proof*⟩

lemma *enc-atom-lang-Arbitrary-Except-True*: $[\![r < length\ I;$
$case\ I\ !\ r\ of\ Inl\ p' \Rightarrow p = p' \mid Inr\ P \Rightarrow p \in P;\ length\ I = n;\ a \in set\ \Sigma]\!] \Longrightarrow$
$[enc\text{-}atom\ I\ p\ a] \in lang\ n\ (Atom\ (Arbitrary\text{-}Except\ r\ True))$
⟨*proof*⟩

lemma *enc-atom-lang-Arbitrary-Except2*: $[\![r < length\ I;\ s < length\ I;$
$case\ I\ !\ r\ of\ Inl\ p' \Rightarrow p = p' \mid Inr\ P \Rightarrow p \in P;$
$case\ I\ !\ s\ of\ Inl\ p' \Rightarrow p = p' \mid Inr\ P \Rightarrow p \in P;\ length\ I = n;\ a \in set\ \Sigma]\!] \Longrightarrow$
$[enc\text{-}atom\ I\ p\ a] \in lang\ n\ (Atom\ (Arbitrary\text{-}Except2\ r\ s))$
⟨*proof*⟩

lemma *enc-atom-lang-Arbitrary-Except-False*: $[\![r < length\ I;$
$case\ I\ !\ r\ of\ Inl\ p' \Rightarrow p \neq p' \mid Inr\ P \Rightarrow p \notin P;\ length\ I = n;\ a \in set\ \Sigma]\!] \Longrightarrow$
$[enc\text{-}atom\ I\ p\ a] \in lang\ n\ (Atom\ (Arbitrary\text{-}Except\ r\ False))$
⟨*proof*⟩

lemma *AQ-D*:
assumes $v \in lang\ n\ (Atom\ (AQ\ m\ a))\ m < n\ a \in set\ \Sigma$
shows $\exists x.\ v = [x] \wedge fst\ x = a \wedge snd\ x\ !\ m$
⟨*proof*⟩

lemma *Arbitrary-ExceptD*:
assumes $v \in lang\ n\ (Atom\ (Arbitrary\text{-}Except\ r\ b))\ r < n$
shows $\exists x.\ v = [x] \wedge snd\ x\ !\ r = b$
⟨*proof*⟩

lemma *Arbitrary-Except2D*:
assumes $v \in lang\ n\ (Atom\ (Arbitrary\text{-}Except2\ r\ s))\ r < n\ s < n$
shows $\exists x.\ v = [x] \wedge snd\ x\ !\ r \wedge snd\ x\ !\ s$
⟨*proof*⟩

lemma *star-Arbitrary-ExceptD*:

$[\![v \in \textit{star}\ (\textit{lang}\ n\ (\textit{Atom}\ (\textit{Arbitrary-Except}\ r\ b))); r < n; i < \textit{length}\ v]\!] \Longrightarrow$
$\textit{snd}\ (v\ !\ i)\ !\ r = b$
$\langle \textit{proof} \rangle$

end

end

10 M2L

10.1 Encodings

context *formula*
begin

fun $\textit{enc} :: \textit{'a interp} \Rightarrow (\textit{'a} \times \textit{bool list})\ \textit{list}$ **where**
$\textit{enc}\ (w, I) = \textit{map-index}\ (\textit{enc-atom}\ I)\ w$

abbreviation $\textit{wf-interp}\ w\ I \equiv (\textit{length}\ w > 0 \wedge$
$(\forall a \in \textit{set}\ w.\ a \in \textit{set}\ \Sigma) \wedge$
$(\forall x \in \textit{set}\ I.\ \textit{case}\ x\ \textit{of}\ \textit{Inl}\ p \Rightarrow p < \textit{length}\ w \mid \textit{Inr}\ P \Rightarrow \forall p \in P.\ p < \textit{length}\ w))$

fun $\textit{wf-interp-for-formula} :: \textit{'a interp} \Rightarrow \textit{'a formula} \Rightarrow \textit{bool}$ **where**
$\textit{wf-interp-for-formula}\ (w, I)\ \varphi =$
$(\textit{wf-interp}\ w\ I \wedge$
$(\forall n \in \textit{FOV}\ \varphi.\ \textit{case}\ I\ !\ n\ \textit{of}\ \textit{Inl}\ \text{-} \Rightarrow \textit{True} \mid \text{-} \Rightarrow \textit{False}) \wedge$
$(\forall n \in \textit{SOV}\ \varphi.\ \textit{case}\ I\ !\ n\ \textit{of}\ \textit{Inl}\ \text{-} \Rightarrow \textit{False} \mid \text{-} \Rightarrow \textit{True}))$

fun $\textit{satisfies} :: \textit{'a interp} \Rightarrow \textit{'a formula} \Rightarrow \textit{bool}$ (**infix** $\langle\models\rangle$ *50*) **where**
$(w, I) \models \textit{FQ}\ a\ m = (w\ !\ (\textit{case}\ I\ !\ m\ \textit{of}\ \textit{Inl}\ p \Rightarrow p) = a)$
$\mid (w, I) \models \textit{FLess}\ m1\ m2 = ((\textit{case}\ I\ !\ m1\ \textit{of}\ \textit{Inl}\ p \Rightarrow p) < (\textit{case}\ I\ !\ m2\ \textit{of}\ \textit{Inl}\ p \Rightarrow p))$
$\mid (w, I) \models \textit{FIn}\ m\ M = ((\textit{case}\ I\ !\ m\ \textit{of}\ \textit{Inl}\ p \Rightarrow p) \in (\textit{case}\ I\ !\ M\ \textit{of}\ \textit{Inr}\ P \Rightarrow P))$
$\mid (w, I) \models \textit{FNot}\ \varphi = (\neg\ (w, I) \models \varphi)$
$\mid (w, I) \models (\textit{FOr}\ \varphi_1\ \varphi_2) = ((w, I) \models \varphi_1 \vee (w, I) \models \varphi_2)$
$\mid (w, I) \models (\textit{FAnd}\ \varphi_1\ \varphi_2) = ((w, I) \models \varphi_1 \wedge (w, I) \models \varphi_2)$
$\mid (w, I) \models (\textit{FExists}\ \varphi) = (\exists p.\ p \in \{0\ ..\ \textit{length}\ w - 1\} \wedge (w, \textit{Inl}\ p\ \#\ I) \models \varphi)$
$\mid (w, I) \models (\textit{FEXISTS}\ \varphi) = (\exists P.\ P \subseteq \{0\ ..\ \textit{length}\ w - 1\} \wedge (w, \textit{Inr}\ P\ \#\ I) \models \varphi)$

definition $\textit{lang}_{M2L} :: \textit{nat} \Rightarrow \textit{'a formula} \Rightarrow (\textit{'a} \times \textit{bool list})\ \textit{list set}$ **where**
$\textit{lang}_{M2L}\ n\ \varphi = \{\textit{enc}\ (w, I) \mid w\ I.$
$\textit{length}\ I = n \wedge \textit{wf-interp-for-formula}\ (w, I)\ \varphi \wedge \textit{satisfies}\ (w, I)\ \varphi\}$

definition $\textit{dec-word} \equiv \textit{map fst}$

definition $\textit{positions-in-row}\ w\ i =$
$\textit{Option.these}\ (\textit{set}\ (\textit{map-index}\ (\lambda p\ \textit{a-bs}.\ \textit{if nth}\ (\textit{snd a-bs})\ i\ \textit{then Some}\ p\ \textit{else None})\ w))$

definition $\mathit{dec\text{-}interp}\ n\ \mathit{FO}\ (w :: ('a \times \mathit{bool\ list})\ \mathit{list}) \equiv \mathit{map}\ (\lambda i.$
$\mathit{if}\ i \in \mathit{FO}$
$\mathit{then}\ \mathit{Inl}\ (\mathit{the\text{-}elem}\ (\mathit{positions\text{-}in\text{-}row}\ w\ i))$
$\mathit{else}\ \mathit{Inr}\ (\mathit{positions\text{-}in\text{-}row}\ w\ i))\ [0..{<}n]$

lemma *positions-in-row*: $\mathit{positions\text{-}in\text{-}row}\ w\ i = \{p.\ p < \mathit{length}\ w \wedge \mathit{snd}\ (w\ !\ p)\ !\ i\}$
⟨*proof*⟩

lemma *positions-in-row-unique*: $\exists!p.\ p < \mathit{length}\ w \wedge \mathit{snd}\ (w\ !\ p)\ !\ i \Longrightarrow$
$\mathit{the\text{-}elem}\ (\mathit{positions\text{-}in\text{-}row}\ w\ i) = (\mathit{THE}\ p.\ p < \mathit{length}\ w \wedge \mathit{snd}\ (w\ !\ p)\ !\ i)$
⟨*proof*⟩

lemma *positions-in-row-length*: $\exists!p.\ p < \mathit{length}\ w \wedge \mathit{snd}\ (w\ !\ p)\ !\ i \Longrightarrow$
$\mathit{the\text{-}elem}\ (\mathit{positions\text{-}in\text{-}row}\ w\ i) < \mathit{length}\ w$
⟨*proof*⟩

lemma *dec-interp-Inl*: $[\![i \in \mathit{FO};\ i < n]\!] \Longrightarrow \exists p.\ \mathit{dec\text{-}interp}\ n\ \mathit{FO}\ x\ !\ i = \mathit{Inl}\ p$
⟨*proof*⟩

lemma *dec-interp-not-Inr*: $[\![\mathit{dec\text{-}interp}\ n\ \mathit{FO}\ x\ !\ i = \mathit{Inr}\ P;\ i \in \mathit{FO};\ i < n]\!] \Longrightarrow \mathit{False}$
⟨*proof*⟩

lemma *dec-interp-Inr*: $[\![i \notin \mathit{FO};\ i < n]\!] \Longrightarrow \exists P.\ \mathit{dec\text{-}interp}\ n\ \mathit{FO}\ x\ !\ i = \mathit{Inr}\ P$
⟨*proof*⟩

lemma *dec-interp-not-Inl*: $[\![\mathit{dec\text{-}interp}\ n\ \mathit{FO}\ x\ !\ i = \mathit{Inl}\ p;\ i \notin \mathit{FO};\ i < n]\!] \Longrightarrow \mathit{False}$
⟨*proof*⟩

lemma *Inl-dec-interp-length*:
assumes $\forall i \in \mathit{FO}.\ \exists!p.\ p < \mathit{length}\ w \wedge \mathit{snd}\ (w\ !\ p)\ !\ i$
shows $\mathit{Inl}\ p \in \mathit{set}\ (\mathit{dec\text{-}interp}\ n\ \mathit{FO}\ w) \Longrightarrow p < \mathit{length}\ w$
⟨*proof*⟩

lemma *Inr-dec-interp-length*: $[\![\mathit{Inr}\ P \in \mathit{set}\ (\mathit{dec\text{-}interp}\ n\ \mathit{FO}\ w);\ p \in P]\!] \Longrightarrow p < \mathit{length}\ w$
⟨*proof*⟩

lemma *the-elem-Collect*[*simp*]:
assumes $\exists!x.\ P\ x$
shows $\mathit{the\text{-}elem}\ (\mathit{Collect}\ P) = (\mathit{The}\ P)$
⟨*proof*⟩

lemma *enc-atom-dec*:
$[\![\mathit{wf\text{-}word}\ n\ w;\ \forall i \in \mathit{FO}.\ i < n \longrightarrow (\exists!p.\ p < \mathit{length}\ w \wedge \mathit{snd}\ (w\ !\ p)\ !\ i);\ p < \mathit{length}\ w]\!] \Longrightarrow$

enc-atom (*dec-interp n FO w*) *p* (*fst* (*w* ! *p*)) = *w* ! *p*
⟨*proof*⟩

lemma *enc-dec*:
⟦*wf-word n w*; ∀ *i* ∈ *FO*. *i* < *n* ⟶ (∃!*p*. *p* < *length w* ∧ *snd* (*w* ! *p*) ! *i*)⟧ ⟹
enc (*dec-word w*, *dec-interp n FO w*) = *w*
⟨*proof*⟩

lemma *dec-word-enc*: *dec-word* (*enc* (*w*, *I*)) = *w*
⟨*proof*⟩

lemma *enc-unique*:
assumes *wf-interp w I i* < *length I*
shows ∃ *p*. *I* ! *i* = *Inl p* ⟹ ∃!*p*. *p* < *length* (*enc* (*w*, *I*)) ∧ *snd* (*enc* (*w*, *I*) ! *p*) ! *i*
⟨*proof*⟩

lemma *dec-interp-enc-Inl*:
⟦*dec-interp n FO* (*enc* (*w*, *I*)) ! *i* = *Inl p*′; *I* ! *i* = *Inl p*; *i* ∈ *FO*; *i* < *n*; *length I* = *n*; *p* < *length w*; *wf-interp w I*⟧ ⟹
p = *p*′
⟨*proof*⟩

lemma *dec-interp-enc-Inr*:
⟦*dec-interp n FO* (*enc* (*w*, *I*)) ! *i* = *Inr P*′; *I* ! *i* = *Inr P*; *i* ∉ *FO*; *i* < *n*; *length I* = *n*; ∀ *p* ∈ *P*. *p* < *length w*⟧ ⟹
P = *P*′
⟨*proof*⟩

lemma *length-dec-interp*[*simp*]: *length* (*dec-interp n FO x*) = *n*
⟨*proof*⟩

lemma *nth-dec-interp*[*simp*]: *i* < *n* ⟹ *dec-interp n* {} *x* ! *i* = *Inr* (*positions-in-row x i*)
⟨*proof*⟩

lemma *set-σD*[*simp*]: (*a*, *bs*) ∈ *set* (σ Σ *n*) ⟹ *a* ∈ *set* Σ
⟨*proof*⟩

lemma *lang-ENC*:
assumes *FO* ⊆ {*0* ..< *n*} *SO* ⊆ {*0* ..< *n*} − *FO*
shows *lang n* (*ENC n FO*) − {[]} = {*enc* (*w*, *I*) | *w I* . *length I* = *n* ∧ *wf-interp w I* ∧
(∀ *i* ∈ *FO*. *case I* ! *i of Inl* - ⇒ *True* | *Inr* - ⇒ *False*) ∧
(∀ *i* ∈ *SO*. *case I* ! *i of Inl* - ⇒ *False* | *Inr* - ⇒ *True*)}
(**is** *?L* = *?R*)
⟨*proof*⟩

lemma *lang-ENC-formula*:

assumes *wf-formula n* φ
shows *lang n* (*ENC n* (*FOV* φ)) $-$ {[]} $=$ {*enc* (*w*, *I*) | *w I* . *length I* $=$ *n* $\wedge$ *wf-interp-for-formula* (*w*, *I*) φ}
⟨*proof*⟩

10.2 Welldefinedness of enc wrt. Models

lemma *enc-alt-def*:
enc (*w*, *x* # *I*) $=$ *map-index* (λ*n* (*a*, *bs*). (*a*, (*case x of Inl p* $\Rightarrow$ *n* $=$ *p* | *Inr P* $\Rightarrow$ *n* $\in$ *P*) # *bs*)) (*enc* (*w*, *I*))
⟨*proof*⟩

lemma *enc-extend-interp*: *enc* (*w*, *I*) $=$ *enc* (*w'*, *I'*) $\Longrightarrow$ *enc* (*w*, *x* # *I*) $=$ *enc* (*w'*, *x* # *I'*)
⟨*proof*⟩

lemma *wf-interp-for-formula-FExists*:
⟦*wf-formula* (*length I*) (*FExists* φ); *w* $\neq$ []⟧$\Longrightarrow$
wf-interp-for-formula (*w*, *I*) (*FExists* φ) $\longleftrightarrow$
($\forall$ *p* $<$ *length w*. *wf-interp-for-formula* (*w*, *Inl p* # *I*) φ)
⟨*proof*⟩

lemma *wf-interp-for-formula-any-Inl*: *wf-interp-for-formula* (*w*, *Inl p* # *I*) φ $\Longrightarrow$
$\forall$ *p* $<$ *length w*. *wf-interp-for-formula* (*w*, *Inl p* # *I*) φ
⟨*proof*⟩

lemma *wf-interp-for-formula-FEXISTS*:
⟦*wf-formula* (*length I*) (*FEXISTS* φ); *w* $\neq$ []⟧$\Longrightarrow$
wf-interp-for-formula (*w*, *I*) (*FEXISTS* φ) $\longleftrightarrow$ ($\forall$ *P* $\subseteq$ {*0* .. *length w* $-$ *1*}. *wf-interp-for-formula* (*w*, *Inr P* # *I*) φ)
⟨*proof*⟩

lemma *wf-interp-for-formula-any-Inr*: *wf-interp-for-formula* (*w*, *Inr P* # *I*) φ $\Longrightarrow$
$\forall$ *P* $\subseteq$ {*0* .. *length w* $-$ *1*}. *wf-interp-for-formula* (*w*, *Inr P* # *I*) φ
⟨*proof*⟩

lemma *enc-word-length*: *enc* (*w*, *I*) $=$ *enc* (*w'*, *I'*) $\Longrightarrow$ *length w* $=$ *length w'*
⟨*proof*⟩

lemma *enc-length*:
assumes *w* $\neq$ [] *enc* (*w*, *I*) $=$ *enc* (*w'*, *I'*)
shows *length I* $=$ *length I'*
⟨*proof*⟩

lemma *wf-interp-for-formula-FOr*:
wf-interp-for-formula (*w*, *I*) (*FOr* φ*1* φ*2*) $=$
(*wf-interp-for-formula* (*w*, *I*) φ*1* $\wedge$ *wf-interp-for-formula* (*w*, *I*) φ*2*)
⟨*proof*⟩

lemma *wf-interp-for-formula-FAnd*:
wf-interp-for-formula $(w,\ I)$ $(FAnd\ \varphi 1\ \varphi 2) =$
$(wf\text{-}interp\text{-}for\text{-}formula\ (w,\ I)\ \varphi 1 \wedge wf\text{-}interp\text{-}for\text{-}formula\ (w,\ I)\ \varphi 2)$
⟨*proof*⟩

lemma *enc-wf-interp*:
assumes *wf-formula* (*length* I) φ *wf-interp-for-formula* $(w,\ I)$ φ
shows *wf-interp-for-formula* (*dec-word* (*enc* $(w,\ I)$), *dec-interp* (*length* I) (*FOV* φ) (*enc* $(w,\ I)$)) φ
(**is** *wf-interp-for-formula* (-, *?dec*) φ)
⟨*proof*⟩

lemma *enc-welldef*: ⟦*enc* $(w,\ I)$ = *enc* $(w',\ I')$; *wf-formula* (*length* I) φ;
wf-interp-for-formula $(w,\ I)$ φ; *wf-interp-for-formula* $(w',\ I')$ φ⟧ $\Longrightarrow$
satisfies $(w,\ I)$ $\varphi \longleftrightarrow$ *satisfies* $(w',\ I')$ φ
⟨*proof*⟩

lemma $lang_{M2L}$*-FOr*:
assumes *wf-formula* n (*FOr* φ_1 φ_2)
shows $lang_{M2L}$ n (*FOr* φ_1 φ_2) $\subseteq$
$(lang_{M2L}\ n\ \varphi_1 \cup lang_{M2L}\ n\ \varphi_2) \cap \{enc\ (w,\ I)\ \mid\ w\ I.\ length\ I = n \wedge$ *wf-interp-for-formula* $(w,\ I)$ (*FOr* φ_1 φ_2)$\}$
(**is** - $\subseteq$ (*?L1* $\cup$ *?L2*) $\cap$ *?ENC*)
⟨*proof*⟩

lemma $lang_{M2L}$*-FAnd*:
assumes *wf-formula* n (*FAnd* φ_1 φ_2)
shows $lang_{M2L}$ n (*FAnd* φ_1 φ_2) $\subseteq$
$lang_{M2L}\ n\ \varphi_1 \cap lang_{M2L}\ n\ \varphi_2 \cap \{enc\ (w,\ I) \mid w\ I.\ length\ I = n \wedge$ *wf-interp-for-formula* $(w,\ I)$ (*FAnd* φ_1 φ_2)$\}$
(**is** - $\subseteq$ *?L1* $\cap$ *?L2* $\cap$ *?ENC*)
⟨*proof*⟩

10.3 From M2L to Regular expressions

fun *rexp-of* :: *nat* $\Rightarrow$ *'a formula* $\Rightarrow$ (*'a atom*) *rexp* **where**
rexp-of n (*FQ a m*) = *Inter* (*TIMES* [*Full*, *Atom* (*AQ m a*), *Full*]) (*ENC* n $\{m\}$)
| *rexp-of* n (*FLess m1 m2*) = (*if m1 = m2 then Zero else Inter*
(*TIMES* [*Full*, *Atom* (*Arbitrary-Except m1 True*), *Full*, *Atom* (*Arbitrary-Except m2 True*), *Full*])
(*ENC* n $\{m1,\ m2\}$))
| *rexp-of* n (*FIn m M*) =
Inter (*TIMES* [*Full*, *Atom* (*Arbitrary-Except2 m M*), *Full*]) (*ENC* n $\{m\}$)
| *rexp-of* n (*FNot* φ) = *Inter* (*rexp.Not* (*rexp-of* n φ)) (*ENC* n (*FOV* (*FNot* φ)))
| *rexp-of* n (*FOr* φ_1 φ_2) = *Inter* (*Plus* (*rexp-of* n φ_1) (*rexp-of* n φ_2)) (*ENC* n (*FOV* (*FOr* φ_1 φ_2)))
| *rexp-of* n (*FAnd* φ_1 φ_2) = *INTERSECT* [*rexp-of* n φ_1, *rexp-of* n φ_2, *ENC* n (*FOV* (*FAnd* φ_1 φ_2))]
| *rexp-of* n (*FExists* φ) = *Pr* (*rexp-of* $(n + 1)$ φ)

| *rexp-of n* (*FEXISTS* φ) = *Pr* (*rexp-of* (*n* + *1*) φ)

fun *rexp-of-alt* :: *nat* ⇒ *'a formula* ⇒ (*'a atom*) *rexp* **where**
rexp-of-alt n (*FQ a m*) = *TIMES* [*Full*, *Atom* (*AQ m a*), *Full*]
| *rexp-of-alt n* (*FLess m1 m2*) = (*if m1* = *m2 then Zero else*
TIMES [*Full*, *Atom* (*Arbitrary-Except m1 True*), *Full*, *Atom* (*Arbitrary-Except m2 True*), *Full*])
| *rexp-of-alt n* (*FIn m M*) = *TIMES* [*Full*, *Atom* (*Arbitrary-Except2 m M*), *Full*]
| *rexp-of-alt n* (*FNot* φ) = *rexp.Not* (*rexp-of-alt n* φ)
| *rexp-of-alt n* (*FOr* φ_1 φ_2) = *Plus* (*rexp-of-alt n* φ_1) (*rexp-of-alt n* φ_2)
| *rexp-of-alt n* (*FAnd* φ_1 φ_2) = *Inter* (*rexp-of-alt n* φ_1) (*rexp-of-alt n* φ_2)
| *rexp-of-alt n* (*FExists* φ) = *Pr* (*Inter* (*rexp-of-alt* (*n* + *1*) φ) (*ENC* (*n* + *1*) (*FOV* φ)))
| *rexp-of-alt n* (*FEXISTS* φ) = *Pr* (*Inter* (*rexp-of-alt* (*n* + *1*) φ) (*ENC* (*n* + *1*) (*FOV* φ)))

definition *rexp-of′ n* φ = *Inter* (*rexp-of-alt n* φ) (*ENC n* (*FOV* φ))

fun *rexp-of-alt′* :: *nat* ⇒ *'a formula* ⇒ (*'a atom*) *rexp* **where**
rexp-of-alt′ n (*FQ a m*) = *TIMES* [*Full*, *Atom* (*AQ m a*), *Full*]
| *rexp-of-alt′ n* (*FLess m1 m2*) = (*if m1* = *m2 then Zero else*
TIMES [*Full*, *Atom* (*Arbitrary-Except m1 True*), *Full*, *Atom* (*Arbitrary-Except m2 True*), *Full*])
| *rexp-of-alt′ n* (*FIn m M*) = *TIMES* [*Full*, *Atom* (*Arbitrary-Except2 m M*), *Full*]
| *rexp-of-alt′ n* (*FNot* φ) = *rexp.Not* (*rexp-of-alt′ n* φ)
| *rexp-of-alt′ n* (*FOr* φ_1 φ_2) = *Plus* (*rexp-of-alt′ n* φ_1) (*rexp-of-alt′ n* φ_2)
| *rexp-of-alt′ n* (*FAnd* φ_1 φ_2) = *Inter* (*rexp-of-alt′ n* φ_1) (*rexp-of-alt′ n* φ_2)
| *rexp-of-alt′ n* (*FExists* φ) = *Pr* (*Inter* (*rexp-of-alt′* (*n* + *1*) φ) (*ENC* (*n* + *1*) {*0*}))
| *rexp-of-alt′ n* (*FEXISTS* φ) = *Pr* (*rexp-of-alt′* (*n* + *1*) φ)

definition *rexp-of′′ n* φ = *Inter* (*rexp-of-alt′ n* φ) (*ENC n* (*FOV* φ))

theorem $lang_{M2L}$*-rexp-of*: *wf-formula n* φ ⟹ $lang_{M2L}$ *n* φ = *lang n* (*rexp-of n* φ) − {[]}
(**is** - ⟹ - = *?L n* φ)
⟨*proof*⟩

lemma *wf-rexp-of*: *wf-formula n* φ ⟹ *wf n* (*rexp-of n* φ)
⟨*proof*⟩

lemma *wf-rexp-of-alt*: *wf-formula n* φ ⟹ *wf n* (*rexp-of-alt n* φ)
⟨*proof*⟩

lemma *wf-rexp-of′*: *wf-formula n* φ ⟹ *wf n* (*rexp-of′ n* φ)
⟨*proof*⟩

lemma *wf-rexp-of-alt′*: *wf-formula n* φ ⟹ *wf n* (*rexp-of-alt′ n* φ)
⟨*proof*⟩

lemma *wf-rexp-of''*: *wf-formula n* $\varphi \Longrightarrow$ *wf n* (*rexp-of'' n* φ)
⟨*proof*⟩

lemma *ENC-Not*: *ENC n* (*FOV* (*FNot* φ)) = *ENC n* (*FOV* φ)
⟨*proof*⟩

lemma *ENC-And*:
wf-formula n (*FAnd* φ ψ) $\Longrightarrow$ *lang n* (*ENC n* (*FOV* (*FAnd* φ ψ))) $-$ {[]} $\subseteq$ *lang n* (*ENC n* (*FOV* φ)) $\cap$ *lang n* (*ENC n* (*FOV* ψ)) $-$ {[]}
⟨*proof*⟩

lemma *ENC-Or*:
wf-formula n (*FOr* φ ψ) $\Longrightarrow$ *lang n* (*ENC n* (*FOV* (*FOr* φ ψ))) $-$ {[]} $\subseteq$ *lang n* (*ENC n* (*FOV* φ)) $\cap$ *lang n* (*ENC n* (*FOV* ψ)) $-$ {[]}
⟨*proof*⟩

lemma *project-enc*: *map* π (*enc* (*w*, *x* # *I*)) = *enc* (*w*, *I*)
⟨*proof*⟩

lemma *list-list-eqI*:
assumes $\forall$ (-, *x*) $\in$ *set xs*. *x* $\neq$ [] $\forall$ (-, *y*) $\in$ *set ys*. *y* $\neq$ []
map (λ(-, *x*). *hd x*) *xs* = *map* (λ(-, *x*). *hd x*) *ys* *map* π *xs* = *map* π *ys*
shows *xs* = *ys*
⟨*proof*⟩

lemma *project-enc-extend*:
assumes *map* π *x* = *enc* (*w*, *I*) $\forall$ (-, *x*) $\in$ *set x*. *x* $\neq$ []
shows *x* = *enc* (*w*, *Inr* (*positions-in-row x 0*) # *I*)
⟨*proof*⟩

lemma *ENC-Exists*:
wf-formula n (*FExists* φ) $\Longrightarrow$ *lang n* (*ENC n* (*FOV* (*FExists* φ))) $-$ {[]} = *map* π ‘ *lang* (*Suc n*) (*ENC* (*Suc n*) (*FOV* φ)) $-$ {[]}
⟨*proof*⟩

lemma *ENC-EXISTS*:
wf-formula n (*FEXISTS* φ) $\Longrightarrow$ *lang n* (*ENC n* (*FOV* (*FEXISTS* φ))) $-$ {[]} = *map* π ‘ *lang* (*Suc n*) (*ENC* (*Suc n*) (*FOV* φ)) $-$ {[]}
⟨*proof*⟩

lemma *map-project-empty*: *map* π ‘ *A* $-$ {[]} = *map* π ‘ (*A* $-$ {[]})
⟨*proof*⟩

lemma $lang_{M2L}$*-rexp-of-rexp-of'*:
wf-formula n $\varphi \Longrightarrow$ *lang n* (*rexp-of n* φ) $-$ {[]} = *lang n* (*rexp-of' n* φ) $-$ {[]}
⟨*proof*⟩

lemma *Int-Diff-both*: $A \cap B - C = (A - C) \cap (B - C)$
⟨*proof*⟩

lemma *lang-ENC-split*:
assumes *finite* X $X = Y1 \cup Y2$ $n = 0 \lor (\forall p \in X.\ p < n)$
shows $lang\ n\ (ENC\ n\ X) = lang\ n\ (ENC\ n\ Y1) \cap lang\ n\ (ENC\ n\ Y2)$
⟨*proof*⟩

lemma *map-project-Int-ENC*:
assumes $0 \notin X$ $X \subseteq \{0\ ..< n + 1\}$ $Z \subseteq lists\ ((set\ o\ \sigma\ \Sigma)\ (n + 1))$
shows $map\ \pi\ `\ (Z \cap lang\ (n + 1)\ (ENC\ (n + 1)\ X) - \{[]\}) =$
$map\ \pi\ `\ Z \cap lang\ n\ (ENC\ n\ ((\lambda x.\ x - 1)\ `\ X)) - \{[]\}$
⟨*proof*⟩

lemma *map-project-ENC*:
assumes $X \subseteq \{0\ ..< n + 1\}$ $Z \subseteq lists\ ((set\ o\ \sigma\ \Sigma)\ (n + 1))$
shows $map\ \pi\ `\ (Z \cap lang\ (n + 1)\ (ENC\ (n + 1)\ X) - \{[]\}) =$
$(if\ 0 \in X$
$then\ map\ \pi\ `\ (Z \cap lang\ (n + 1)\ (ENC\ (n + 1)\ \{0\})) \cap lang\ n\ (ENC\ n\ ((\lambda x.\ x - 1)\ `\ (X - \{0\}))) - \{[]\}$
$else\ map\ \pi\ `\ Z \cap lang\ n\ (ENC\ n\ ((\lambda x.\ x - 1)\ `\ (X - \{0\}))) - \{[]\})$
(**is** $?L = (if\ -\ then\ ?R1\ else\ ?R2)$)
⟨*proof*⟩

abbreviation $\mathfrak{L} \equiv project.lang\ (set \circ \sigma\ \Sigma)\ \pi$

lemma $lang_{M2L}$*-rexp-of′-rexp-of″*:
$wf\text{-}formula\ n\ \varphi \Longrightarrow lang\ n\ (rexp\text{-}of'\ n\ \varphi) - \{[]\} = lang\ n\ (rexp\text{-}of''\ n\ \varphi) - \{[]\}$
⟨*proof*⟩

theorem $lang_{M2L}$*-rexp-of′*: $wf\text{-}formula\ n\ \varphi \Longrightarrow lang_{M2L}\ n\ \varphi = lang\ n\ (rexp\text{-}of'\ n\ \varphi) - \{[]\}$
⟨*proof*⟩

theorem $lang_{M2L}$*-rexp-of″*: $wf\text{-}formula\ n\ \varphi \Longrightarrow lang_{M2L}\ n\ \varphi = lang\ n\ (rexp\text{-}of''\ n\ \varphi) - \{[]\}$
⟨*proof*⟩

end

11 Normalization of M2L Formulas

fun *nNot* **where**
$nNot\ (FNot\ \varphi) = \varphi$
$|\ nNot\ (FAnd\ \varphi 1\ \varphi 2) = FOr\ (nNot\ \varphi 1)\ (nNot\ \varphi 2)$
$|\ nNot\ (FOr\ \varphi 1\ \varphi 2) = FAnd\ (nNot\ \varphi 1)\ (nNot\ \varphi 2)$
$|\ nNot\ \varphi = FNot\ \varphi$

primrec *norm* **where**
  $norm\ (FQ\ a\ m) = FQ\ a\ m$
| $norm\ (FLess\ m\ n) = FLess\ m\ n$
| $norm\ (FIn\ m\ M) = FIn\ m\ M$
| $norm\ (FOr\ \varphi\ \psi) = FOr\ (norm\ \varphi)\ (norm\ \psi)$
| $norm\ (FAnd\ \varphi\ \psi) = FAnd\ (norm\ \varphi)\ (norm\ \psi)$
| $norm\ (FNot\ \varphi) = nNot\ (norm\ \varphi)$
| $norm\ (FExists\ \varphi) = FExists\ (norm\ \varphi)$
| $norm\ (FEXISTS\ \varphi) = FEXISTS\ (norm\ \varphi)$

context *formula*
begin

lemma *satisfies-nNot*[*simp*]: $satisfies\ (w,\ I)\ (nNot\ \varphi) = satisfies\ (w,I)\ (FNot\ \varphi)$
  ⟨*proof*⟩

lemma *FOV-nNot*[*simp*]: $FOV\ (nNot\ \varphi) = FOV\ (FNot\ \varphi)$
  ⟨*proof*⟩

lemma *SOV-nNot*[*simp*]: $SOV\ (nNot\ \varphi) = SOV\ (FNot\ \varphi)$
  ⟨*proof*⟩

lemma *pre-wf-formula-nNot*[*simp*]: *pre-wf-formula* $n\ (nNot\ \varphi)$ = *pre-wf-formula* $n\ (FNot\ \varphi)$
  ⟨*proof*⟩

lemma *FOV-norm*[*simp*]: $FOV\ (norm\ \varphi) = FOV\ \varphi$
  ⟨*proof*⟩

lemma *SOV-norm*[*simp*]: $SOV\ (norm\ \varphi) = SOV\ \varphi$
  ⟨*proof*⟩

lemma *pre-wf-formula-norm*[*simp*]: *pre-wf-formula* $n\ (norm\ \varphi)$ = *pre-wf-formula* $n\ \varphi$
  ⟨*proof*⟩

lemma *satisfies-norm*[*simp*]: $satisfies\ (w,\ I)\ (norm\ \varphi) = satisfies\ (w,\ I)\ \varphi$
  ⟨*proof*⟩

lemma $lang_{M2L}$*-norm*[*simp*]: $lang_{M2L}\ n\ (norm\ \varphi) = lang_{M2L}\ n\ \varphi$
  ⟨*proof*⟩

end

12 Deciding Equivalence of M2L Formulas

global-interpretation *embed set o* σ Σ *wf-atom* Σ π *lookup* ε Σ
for Σ :: *'a* :: *linorder list*
defines
$\mathfrak{D}$ = *embed.lderiv lookup* (ε Σ)
and *Co*$\mathfrak{D}$ = *embed.lderiv-dual lookup* (ε Σ)
⟨*proof*⟩

lemma *enum-not-empty*[*simp*]: *Enum.enum* ≠ [] (**is** *?enum* ≠ [])
⟨*proof*⟩

global-interpretation Φ: *formula Enum.enum* :: *'a* :: {*enum*, *linorder*} *list*
defines
pre-wf-formula = Φ.*pre-wf-formula*
and *wf-formula* = Φ.*wf-formula*
and *rexp-of* = Φ.*rexp-of*
and *rexp-of-alt* = Φ.*rexp-of-alt*
and *rexp-of-alt'* = Φ.*rexp-of-alt'*
and *rexp-of'* = Φ.*rexp-of'*
and *rexp-of''* = Φ.*rexp-of''*
and *valid-ENC* = Φ.*valid-ENC*
and *ENC* = Φ.*ENC*
and *dec-interp* = Φ.*dec-interp*
⟨*proof*⟩

lemma *lang-Plus-Zero*: *lang* Σ *n* (*Plus r One*) = *lang* Σ *n* (*Plus s One*) ⟷ *lang* Σ *n r* − {[]} = *lang* Σ *n s* − {[]}
⟨*proof*⟩

lemmas $lang_{M2L}$*-rexp-of-norm* = *trans*[*OF sym*[*OF* Φ.$lang_{M2L}$*-norm*] Φ.$lang_{M2L}$*-rexp-of*]
lemmas $lang_{M2L}$*-rexp-of'-norm* = *trans*[*OF sym*[*OF* Φ.$lang_{M2L}$*-norm*] Φ.$lang_{M2L}$*-rexp-of'*]
lemmas $lang_{M2L}$*-rexp-of''-norm* = *trans*[*OF sym*[*OF* Φ.$lang_{M2L}$*-norm*] Φ.$lang_{M2L}$*-rexp-of''*]

⟨*ML*⟩

global-interpretation *D*: *rexp-DFA* σ Σ *wf-atom* Σ π *lookup* λx. «*pnorm* (*inorm* *x*)»
λa *r*. «$\mathfrak{D}$ Σ *a r*» *final alphabet.wf* (*wf-atom* Σ) *n pnorm lang* Σ *n n*
for Σ :: *'a* :: *linorder list* **and** *n* :: *nat*
defines
test = *rexp-DA.test* (*final* :: *'a atom rexp* ⇒ *bool*)
and *step* = *rexp-DA.step* (σ Σ) (λa *r*. «$\mathfrak{D}$ Σ *a r*») *pnorm n*
and *closure* = *rexp-DA.closure* (σ Σ) (λa *r*. «$\mathfrak{D}$ Σ *a r*») *final pnorm n*
and *check-eqvRE* = *rexp-DA.check-eqv* (σ Σ) (λx. «*pnorm* (*inorm x*)») (λa *r*. «$\mathfrak{D}$ Σ *a r*») *final pnorm n*
and *test-invariant* = *rexp-DA.test-invariant* (*final* :: *'a atom rexp* ⇒ *bool*) ::
((*'a* × *bool list*) *list* × -) *list* × - ⇒ *bool*
and *step-invariant* = *rexp-DA.step-invariant* (σ Σ) (λa *r*. «$\mathfrak{D}$ Σ *a r*») *pnorm n*
and *closure-invariant* = *rexp-DA.closure-invariant* (σ Σ) (λa *r*. «$\mathfrak{D}$ Σ *a r*») *final pnorm n*

and *counterexampleRE* = *rexp-DA.counterexample* (σ Σ) (λx. «*pnorm* (*inorm* *x*)») (λa *r*. «$\mathfrak{D}$ Σ *a r*») *final pnorm n*
and *reachable* = *rexp-DA.reachable* (σ Σ) (λx. «*pnorm* (*inorm x*)») (λa *r*. «$\mathfrak{D}$ Σ *a r*») *pnorm n*
and *automaton* = *rexp-DA.automaton* (σ Σ) (λx. «*pnorm* (*inorm x*)») (λa *r*. «$\mathfrak{D}$ Σ *a r*») *pnorm n*
⟨*proof*⟩

definition *check-eqv* **where**
check-eqv n φ ψ $\longleftrightarrow$ *wf-formula n* (*FOr* φ ψ) $\wedge$
slow.check-eqvRE Enum.enum n (*Plus* (*rexp-of''* *n* (*norm* φ)) *One*) (*Plus* (*rexp-of''* *n* (*norm* ψ)) *One*)

definition *counterexample* **where**
counterexample n φ ψ =
map-option (λw. *dec-interp n* (*FOV* (*FOr* φ ψ)) *w*)
(*slow.counterexampleRE Enum.enum n* (*Plus* (*rexp-of''* *n* (*norm* φ)) *One*) (*Plus* (*rexp-of''* *n* (*norm* ψ)) *One*))

lemma *soundness*: *slow.check-eqv n* φ ψ $\Longrightarrow$ $\Phi.lang_{M2L}$ *n* φ = $\Phi.lang_{M2L}$ *n* ψ
⟨*proof*⟩

lemma *completeness*:
assumes $\Phi.lang_{M2L}$ *n* φ = $\Phi.lang_{M2L}$ *n* ψ *wf-formula n* (*FOr* φ ψ)
shows *slow.check-eqv n* φ ψ
⟨*proof*⟩

⟨*ML*⟩

global-interpretation *D*: *rexp-DA-no-post* σ Σ *wf-atom* Σ π *lookup* λx. *pnorm* (*inorm x*)
λa *r*. *pnorm* ($\mathfrak{D}$ Σ *a r*) *final alphabet.wf* (*wf-atom* Σ) *n lang* Σ *n n*
for Σ :: *'a* :: *linorder list* **and** *n* :: *nat*
defines
test = *rexp-DA.test* (*final* :: *'a atom rexp* $\Rightarrow$ *bool*)
and *step* = *rexp-DA.step* (σ Σ) (λa *r*. *pnorm* ($\mathfrak{D}$ Σ *a r*)) *id n*
and *closure* = *rexp-DA.closure* (σ Σ) (λa *r*. *pnorm* ($\mathfrak{D}$ Σ *a r*)) *final id n*
and *check-eqvRE* = *rexp-DA.check-eqv* (σ Σ) (λx. *pnorm* (*inorm x*)) (λa *r*. *pnorm* ($\mathfrak{D}$ Σ *a r*)) *final id n*
and *test-invariant* = *rexp-DA.test-invariant* (*final* :: *'a atom rexp* $\Rightarrow$ *bool*) ::
((*'a* $\times$ *bool list*) *list* $\times$ -) *list* $\times$ - $\Rightarrow$ *bool*
and *step-invariant* = *rexp-DA.step-invariant* (σ Σ) (λa *r*. *pnorm* ($\mathfrak{D}$ Σ *a r*)) *id n*
and *closure-invariant* = *rexp-DA.closure-invariant* (σ Σ) (λa *r*. *pnorm* ($\mathfrak{D}$ Σ *a r*)) *final id n*
and *counterexampleRE* = *rexp-DA.counterexample* (σ Σ) (λx. *pnorm* (*inorm x*)) (λa *r*. *pnorm* ($\mathfrak{D}$ Σ *a r*)) *final id n*
and *reachable* = *rexp-DA.reachable* (σ Σ) (λx. *pnorm* (*inorm x*)) (λa *r*. *pnorm* ($\mathfrak{D}$ Σ *a r*)) *id n*

and *automaton* = *rexp-DA.automaton* (σ Σ) (λx. *pnorm* (*inorm* x)) (λa r. *pnorm* ($\mathfrak{D}$ Σ a r)) *id* n
⟨*proof*⟩

definition *check-eqv* **where**
check-eqv n φ ψ $\longleftrightarrow$ *wf-formula* n (*FOr* φ ψ) $\wedge$
fast.check-eqvRE Enum.enum n (*Plus* (*rexp-of''* n (*norm* φ)) *One*) (*Plus* (*rexp-of''* n (*norm* ψ)) *One*)

definition *counterexample* **where**
counterexample n φ ψ =
map-option (λw. *dec-interp* n (*FOV* (*FOr* φ ψ)) w)
(*fast.counterexampleRE Enum.enum* n (*Plus* (*rexp-of''* n (*norm* φ)) *One*) (*Plus* (*rexp-of''* n (*norm* ψ)) *One*))

lemma *soundness*: *fast.check-eqv* n φ ψ $\Longrightarrow$ Φ.*lang*$_{M2L}$ n φ = Φ.*lang*$_{M2L}$ n ψ
⟨*proof*⟩

⟨*ML*⟩

global-interpretation *D*: *rexp-DA-no-post* σ Σ *wf-atom* Σ π *lookup*
λx. *pnorm-dual* (*rexp-dual-of* (*inorm* x)) λa r. *pnorm-dual* (*Co*$\mathfrak{D}$ Σ a r) *final-dual*
alphabet.wf-dual (*wf-atom* Σ) n *lang-dual* Σ n n
for Σ :: *'a* :: *linorder list* **and** n :: *nat*
defines
test = *rexp-DA.test* (*final-dual* :: *'a atom rexp-dual* $\Rightarrow$ *bool*)
and *step* = *rexp-DA.step* (σ Σ) (λa r. *pnorm-dual* (*Co*$\mathfrak{D}$ Σ a r)) *id* n
and *closure* = *rexp-DA.closure* (σ Σ) (λa r. *pnorm-dual* (*Co*$\mathfrak{D}$ Σ a r)) *final-dual* *id* n
and *check-eqvRE* = *rexp-DA.check-eqv* (σ Σ) (λx. *pnorm-dual* (*rexp-dual-of* (*inorm* x))) (λa r. *pnorm-dual* (*Co*$\mathfrak{D}$ Σ a r)) *final-dual id* n
and *test-invariant* = *rexp-DA.test-invariant* (*final-dual* :: *'a atom rexp-dual* $\Rightarrow$ *bool*) ::
((*'a* $\times$ *bool list*) *list* $\times$ -) *list* $\times$ - $\Rightarrow$ *bool*
and *step-invariant* = *rexp-DA.step-invariant* (σ Σ) (λa r. *pnorm-dual* (*Co*$\mathfrak{D}$ Σ a r)) *id* n
and *closure-invariant* = *rexp-DA.closure-invariant* (σ Σ) (λa r. *pnorm-dual* (*Co*$\mathfrak{D}$ Σ a r)) *final-dual id* n
and *counterexampleRE* = *rexp-DA.counterexample* (σ Σ) (λx. *pnorm-dual* (*rexp-dual-of* (*inorm* x))) (λa r. *pnorm-dual* (*Co*$\mathfrak{D}$ Σ a r)) *final-dual id* n
and *reachable* = *rexp-DA.reachable* (σ Σ) (λx. *pnorm-dual* (*rexp-dual-of* (*inorm* x))) (λa r. *pnorm-dual* (*Co*$\mathfrak{D}$ Σ a r)) *id* n
and *automaton* = *rexp-DA.automaton* (σ Σ) (λx. *pnorm-dual* (*rexp-dual-of* (*inorm* x))) (λa r. *pnorm-dual* (*Co*$\mathfrak{D}$ Σ a r)) *id* n
⟨*proof*⟩

definition *check-eqv* **where**
check-eqv n φ ψ $\longleftrightarrow$ *wf-formula* n (*FOr* φ ψ) $\wedge$
dual.check-eqvRE Enum.enum n (*Plus* (*rexp-of''* n (*norm* φ)) *One*) (*Plus* (*rexp-of''*

n (*norm* ψ)) *One*)

definition *counterexample* **where**
counterexample $n\ \varphi\ \psi$ =
map-option (λw. *dec-interp* n (*FOV* (*FOr* $\varphi\ \psi$)) w)
(*dual.counterexampleRE Enum.enum* n (*Plus* (*rexp-of''* n (*norm* φ)) *One*) (*Plus* (*rexp-of''* n (*norm* ψ)) *One*))

lemma *soundness*: *dual.check-eqv* $n\ \varphi\ \psi \Longrightarrow \Phi.lang_{M2L}\ n\ \varphi = \Phi.lang_{M2L}\ n\ \psi$
⟨*proof*⟩

⟨*ML*⟩

13 WS1S

13.1 Encodings

definition *cut-same* $x\ s$ = *stake* (*LEAST* n. *sdrop* $n\ s$ = *sconst* x) s

abbreviation *poss* $I \equiv (\bigcup x{\in}set\ I.\ case\ x\ of\ Inl\ p \Rightarrow \{p\}\ |\ Inr\ P \Rightarrow P)$

declare *smap-sconst*[*simp*]

lemma (**in** *wellorder*) *min-Least*:
$[\![\exists n.\ P\ n; \exists n.\ Q\ n]\!] \Longrightarrow min\ (Least\ P)\ (Least\ Q) = (LEAST\ n.\ P\ n \lor Q\ n)$
⟨*proof*⟩

lemma *sconst-collapse*: y ## *sconst* y = *sconst* y
⟨*proof*⟩

lemma *shift-sconst-inj*: $[\![$*length* x = *length* y; x @− *sconst* z = y @− *sconst* $z]\!] \Longrightarrow$ $x = y$
⟨*proof*⟩

context *formula*
begin

definition *any* $\equiv$ *hd* Σ

lemma *any-*Σ[*simp*]: *any* $\in$ *set* Σ
⟨*proof*⟩

lemma *any-*σ[*simp*]: *length bs* = $n \Longrightarrow$ (*any*, *bs*) $\in$ *set* ($\sigma\ \Sigma\ n$)
⟨*proof*⟩

fun *stream-enc* :: *'a interp* $\Rightarrow$ (*'a* $\times$ *bool list*) *stream* **where**
stream-enc (w, I) = *smap2* (*enc-atom* I) *nats* (w @− *sconst any*)

lemma *tl-stream-enc*[*simp*]: *smap* π (*stream-enc* (w, x # I)) = *stream-enc* (w, I)

$\langle proof \rangle$

lemma *enc-atom-max*: $[\![\forall x{\in}set\ I.\ case\ x\ of\ Inl\ p \Rightarrow p \leq n \mid Inr\ P \Rightarrow \forall p{\in}P.\ p \leq n;\ n \leq n']\!] \Longrightarrow$
$enc\text{-}atom\ I\ (Suc\ n')\ a = (a,\ replicate\ (length\ I)\ False)$
$\langle proof \rangle$

lemma *ex-Loop-stream-enc*:
assumes $\forall x \in set\ I.\ case\ x\ of\ Inr\ P \Rightarrow finite\ P \mid \text{-} \Rightarrow True$
shows $\exists n.\ sdrop\ n\ (stream\text{-}enc\ (w,\ I)) = sconst\ (any,\ replicate\ (length\ I)\ False)$
$\langle proof \rangle$

lemma *length-snth-enc*[*simp*]: $length\ (snd\ (stream\text{-}enc\ (w,\ I)\ !!\ n)) = length\ I$
$\langle proof \rangle$

lemma *sset-singleton*[*simp*]: $sset\ s \subseteq \{x\} \longleftrightarrow sset\ s = \{x\}$
$\langle proof \rangle$

lemma *drop-sconstE*: $[\![drop\ n\ w\ @-\ sconst\ y = sconst\ y;\ p < length\ w;\ \neg\ p < n]\!] \Longrightarrow w\ !\ p = y$
$\langle proof \rangle$

lemma *less-length-cut-same*:
$[\![(w\ @-\ sconst\ y)\ !!\ p = a]\!] \Longrightarrow a = y \lor (p < length\ (cut\text{-}same\ y\ (w\ @-\ sconst\ y)) \land w\ !\ p = a)$
$\langle proof \rangle$

lemma *less-length-cut-same-Inl*:
$[\![(\forall x \in set\ I.\ case\ x\ of\ Inr\ P \Rightarrow finite\ P \mid \text{-} \Rightarrow True);\ r < length\ I;\ I\ !\ r = Inl\ p]\!] \Longrightarrow$
$p < length\ (cut\text{-}same\ (any,\ replicate\ (length\ I)\ False)\ (stream\text{-}enc\ (w,\ I)))$
$\langle proof \rangle$

lemma *less-length-cut-same-Inr*:
$[\![(\forall x \in set\ I.\ case\ x\ of\ Inr\ P \Rightarrow finite\ P \mid \text{-} \Rightarrow True);\ r < length\ I;\ I\ !\ r = Inr\ P]\!] \Longrightarrow$
$\forall p \in P.\ p < length\ (cut\text{-}same\ (any,\ replicate\ (length\ I)\ False)\ (stream\text{-}enc\ (w,\ I)))$
$\langle proof \rangle$

fun *enc* :: $'a\ interp \Rightarrow ('a \times bool\ list)\ list\ set$ **where**
$enc\ (w,\ I) = \{x.\ \exists n.\ x = (cut\text{-}same\ (any,\ replicate\ (length\ I)\ False)\ (stream\text{-}enc\ (w,\ I))\ @$
$replicate\ n\ (any,\ replicate\ (length\ I)\ False))\}$

lemma *cut-same-all*[*simp*]: $cut\text{-}same\ x\ (sconst\ x) = []$
$\langle proof \rangle$

lemma *cut-same-stop*[*simp*]:

assumes $x \neq y$
shows *cut-same* x (*xs* @− y ## *sconst* x) = *xs* @ [y] (**is** *cut-same* x *?s* = -)
⟨*proof*⟩

lemma *cut-same-shift-sconst*: $\exists\, n.$ w = *cut-same* x (w @− *sconst* x) @ *replicate* n x
⟨*proof*⟩

lemma *set-cut-same*: *set* (*cut-same* x (w @− *sconst* x)) $\subseteq$ *set* w
⟨*proof*⟩

lemma *stream-enc-cut-same*:
assumes ($\forall\, x \in$ *set* I. *case* x *of Inr* $P \Rightarrow$ *finite* P | - $\Rightarrow$ *True*)
shows *stream-enc* (w, I) = *cut-same* (*any*, *replicate* (*length* I) *False*) (*stream-enc* (w, I)) @−
sconst (*any*, *replicate* (*length* I) *False*)
⟨*proof*⟩

lemma *stream-enc-enc*:
assumes ($\forall\, x \in$ *set* I. *case* x *of Inr* $P \Rightarrow$ *finite* P | - $\Rightarrow$ *True*) **and** v: $v \in$ *enc* (w, I)
shows *stream-enc* (w, I) = v @− *sconst* (*any*, *replicate* (*length* I) *False*)
(**is** *?s* = *?v* @− *sconst* *?F*)
⟨*proof*⟩

lemma *stream-enc-enc-some*:
assumes ($\forall\, x \in$ *set* I. *case* x *of Inr* $P \Rightarrow$ *finite* P | - $\Rightarrow$ *True*)
shows *stream-enc* (w, I) = (*SOME* v. $v \in$ *enc* (w, I)) @− *sconst* (*any*, *replicate* (*length* I) *False*)
⟨*proof*⟩

lemma *enc-unique-length*: $v \in$ *enc* (w, I) $\Longrightarrow \forall\, v'$. *length* v' = *length* $v \wedge v' \in$ *enc* (w, I) $\longrightarrow v = v'$
⟨*proof*⟩

lemma *sdrop-sconst*: *sdrop* n s = *sconst* $x \Longrightarrow n \leq m \Longrightarrow s$!! $m = x$
⟨*proof*⟩

lemma *fin-cut-same-tl*:
assumes $\exists\, n.$ *sdrop* n s = *sconst* x
shows *fin-cut-same* (π x) (*map* π (*cut-same* x s)) = *cut-same* (π x) (*smap* π s)
⟨*proof*⟩

lemma *tl-enc*[*simp*]:
assumes $\forall\, x \in$ *set* (x # I). *case* x *of Inr* $P \Rightarrow$ *finite* P | - $\Rightarrow$ *True*
shows *SAMEQUOT* (*any*, *replicate* (*length* I) *False*) (*map* π ‘ *enc* (w, x # I)) = *enc* (w, I)
⟨*proof*⟩

lemma *encD*:
$[\![v \in enc\ (w, I); (\forall x \in set\ I.\ case\ x\ of\ Inr\ P \Rightarrow finite\ P \mid \text{-} \Rightarrow True)]\!] \Longrightarrow$
$v = map\ (case\text{-}prod\ (enc\text{-}atom\ I))\ (zip\ [0\ ..< length\ v]\ (stake\ (length\ v)\ (w\ @- sconst\ any)))$
$\langle proof \rangle$

lemma *enc-Inl*: $[\![x \in enc\ (w, I); (\forall x \in set\ I.\ case\ x\ of\ Inr\ P \Rightarrow finite\ P \mid \text{-} \Rightarrow True);$
$m < length\ I; I\ !\ m = Inl\ p]\!] \Longrightarrow p < length\ x \wedge snd\ (x\ !\ p)\ !\ m$
$\langle proof \rangle$

lemma *enc-Inr*: **assumes** $x \in enc\ (w, I)$ $\forall x \in set\ I.\ case\ x\ of\ Inr\ P \Rightarrow finite\ P \mid \text{-} \Rightarrow True$
$M < length\ I$ $I\ !\ M = Inr\ P$
shows $p \in P \longleftrightarrow p < length\ x \wedge snd\ (x\ !\ p)\ !\ M$
$\langle proof \rangle$

lemma *enc-length*:
assumes $enc\ (w, I) = enc\ (w', I')$
shows $length\ I = length\ I'$
$\langle proof \rangle$

lemma *enc-stream-enc*:
$[\![(\forall x \in set\ I.\ case\ x\ of\ Inr\ P \Rightarrow finite\ P \mid \text{-} \Rightarrow True);$
$(\forall x \in set\ I'.\ case\ x\ of\ Inr\ P \Rightarrow finite\ P \mid \text{-} \Rightarrow True);$
$enc\ (w, I) = enc\ (w', I')]\!] \Longrightarrow stream\text{-}enc\ (w, I) = stream\text{-}enc\ (w', I')$
$\langle proof \rangle$

abbreviation *wf-interp* $w\ I \equiv$
$((\forall a \in set\ w.\ a \in set\ \Sigma) \wedge (\forall x \in set\ I.\ case\ x\ of\ Inr\ P \Rightarrow finite\ P \mid \text{-} \Rightarrow True))$

fun *wf-interp-for-formula* :: $'a\ interp \Rightarrow 'a\ formula \Rightarrow bool$ **where**
$wf\text{-}interp\text{-}for\text{-}formula\ (w, I)\ \varphi =$
$(wf\text{-}interp\ w\ I\ \wedge$
$(\forall n \in FOV\ \varphi.\ case\ I\ !\ n\ of\ Inl\ \text{-} \Rightarrow True \mid \text{-} \Rightarrow False) \wedge$
$(\forall n \in SOV\ \varphi.\ case\ I\ !\ n\ of\ Inl\ \text{-} \Rightarrow False \mid Inr\ \text{-} \Rightarrow True))$

fun *satisfies* :: $'a\ interp \Rightarrow 'a\ formula \Rightarrow bool$ (**infix** $\langle\models\rangle$ *50*) **where**
$(w, I) \models FQ\ a\ m = ((case\ I\ !\ m\ of\ Inl\ p \Rightarrow if\ p < length\ w\ then\ w\ !\ p\ else\ any) = a)$
$\mid (w, I) \models FLess\ m1\ m2 = ((case\ I\ !\ m1\ of\ Inl\ p \Rightarrow p) < (case\ I\ !\ m2\ of\ Inl\ p \Rightarrow p))$
$\mid (w, I) \models FIn\ m\ M = ((case\ I\ !\ m\ of\ Inl\ p \Rightarrow p) \in (case\ I\ !\ M\ of\ Inr\ P \Rightarrow P))$
$\mid (w, I) \models FNot\ \varphi = (\neg\ (w, I) \models \varphi)$
$\mid (w, I) \models (FOr\ \varphi_1\ \varphi_2) = ((w, I) \models \varphi_1 \vee (w, I) \models \varphi_2)$
$\mid (w, I) \models (FAnd\ \varphi_1\ \varphi_2) = ((w, I) \models \varphi_1 \wedge (w, I) \models \varphi_2)$
$\mid (w, I) \models (FExists\ \varphi) = (\exists p.\ (w, Inl\ p\ \#\ I) \models \varphi)$
$\mid (w, I) \models (FEXISTS\ \varphi) = (\exists P.\ finite\ P \wedge (w, Inr\ P\ \#\ I) \models \varphi)$

definition $lang_{WS1S}$ :: $nat \Rightarrow 'a\ formula \Rightarrow ('a \times bool\ list)\ list\ set$ **where**
$lang_{WS1S}\ n\ \varphi = \bigcup\{enc\ (w,\ I) \mid w\ I\ .\ length\ I = n \wedge wf\text{-}interp\text{-}for\text{-}formula\ (w,\ I)\ \varphi \wedge (w,\ I) \models \varphi\}$

lemma *encD-ex*: $[\![x \in enc\ (w,\ I);\ (\forall x \in set\ I.\ case\ x\ of\ Inr\ P \Rightarrow finite\ P \mid \text{-} \Rightarrow True)]\!] \Longrightarrow$
$\exists n.\ x = map\ (case\text{-}prod\ (enc\text{-}atom\ I))\ (zip\ [0\ ..<\ n]\ (stake\ n\ (w\ @-\ sconst\ any)))$
⟨*proof*⟩

lemma *enc-set-σ*: $[\![x \in enc\ (w,\ I);\ (\forall x \in set\ I.\ case\ x\ of\ Inr\ P \Rightarrow finite\ P \mid \text{-} \Rightarrow True);$
$length\ I = n;\ a \in set\ x;\ set\ w \subseteq set\ \Sigma]\!] \Longrightarrow a \in set\ (\sigma\ \Sigma\ n)$
⟨*proof*⟩

definition *positions-in-row s i* =
Option.these (*sset* (*smap2* (*λp* (-, *bs*). *if nth bs i then Some p else None*) *nats s*))

lemma *positions-in-row*: *positions-in-row s i* = {*p*. *snd* (*s* !! *p*) ! *i*}
⟨*proof*⟩

lemma *positions-in-row-unique*: ∃!*p*. *snd* (*s* !! *p*) ! *i* ⟹
the-elem (*positions-in-row s i*) = (*THE p*. *snd* (*s* !! *p*) ! *i*)
⟨*proof*⟩

lemma *positions-in-row-nth*: ∃!*p*. *snd* (*s* !! *p*) ! *i* ⟹
snd (*s* !! *the-elem* (*positions-in-row s i*)) ! *i*
⟨*proof*⟩

definition *dec-word s* = *cut-same any* (*smap fst s*)

lemma *dec-word-stream-enc*: *dec-word* (*stream-enc* (*w*, *I*)) = *cut-same any* (*w* @− *sconst any*)
⟨*proof*⟩

definition *stream-dec n FO* (*s* :: (*'a* × *bool list*) *stream*) = *map* (*λi*.
if i ∈ *FO*
then Inl (*the-elem* (*positions-in-row s i*))
else Inr (*positions-in-row s i*)) [*0*..<*n*]

lemma *stream-dec-Inl*: ⟦*i* ∈ *FO*; *i* < *n*⟧ ⟹ ∃*p*. *stream-dec n FO s* ! *i* = *Inl p*
⟨*proof*⟩

lemma *stream-dec-not-Inr*: ⟦*stream-dec n FO s* ! *i* = *Inr P*; *i* ∈ *FO*; *i* < *n*⟧ ⟹ *False*
⟨*proof*⟩

lemma *stream-dec-Inr*: ⟦*i* ∉ *FO*; *i* < *n*⟧ ⟹ ∃*P*. *stream-dec n FO s* ! *i* = *Inr P*
⟨*proof*⟩

lemma *stream-dec-not-Inl*: $[\![$*stream-dec n FO s* ! i = *Inl p*; $i \notin$ *FO*; $i < n]\!] \Longrightarrow$ *False*
⟨*proof*⟩

lemma *Inr-dec-finite*: $[\![\forall\, i<n.$ *finite* {*p. snd* (*s* !! *p*) ! *i*}; *Inr P* $\in$ *set* (*stream-dec n FO s*)$]\!] \Longrightarrow$
finite P
⟨*proof*⟩

lemma *enc-atom-dec*:
$[\![\forall\, p.$ *length* (*snd* (*s* !! *p*)) = *n*; $\forall\, i \in$ *FO*. $i < n \longrightarrow (\exists!p.$ *snd* (*s* !! *p*) ! *i*); *a* = *fst* (*s* !! *p*)$]\!] \Longrightarrow$
enc-atom (*stream-dec n FO s*) *p a* = *s* !! *p*
⟨*proof*⟩

lemma *length-stream-dec*[*simp*]: *length* (*stream-dec n FO x*) = *n*
⟨*proof*⟩

lemma *stream-enc-dec*:
$[\![\exists\, n.$ *sdrop n* (*smap fst s*) = *sconst any*;
stream-all ($\lambda x.$ *length* (*snd x*) = *n*) *s*; $\forall\, i \in$ *FO*. ($\exists!p.$ *snd* (*s* !! *p*) ! *i*)$]\!] \Longrightarrow$
stream-enc (*dec-word s*, *stream-dec n FO s*) = *s*
⟨*proof*⟩

lemma *stream-enc-unique*:
$i <$ *length I* $\Longrightarrow \exists\, p.$ *I* ! *i* = *Inl p* $\Longrightarrow \exists!p.$ *snd* (*stream-enc* (*w*, *I*) !! *p*) ! *i*
⟨*proof*⟩

lemma *stream-dec-enc-Inl*:
$[\![$*stream-dec n FO* (*stream-enc* (*w*, *I*)) ! *i* = *Inl p'*; *I* ! *i* = *Inl p*; $i \in$ *FO*; $i < n$; *length I* = $n]\!] \Longrightarrow$
$p = p'$
⟨*proof*⟩

lemma *stream-dec-enc-Inr*:
$[\![$*stream-dec n FO* (*stream-enc* (*w*, *I*)) ! *i* = *Inr P'*; *I* ! *i* = *Inr P*; $i \notin$ *FO*; $i <$ *n*; *length I* = $n]\!] \Longrightarrow$
$P = P'$
⟨*proof*⟩

lemma *Collect-snth*: {*p. P* ((*x* ## *s*) !! *p*)} $\subseteq$ {*0*} $\cup$ *Suc* ‘ {*p. P* (*s* !! *p*)}
⟨*proof*⟩

lemma *finite-True-in-row*: $\forall\, i < n.$ *finite* {*p. snd* ((*w* @− *sconst* (*any*, *replicate n False*)) !! *p*) ! *i*}
⟨*proof*⟩

lemma *lang-ENC*:

assumes $FO \subseteq \{0 ..< n\}$ $SO \subseteq \{0 ..< n\} - FO$
shows *lang n* (*ENC n FO*) $= \bigcup\{$*enc* (*w*, *I*) $\mid$ *w I* . *length* $I = n \wedge$ *wf-interp w I* $\wedge$
($\forall i \in FO$. *case I* ! *i of Inl* - $\Rightarrow$ *True* $\mid$ *Inr* - $\Rightarrow$ *False*) $\wedge$
($\forall i \in SO$. *case I* ! *i of Inl* - $\Rightarrow$ *False* $\mid$ *Inr* - $\Rightarrow$ *True*)$\}$
(**is** *?L* = *?R*)
$\langle$*proof*$\rangle$

lemma *lang-ENC-formula*:
assumes *wf-formula n* φ
shows *lang n* (*ENC n* (*FOV* φ)) $= \bigcup\{$*enc* (*w*, *I*) $\mid$ *w I* . *length* $I = n \wedge$ *wf-interp-for-formula* (*w*, *I*) $\varphi\}$
$\langle$*proof*$\rangle$

13.2 Welldefinedness of enc wrt. Models

lemma *wf-interp-for-formula-FExists*:
$[\![$*wf-formula* (*length I*) (*FExists* φ)$]\!] \Longrightarrow$
wf-interp-for-formula (*w*, *I*) (*FExists* φ) $\longleftrightarrow$ ($\forall p$. *wf-interp-for-formula* (*w*, *Inl p* # *I*) φ)
$\langle$*proof*$\rangle$

lemma *wf-interp-for-formula-any-Inl*: *wf-interp-for-formula* (*w*, *Inl p* # *I*) $\varphi \Longrightarrow$
$\forall p$. *wf-interp-for-formula* (*w*, *Inl p* # *I*) φ
$\langle$*proof*$\rangle$

lemma *wf-interp-for-formula-FEXISTS*:
$[\![$*wf-formula* (*length I*) (*FEXISTS* φ)$]\!] \Longrightarrow$
wf-interp-for-formula (*w*, *I*) (*FEXISTS* φ) $\longleftrightarrow$ ($\forall P$. *finite P* $\longrightarrow$ *wf-interp-for-formula* (*w*, *Inr P* # *I*) φ)
$\langle$*proof*$\rangle$

lemma *wf-interp-for-formula-any-Inr*: *wf-interp-for-formula* (*w*, *Inr P* # *I*) $\varphi \Longrightarrow$
$\forall P$. *finite P* $\longrightarrow$ *wf-interp-for-formula* (*w*, *Inr P* # *I*) φ
$\langle$*proof*$\rangle$

lemma *wf-interp-for-formula-FOr*:
wf-interp-for-formula (*w*, *I*) (*FOr* $\varphi 1$ $\varphi 2$) =
(*wf-interp-for-formula* (*w*, *I*) $\varphi 1$ $\wedge$ *wf-interp-for-formula* (*w*, *I*) $\varphi 2$)
$\langle$*proof*$\rangle$

lemma *wf-interp-for-formula-FAnd*:
wf-interp-for-formula (*w*, *I*) (*FAnd* $\varphi 1$ $\varphi 2$) =
(*wf-interp-for-formula* (*w*, *I*) $\varphi 1$ $\wedge$ *wf-interp-for-formula* (*w*, *I*) $\varphi 2$)
$\langle$*proof*$\rangle$

lemma *enc-wf-interp*:
$[\![$*wf-formula* (*length I*) φ; *wf-interp-for-formula* (*w*, *I*) φ; $x \in$ *enc* (*w*, *I*)$]\!] \Longrightarrow$
wf-interp-for-formula (*dec-word* (*x* @$-$ *sconst* (*any*, *replicate* (*length I*) *False*)),

stream-dec (*length* I) (*FOV* φ) (x @− *sconst* (*any*, *replicate* (*length* I) *False*)))
φ
⟨*proof*⟩

lemma *enc-atom-welldef*: $\forall\, x\ a.$ *enc-atom* $I\ x\ a$ = *enc-atom* $I'\ x\ a \Longrightarrow m <$ *length* $I \Longrightarrow$
(*case* (I ! m, I' ! m) *of* (*Inl* p, *Inl* q) $\Rightarrow p = q$ | (*Inr* P, *Inr* Q) $\Rightarrow P = Q$ | - $\Rightarrow$ *True*)
⟨*proof*⟩

lemma *stream-enc-welldef*: ⟦*stream-enc* (w, I) = *stream-enc* (w', I'); *wf-formula* (*length* I) φ;
wf-interp-for-formula (w, I) φ; *wf-interp-for-formula* (w', I') φ⟧ $\Longrightarrow$
(w, I) $\models \varphi \longleftrightarrow$ (w', I') $\models \varphi$
⟨*proof*⟩

lemma $lang_{WS1S}$-*FOr*:
assumes *wf-formula* n (*FOr* φ_1 φ_2)
shows $lang_{WS1S}$ n (*FOr* φ_1 φ_2) $\subseteq$
($lang_{WS1S}$ n φ_1 $\cup$ $lang_{WS1S}$ n φ_2) $\cap \bigcup\{$*enc* (w, I) | w I. *length* $I = n \wedge$ *wf-interp-for-formula* (w, I) (*FOr* φ_1 φ_2)$\}$
(**is** - $\subseteq$ (*?L1* $\cup$ *?L2*) $\cap$ *?ENC*)
⟨*proof*⟩

lemma $lang_{WS1S}$-*FAnd*:
assumes *wf-formula* n (*FAnd* φ_1 φ_2)
shows $lang_{WS1S}$ n (*FAnd* φ_1 φ_2) $\subseteq$
$lang_{WS1S}$ n φ_1 $\cap$ $lang_{WS1S}$ n φ_2 $\cap \bigcup\{$*enc* (w, I) | w I. *length* $I = n \wedge$ *wf-interp-for-formula* (w, I) (*FAnd* φ_1 φ_2)$\}$
⟨*proof*⟩

13.3 From WS1S to Regular expressions

fun *rexp-of* :: *nat* $\Rightarrow$ '*a formula* $\Rightarrow$ ('*a atom*) *rexp* **where**
rexp-of n (*FQ* a m) =
Inter (*TIMES* [*rexp.Not Zero*, *Atom* (*AQ* m a), *rexp.Not Zero*])
(*ENC* n (*FOV* (*FQ* a m)))
| *rexp-of* n (*FLess m1 m2*) = (*if m1 = m2 then Zero else*
Inter (*TIMES* [*rexp.Not Zero*, *Atom* (*Arbitrary-Except m1 True*),
rexp.Not Zero, *Atom* (*Arbitrary-Except m2 True*),
rexp.Not Zero]) (*ENC* n (*FOV* (*FLess m1 m2* :: '*a formula*))))
| *rexp-of* n (*FIn* m M) =
Inter (*TIMES* [*rexp.Not Zero*, *Atom* (*Arbitrary-Except2* m M), *rexp.Not Zero*])
(*ENC* n (*FOV* (*FIn* m M :: '*a formula*)))
| *rexp-of* n (*FNot* φ) = *Inter* (*rexp.Not* (*rexp-of* n φ)) (*ENC* n (*FOV* (*FNot* φ)))
| *rexp-of* n (*FOr* φ_1 φ_2) = *Inter* (*Plus* (*rexp-of* n φ_1) (*rexp-of* n φ_2)) (*ENC* n (*FOV* (*FOr* φ_1 φ_2)))
| *rexp-of* n (*FAnd* φ_1 φ_2) = *INTERSECT* [*rexp-of* n φ_1, *rexp-of* n φ_2, *ENC* n (*FOV* (*FAnd* φ_1 φ_2))]

| *rexp-of n* (*FExists* φ) = *samequot-exec* (*any*, *replicate n False*) (*Pr* (*rexp-of* (*n* + *1*) φ))
| *rexp-of n* (*FEXISTS* φ) = *samequot-exec* (*any*, *replicate n False*) (*Pr* (*rexp-of* (*n* + *1*) φ))

fun *rexp-of-alt* :: *nat* ⇒ *'a formula* ⇒ (*'a atom*) *rexp* **where**
rexp-of-alt n (*FQ a m*) =
TIMES [*rexp.Not Zero*, *Atom* (*AQ m a*), *rexp.Not Zero*]
| *rexp-of-alt n* (*FLess m1 m2*) = (*if m1* = *m2 then Zero else*
TIMES [*rexp.Not Zero*, *Atom* (*Arbitrary-Except m1 True*),
rexp.Not Zero, *Atom* (*Arbitrary-Except m2 True*),
rexp.Not Zero])
| *rexp-of-alt n* (*FIn m M*) =
TIMES [*rexp.Not Zero*, *Atom* (*Arbitrary-Except2 m M*), *rexp.Not Zero*]
| *rexp-of-alt n* (*FNot* φ) = *rexp.Not* (*rexp-of-alt n* φ)
| *rexp-of-alt n* (*FOr* φ_1 φ_2) = *Plus* (*rexp-of-alt n* φ_1) (*rexp-of-alt n* φ_2)
| *rexp-of-alt n* (*FAnd* φ_1 φ_2) = *Inter* (*rexp-of-alt n* φ_1) (*rexp-of-alt n* φ_2)
| *rexp-of-alt n* (*FExists* φ) = *samequot-exec* (*any*, *replicate n False*) (*Pr* (*Inter* (*rexp-of-alt* (*n* + *1*) φ) (*ENC* (*Suc n*) (*FOV* φ))))
| *rexp-of-alt n* (*FEXISTS* φ) = *samequot-exec* (*any*, *replicate n False*) (*Pr* (*Inter* (*rexp-of-alt* (*n* + *1*) φ) (*ENC* (*Suc n*) (*FOV* φ))))

definition *rexp-of′ n* φ = *Inter* (*rexp-of-alt n* φ) (*ENC n* (*FOV* φ))

fun *rexp-of-alt′* :: *nat* ⇒ *'a formula* ⇒ (*'a atom*) *rexp* **where**
rexp-of-alt′ n (*FQ a m*) = *TIMES* [*Full*, *Atom* (*AQ m a*), *Full*]
| *rexp-of-alt′ n* (*FLess m1 m2*) = (*if m1* = *m2 then Zero else*
TIMES [*Full*, *Atom* (*Arbitrary-Except m1 True*), *Full*, *Atom* (*Arbitrary-Except m2 True*), *Full*])
| *rexp-of-alt′ n* (*FIn m M*) = *TIMES* [*Full*, *Atom* (*Arbitrary-Except2 m M*), *Full*]
| *rexp-of-alt′ n* (*FNot* φ) = *rexp.Not* (*rexp-of-alt′ n* φ)
| *rexp-of-alt′ n* (*FOr* φ_1 φ_2) = *Plus* (*rexp-of-alt′ n* φ_1) (*rexp-of-alt′ n* φ_2)
| *rexp-of-alt′ n* (*FAnd* φ_1 φ_2) = *Inter* (*rexp-of-alt′ n* φ_1) (*rexp-of-alt′ n* φ_2)
| *rexp-of-alt′ n* (*FExists* φ) = *samequot-exec* (*any*, *replicate n False*) (*Pr* (*Inter* (*rexp-of-alt′* (*n* + *1*) φ) (*ENC* (*n* + *1*) {*0*})))
| *rexp-of-alt′ n* (*FEXISTS* φ) = *samequot-exec* (*any*, *replicate n False*) (*Pr* (*rexp-of-alt′* (*n* + *1*) φ))

definition *rexp-of″ n* φ = *Inter* (*rexp-of-alt′ n* φ) (*ENC n* (*FOV* φ))

lemma *enc-eqI*:
assumes $x \in$ *enc* (*w*, *I*) $x \in$ *enc* (*w′*, *I′*) *wf-interp-for-formula* (*w*, *I*) φ *wf-interp-for-formula* (*w′*, *I′*) φ
length I = *length I′*
shows *enc* (*w*, *I*) = *enc* (*w′*, *I′*)
⟨*proof*⟩

lemma *enc-eq-welldef*:
⟦*enc* (*w*, *I*) = *enc* (*w′*, *I′*); *wf-formula* (*length I*) φ; *wf-interp-for-formula* (*w*, *I*)

φ ;*wf-interp-for-formula* (w', I') $\varphi$$\rrbracket \Longrightarrow$
$(w, I) \models \varphi \longleftrightarrow (w', I') \models \varphi$
⟨*proof*⟩

lemma *enc-welldef*:
$\llbracket x \in$ *enc* (w, I); $x \in$ *enc* (w', I'); *length* I = *length* I'; *wf-formula* (*length* I) φ;
wf-interp-for-formula (w, I) φ ;*wf-interp-for-formula* (w', I') $\varphi$$\rrbracket \Longrightarrow$
$(w, I) \models \varphi \longleftrightarrow (w', I') \models \varphi$
⟨*proof*⟩

lemma *wf-rexp-of*: *wf-formula* n $\varphi \Longrightarrow$ *wf* n (*rexp-of* n φ)
⟨*proof*⟩

theorem *lang*$_{WS1S}$*-rexp-of*: *wf-formula* n $\varphi \Longrightarrow$ *lang*$_{WS1S}$ n φ = *lang* n (*rexp-of* n φ)
(**is** - $\Longrightarrow$ - = *?L* n φ)
⟨*proof*⟩

lemma *wf-rexp-of-alt*: *wf-formula* n $\varphi \Longrightarrow$ *wf* n (*rexp-of-alt* n φ)
⟨*proof*⟩

lemma *wf-rexp-of'*: *wf-formula* n $\varphi \Longrightarrow$ *wf* n (*rexp-of'* n φ)
⟨*proof*⟩

lemma *wf-rexp-of-alt'*: *wf-formula* n $\varphi \Longrightarrow$ *wf* n (*rexp-of-alt'* n φ)
⟨*proof*⟩

lemma *wf-rexp-of''*: *wf-formula* n $\varphi \Longrightarrow$ *wf* n (*rexp-of''* n φ)
⟨*proof*⟩

lemma *ENC-FNot*: *ENC* n (*FOV* (*FNot* φ)) = *ENC* n (*FOV* φ)
⟨*proof*⟩

lemma *ENC-FAnd*:
wf-formula n (*FAnd* φ ψ) $\Longrightarrow$ *lang* n (*ENC* n (*FOV* (*FAnd* φ ψ))) $\subseteq$ *lang* n (*ENC* n (*FOV* φ)) $\cap$ *lang* n (*ENC* n (*FOV* ψ))
⟨*proof*⟩

lemma *ENC-FOr*:
wf-formula n (*FOr* φ ψ) $\Longrightarrow$ *lang* n (*ENC* n (*FOV* (*FOr* φ ψ))) $\subseteq$ *lang* n (*ENC* n (*FOV* φ)) $\cap$ *lang* n (*ENC* n (*FOV* ψ))
⟨*proof*⟩

lemma *ENC-FExists*:
wf-formula n (*FExists* φ) $\Longrightarrow$ *lang* n (*ENC* n (*FOV* (*FExists* φ))) =
SAMEQUOT (*any*, *replicate n False*) (*map* π ‘ *lang* (*Suc n*) (*ENC* (*Suc n*) (*FOV* φ))) (**is** - $\Longrightarrow$ *?L* = *?R*)
⟨*proof*⟩

lemma *ENC-FEXISTS*:
wf-formula n (*FEXISTS* φ) $\Longrightarrow$ *lang n* (*ENC n* (*FOV* (*FEXISTS* φ))) =
SAMEQUOT (*any*, *replicate n False*) (*map* π ‘ *lang* (*Suc n*) (*ENC* (*Suc n*) (*FOV* φ))) (**is** - $\Longrightarrow$ *?L* = *?R*)
⟨*proof*⟩

lemma $lang_{WS1S}$*-rexp-of-rexp-of'*:
wf-formula n $\varphi \Longrightarrow$ *lang n* (*rexp-of n* φ) = *lang n* (*rexp-of' n* φ)
⟨*proof*⟩

lemma *SAMEQUTO-UN*[*simp*]: *SAMEQUOT x* ($\bigcup y \in A.\ B\ y$) = ($\bigcup y \in A.$ *SAMEQUOT x* (*B y*))
⟨*proof*⟩

lemma *finite-positions-in-row*[*simp*]:
$n > 0 \Longrightarrow$ *finite* (*positions-in-row* (*x* @− *sconst* (*any*, *replicate n False*)) *0*)
⟨*proof*⟩

lemma *fin-cut-same-snoc*: *fin-cut-same x* (*xs* @ [*y*]) = (*if x* = *y then fin-cut-same x xs else xs* @ [*y*])
⟨*proof*⟩

lemma *fin-cut-same-idem*: *fin-cut-same x* (*fin-cut-same x xs*) = *fin-cut-same x xs*
⟨*proof*⟩

lemma *cut-same-sconst*: *cut-same x* (*xs* @− *sconst x*) = *fin-cut-same x xs*
⟨*proof*⟩

lemma *length-cut-same*: *length* (*cut-same x s*) = (*LEAST n. sdrop n s* = *sconst x*)
⟨*proof*⟩

lemma *enc-alt*: *wf-interp w I* $\Longrightarrow$
x ∈ *enc* (*w*, *I*) ⟷ *x* @− *sconst* ((*any*, *replicate* (*length I*) *False*)) = *stream-enc* (*w*, *I*)
⟨*proof*⟩

lemma *stream-stream-eqI*: ⟦∀ (-, *x*) ∈ *sset xs*. *x* ≠ []; ∀ (-, *x*) ∈ *sset ys*. *x* ≠ [];
smap (λ(-, *x*). *hd x*) *xs* = *smap* (λ(-, *x*). *hd x*) *ys*; *smap* π *xs* = *smap* π *ys*⟧ $\Longrightarrow$ *xs* = *ys*
⟨*proof*⟩

lemma *project-enc-extend*:
fixes *x I*
defines *n* ≡ *length I*
defines *z* ≡ λ*n*. (*any*, *replicate n False*)
defines *I'* ≡ *Inr* (*positions-in-row* (*x* @− *sconst* (*z* (*Suc n*))) *0*) # *I*
assumes *wf*: *wf-interp w I*

assumes *enc*: $\textit{fin-cut-same}\ (z\ n)\ (\textit{map}\ \pi\ x)\ @\ \textit{replicate}\ m\ (z\ n) \in \textit{enc}\ (w,\ I)$
assumes *nonempty*: $\forall (\text{-},\ x) \in \textit{set}\ x.\ x \neq []$
shows $x \in \textit{enc}\ (w,\ I')$
⟨*proof*⟩

lemma *pred-case-conv*: $x - \textit{Suc}\ 0 = (\textit{case}\ x\ \textit{of}\ 0 \Rightarrow 0 \mid \textit{Suc}\ m \Rightarrow m)$
⟨*proof*⟩

lemma *in-pred-image-iff*: $0 \notin X \Longrightarrow (x \in (\lambda x.\ x - \textit{Suc}\ 0)\ `\ X) = (\textit{Suc}\ x \in X)$
⟨*proof*⟩

lemma *map-project-Int-ENC*:
fixes $X\ Z\ n$
defines $z \equiv (\textit{any},\ \textit{replicate}\ n\ \textit{False})$
assumes $0 \notin X\ X \subseteq \{0\ ..{<}\ n + 1\}\ Z \subseteq \textit{lists}\ ((\textit{set}\ o\ \sigma\ \Sigma)\ (n + 1))$
shows $\textit{SAMEQUOT}\ z\ (\textit{map}\ \pi\ `\ (Z \cap \textit{lang}\ (n + 1)\ (\textit{ENC}\ (n + 1)\ X))) =$
$\textit{SAMEQUOT}\ z\ (\textit{map}\ \pi\ `\ Z) \cap \textit{lang}\ n\ (\textit{ENC}\ n\ ((\lambda x.\ x - 1)\ `\ X))$
⟨*proof*⟩

lemma *lang-ENC-split*:
assumes *finite* $X\ X = Y1 \cup Y2\ n = 0 \lor (\forall p \in X.\ p < n)$
shows $\textit{lang}\ n\ (\textit{ENC}\ n\ X) = \textit{lang}\ n\ (\textit{ENC}\ n\ Y1) \cap \textit{lang}\ n\ (\textit{ENC}\ n\ Y2)$
⟨*proof*⟩

lemma *map-project-ENC*:
fixes n
assumes $X \subseteq \{0\ ..{<}\ n + 1\}\ Z \subseteq \textit{lists}\ ((\textit{set}\ o\ \sigma\ \Sigma)\ (n + 1))$
defines $z \equiv (\textit{any},\ \textit{replicate}\ n\ \textit{False})$
shows $\textit{SAMEQUOT}\ z\ (\textit{map}\ \pi\ `\ (Z \cap \textit{lang}\ (n + 1)\ (\textit{ENC}\ (n + 1)\ X))) =$
$(\textit{if}\ 0 \in X$
$\textit{then}\ \textit{SAMEQUOT}\ z\ (\textit{map}\ \pi\ `\ (Z \cap \textit{lang}\ (n + 1)\ (\textit{ENC}\ (n + 1)\ \{0\}))) \cap \textit{lang}$
$n\ (\textit{ENC}\ n\ ((\lambda x.\ x - 1)\ `\ (X - \{0\})))$
$\textit{else}\ \textit{SAMEQUOT}\ z\ (\textit{map}\ \pi\ `\ Z) \cap \textit{lang}\ n\ (\textit{ENC}\ n\ ((\lambda x.\ x - 1)\ `\ (X - \{0\}))))$
(**is** $?L = (\textit{if}\ \text{-}\ \textit{then}\ ?R1\ \textit{else}\ ?R2)$)
⟨*proof*⟩

lemma $\textit{lang}_{M2L}$*-rexp-of′-rexp-of″*:
$\textit{wf-formula}\ n\ \varphi \Longrightarrow \textit{lang}\ n\ (\textit{rexp-of}'\ n\ \varphi) = \textit{lang}\ n\ (\textit{rexp-of}''\ n\ \varphi)$
⟨*proof*⟩

theorem $\textit{lang}_{WS1S}$*-rexp-of′*: $\textit{wf-formula}\ n\ \varphi \Longrightarrow \textit{lang}_{WS1S}\ n\ \varphi = \textit{lang}\ n\ (\textit{rexp-of}'$
$n\ \varphi)$
⟨*proof*⟩

theorem $\textit{lang}_{WS1S}$*-rexp-of″*: $\textit{wf-formula}\ n\ \varphi \Longrightarrow \textit{lang}_{WS1S}\ n\ \varphi = \textit{lang}\ n\ (\textit{rexp-of}''$
$n\ \varphi)$
⟨*proof*⟩

end

14 Normalization of WS1S Formulas

fun *nNot* **where**
  nNot $(FNot\ \varphi) = \varphi$
| *nNot* $(FAnd\ \varphi 1\ \varphi 2) = FOr\ (nNot\ \varphi 1)\ (nNot\ \varphi 2)$
| *nNot* $(FOr\ \varphi 1\ \varphi 2) = FAnd\ (nNot\ \varphi 1)\ (nNot\ \varphi 2)$
| *nNot* $\varphi = FNot\ \varphi$

primrec *norm* **where**
  norm $(FQ\ a\ m) = FQ\ a\ m$
| *norm* $(FLess\ m\ n) = FLess\ m\ n$
| *norm* $(FIn\ m\ M) = FIn\ m\ M$
| *norm* $(FOr\ \varphi\ \psi) = FOr\ (norm\ \varphi)\ (norm\ \psi)$
| *norm* $(FAnd\ \varphi\ \psi) = FAnd\ (norm\ \varphi)\ (norm\ \psi)$
| *norm* $(FNot\ \varphi) = nNot\ (norm\ \varphi)$
| *norm* $(FExists\ \varphi) = FExists\ (norm\ \varphi)$
| *norm* $(FEXISTS\ \varphi) = FEXISTS\ (norm\ \varphi)$

context *formula*
begin

lemma *satisfies-nNot*[*simp*]: $(w, I) \models nNot\ \varphi \longleftrightarrow (w, I) \models FNot\ \varphi$
  ⟨*proof*⟩

lemma *FOV-nNot*[*simp*]: $FOV\ (nNot\ \varphi) = FOV\ (FNot\ \varphi)$
  ⟨*proof*⟩

lemma *SOV-nNot*[*simp*]: $SOV\ (nNot\ \varphi) = SOV\ (FNot\ \varphi)$
  ⟨*proof*⟩

lemma *pre-wf-formula-nNot*[*simp*]: *pre-wf-formula* $n\ (nNot\ \varphi) =$ *pre-wf-formula* $n\ (FNot\ \varphi)$
  ⟨*proof*⟩

lemma *FOV-norm*[*simp*]: $FOV\ (norm\ \varphi) = FOV\ \varphi$
  ⟨*proof*⟩

lemma *SOV-norm*[*simp*]: $SOV\ (norm\ \varphi) = SOV\ \varphi$
  ⟨*proof*⟩

lemma *pre-wf-formula-norm*[*simp*]: *pre-wf-formula* $n\ (norm\ \varphi) =$ *pre-wf-formula* $n\ \varphi$
  ⟨*proof*⟩

lemma *satisfies-norm*[*simp*]: $wI \models norm\ \varphi \longleftrightarrow wI \models \varphi$
  ⟨*proof*⟩

lemma *lang*$_{WS1S}$*-norm*[*simp*]: *lang*$_{WS1S}$ *n* (*norm* φ) = *lang*$_{WS1S}$ *n* φ
⟨*proof*⟩

end

15 Deciding Equivalence of WS1S Formulas

global-interpretation *embed2 set o* σ Σ *wf-atom* Σ π *lookup* ε Σ *case-prod Singleton*
for Σ :: *'a* :: *linorder list*
defines
$\mathfrak{D}$ = *embed.lderiv lookup* (ε Σ)
and *Co*$\mathfrak{D}$ = *embed.lderiv-dual lookup* (ε Σ)
and *r*$\mathfrak{D}$ = *embed.rderiv lookup* (ε Σ)
and *r*$\mathfrak{D}$*-add* = *embed2.rderiv-and-add lookup* (ε Σ)
and $\mathfrak{Q}$ = *embed2.samequot-exec lookup* (ε Σ) (*case-prod Singleton*)
⟨*proof*⟩

lemma *enum-not-empty*[*simp*]: *Enum.enum* $\neq$ [] (**is** *?enum* $\neq$ [])
⟨*proof*⟩

global-interpretation Φ: *formula Enum.enum* :: *'a* :: {*enum*, *linorder*} *list*
rewrites *embed2.samequot-exec lookup* (ε (*Enum.enum* :: *'a* :: {*enum*, *linorder*} *list*)) (*case-prod Singleton*) = $\mathfrak{Q}$ *Enum.enum*
defines
pre-wf-formula = Φ.*pre-wf-formula*
and *wf-formula* = Φ.*wf-formula*
and *rexp-of* = Φ.*rexp-of*
and *rexp-of-alt* = Φ.*rexp-of-alt*
and *rexp-of-alt'* = Φ.*rexp-of-alt'*
and *rexp-of'* = Φ.*rexp-of'*
and *rexp-of''* = Φ.*rexp-of''*
and *valid-ENC* = Φ.*valid-ENC*
and *ENC* = Φ.*ENC*
and *dec-interp* = Φ.*stream-dec*
and *any* = Φ.*any*
⟨*proof*⟩

lemmas *lang*$_{WS1S}$*-rexp-of-norm* = *trans*[*OF sym*[*OF* Φ.*lang*$_{WS1S}$*-norm*] Φ.*lang*$_{WS1S}$*-rexp-of*]
lemmas *lang*$_{WS1S}$*-rexp-of'-norm* = *trans*[*OF sym*[*OF* Φ.*lang*$_{WS1S}$*-norm*] Φ.*lang*$_{WS1S}$*-rexp-of'*]
lemmas *lang*$_{WS1S}$*-rexp-of''-norm* = *trans*[*OF sym*[*OF* Φ.*lang*$_{WS1S}$*-norm*] Φ.*lang*$_{WS1S}$*-rexp-of''*]

⟨*ML*⟩

global-interpretation *D*: *rexp-DFA* σ Σ *wf-atom* Σ π *lookup* λx. «*pnorm* (*inorm* *x*)»

$\lambda a\ r.$ «$\mathfrak{D}\ \Sigma\ a\ r$» *final alphabet.wf* (*wf-atom* Σ) *n pnorm lang* Σ *n n*
for Σ :: *'a* :: *linorder list* **and** *n* :: *nat*
defines
test = *rexp-DA.test* (*final* :: *'a atom rexp* $\Rightarrow$ *bool*)
and *step* = *rexp-DA.step* ($\sigma\ \Sigma$) ($\lambda a\ r.$ «$\mathfrak{D}\ \Sigma\ a\ r$») *pnorm n*
and *closure* = *rexp-DA.closure* ($\sigma\ \Sigma$) ($\lambda a\ r.$ «$\mathfrak{D}\ \Sigma\ a\ r$») *final pnorm n*
and *check-eqvRE* = *rexp-DA.check-eqv* ($\sigma\ \Sigma$) ($\lambda x.$ «*pnorm* (*inorm x*)») ($\lambda a\ r.$ «$\mathfrak{D}\ \Sigma\ a\ r$») *final pnorm n*
and *test-invariant* = *rexp-DA.test-invariant* (*final* :: *'a atom rexp* $\Rightarrow$ *bool*) :: ((*'a* $\times$ *bool list*) *list* $\times$ -) *list* $\times$ - $\Rightarrow$ *bool*
and *step-invariant* = *rexp-DA.step-invariant* ($\sigma\ \Sigma$) ($\lambda a\ r.$ «$\mathfrak{D}\ \Sigma\ a\ r$») *pnorm n*
and *closure-invariant* = *rexp-DA.closure-invariant* ($\sigma\ \Sigma$) ($\lambda a\ r.$ «$\mathfrak{D}\ \Sigma\ a\ r$») *final pnorm n*
and *counterexampleRE* = *rexp-DA.counterexample* ($\sigma\ \Sigma$) ($\lambda x.$ «*pnorm* (*inorm x*)») ($\lambda a\ r.$ «$\mathfrak{D}\ \Sigma\ a\ r$») *final pnorm n*
and *reachable* = *rexp-DA.reachable* ($\sigma\ \Sigma$) ($\lambda x.$ «*pnorm* (*inorm x*)») ($\lambda a\ r.$ «$\mathfrak{D}\ \Sigma\ a\ r$») *pnorm n*
and *automaton* = *rexp-DA.automaton* ($\sigma\ \Sigma$) ($\lambda x.$ «*pnorm* (*inorm x*)») ($\lambda a\ r.$ «$\mathfrak{D}\ \Sigma\ a\ r$») *pnorm n*
⟨*proof*⟩

definition *check-eqv* **where**
check-eqv n $\varphi\ \psi \longleftrightarrow$ *wf-formula n* (*FOr* $\varphi\ \psi$) $\wedge$
slow.check-eqvRE Enum.enum n (*rexp-of''* *n* (*norm* φ)) (*rexp-of''* *n* (*norm* ψ))

definition *counterexample* **where**
counterexample n $\varphi\ \psi$ =
map-option ($\lambda w.$ *dec-interp n* (*FOV* (*FOr* $\varphi\ \psi$)) (*w* @− *sconst* (*any*, *replicate n False*)))
(*slow.counterexampleRE Enum.enum n* (*rexp-of''* *n* (*norm* φ)) (*rexp-of''* *n* (*norm* ψ)))

lemma *soundness*: *slow.check-eqv n* $\varphi\ \psi \Longrightarrow \Phi.lang_{WS1S}\ n\ \varphi = \Phi.lang_{WS1S}\ n\ \psi$
⟨*proof*⟩

lemma *completeness*:
assumes $\Phi.lang_{WS1S}\ n\ \varphi = \Phi.lang_{WS1S}\ n\ \psi$ *wf-formula n* (*FOr* $\varphi\ \psi$)
shows *slow.check-eqv n* $\varphi\ \psi$
⟨*proof*⟩

⟨*ML*⟩

global-interpretation *D*: *rexp-DA-no-post* $\sigma\ \Sigma$ *wf-atom* $\Sigma\ \pi$ *lookup* $\lambda x.$ *pnorm* (*inorm x*)
$\lambda a\ r.$ *pnorm* ($\mathfrak{D}\ \Sigma\ a\ r$) *final alphabet.wf* (*wf-atom* Σ) *n lang* Σ *n n*
for Σ :: *'a* :: *linorder list* **and** *n* :: *nat*
defines
test = *rexp-DA.test* (*final* :: *'a atom rexp* $\Rightarrow$ *bool*)
and *step* = *rexp-DA.step* ($\sigma\ \Sigma$) ($\lambda a\ r.$ *pnorm* ($\mathfrak{D}\ \Sigma\ a\ r$)) *id n*

and *closure* = *rexp-DA.closure* $(\sigma\ \Sigma)$ $(\lambda a\ r.\ pnorm\ (\mathfrak{D}\ \Sigma\ a\ r))$ *final id n*
and *check-eqvRE* = *rexp-DA.check-eqv* $(\sigma\ \Sigma)$ $(\lambda x.\ pnorm\ (inorm\ x))$ $(\lambda a\ r.\ pnorm\ (\mathfrak{D}\ \Sigma\ a\ r))$ *final id n*
and *test-invariant* = *rexp-DA.test-invariant* (*final* :: *'a atom rexp* $\Rightarrow$ *bool*) ::
(('a $\times$ *bool list*) *list* $\times$ -) *list* $\times$ - $\Rightarrow$ *bool*
and *step-invariant* = *rexp-DA.step-invariant* $(\sigma\ \Sigma)$ $(\lambda a\ r.\ pnorm\ (\mathfrak{D}\ \Sigma\ a\ r))$ *id n*
and *closure-invariant* = *rexp-DA.closure-invariant* $(\sigma\ \Sigma)$ $(\lambda a\ r.\ pnorm\ (\mathfrak{D}\ \Sigma\ a\ r))$ *final id n*
and *counterexampleRE* = *rexp-DA.counterexample* $(\sigma\ \Sigma)$ $(\lambda x.\ pnorm\ (inorm\ x))$ $(\lambda a\ r.\ pnorm\ (\mathfrak{D}\ \Sigma\ a\ r))$ *final id n*
and *reachable* = *rexp-DA.reachable* $(\sigma\ \Sigma)$ $(\lambda x.\ pnorm\ (inorm\ x))$ $(\lambda a\ r.\ pnorm\ (\mathfrak{D}\ \Sigma\ a\ r))$ *id n*
and *automaton* = *rexp-DA.automaton* $(\sigma\ \Sigma)$ $(\lambda x.\ pnorm\ (inorm\ x))$ $(\lambda a\ r.\ pnorm\ (\mathfrak{D}\ \Sigma\ a\ r))$ *id n*
⟨*proof*⟩

definition *check-eqv* **where**
check-eqv n $\varphi\ \psi \longleftrightarrow$ *wf-formula n* (*FOr* $\varphi\ \psi$) $\wedge$
fast.check-eqvRE Enum.enum n (*rexp-of''* *n* (*norm* φ)) (*rexp-of''* *n* (*norm* ψ))

definition *counterexample* **where**
counterexample n $\varphi\ \psi$ =
map-option ($\lambda w.$ *dec-interp n* (*FOV* (*FOr* $\varphi\ \psi$)) (*w* @− *sconst* (*any*, *replicate n False*)))
(*fast.counterexampleRE Enum.enum n* (*rexp-of''* *n* (*norm* φ)) (*rexp-of''* *n* (*norm* ψ)))

lemma *soundness*: *fast.check-eqv n* $\varphi\ \psi \Longrightarrow \Phi.lang_{WS1S}\ n\ \varphi = \Phi.lang_{WS1S}\ n\ \psi$
⟨*proof*⟩

⟨*ML*⟩

global-interpretation *D*: *rexp-DA-no-post* $\sigma\ \Sigma$ *wf-atom* $\Sigma\ \pi$ *lookup*
$\lambda x.$ *pnorm-dual* (*rexp-dual-of* (*inorm x*)) $\lambda a\ r.$ *pnorm-dual* ($Co\mathfrak{D}\ \Sigma\ a\ r$) *final-dual alphabet.wf-dual* (*wf-atom* Σ) *n lang-dual* Σ *n n*
for Σ :: *'a* :: *linorder list* **and** *n* :: *nat*
defines
test = *rexp-DA.test* (*final-dual* :: *'a atom rexp-dual* $\Rightarrow$ *bool*)
and *step* = *rexp-DA.step* $(\sigma\ \Sigma)$ $(\lambda a\ r.\ pnorm\text{-}dual\ (Co\mathfrak{D}\ \Sigma\ a\ r))$ *id n*
and *closure* = *rexp-DA.closure* $(\sigma\ \Sigma)$ $(\lambda a\ r.\ pnorm\text{-}dual\ (Co\mathfrak{D}\ \Sigma\ a\ r))$ *final-dual id n*
and *check-eqvRE* = *rexp-DA.check-eqv* $(\sigma\ \Sigma)$ $(\lambda x.\ pnorm\text{-}dual\ (rexp\text{-}dual\text{-}of\ (inorm\ x)))$ $(\lambda a\ r.\ pnorm\text{-}dual\ (Co\mathfrak{D}\ \Sigma\ a\ r))$ *final-dual id n*
and *test-invariant* = *rexp-DA.test-invariant* (*final-dual* :: *'a atom rexp-dual* $\Rightarrow$ *bool*) ::
(('a $\times$ *bool list*) *list* $\times$ -) *list* $\times$ - $\Rightarrow$ *bool*
and *step-invariant* = *rexp-DA.step-invariant* $(\sigma\ \Sigma)$ $(\lambda a\ r.\ pnorm\text{-}dual\ (Co\mathfrak{D}\ \Sigma\ a\ r))$ *id n*

and *closure-invariant* = *rexp-DA.closure-invariant* $(\sigma\ \Sigma)$ $(\lambda a\ r.\ \textit{pnorm-dual}\ (\textit{Co}\mathfrak{D}\ \Sigma\ a\ r))$ *final-dual id n*
and *counterexampleRE* = *rexp-DA.counterexample* $(\sigma\ \Sigma)$ $(\lambda x.\ \textit{pnorm-dual}\ (\textit{rexp-dual-of}\ (\textit{inorm}\ x)))$ $(\lambda a\ r.\ \textit{pnorm-dual}\ (\textit{Co}\mathfrak{D}\ \Sigma\ a\ r))$ *final-dual id n*
and *reachable* = *rexp-DA.reachable* $(\sigma\ \Sigma)$ $(\lambda x.\ \textit{pnorm-dual}\ (\textit{rexp-dual-of}\ (\textit{inorm}\ x)))$ $(\lambda a\ r.\ \textit{pnorm-dual}\ (\textit{Co}\mathfrak{D}\ \Sigma\ a\ r))$ *id n*
and *automaton* = *rexp-DA.automaton* $(\sigma\ \Sigma)$ $(\lambda x.\ \textit{pnorm-dual}\ (\textit{rexp-dual-of}\ (\textit{inorm}\ x)))$ $(\lambda a\ r.\ \textit{pnorm-dual}\ (\textit{Co}\mathfrak{D}\ \Sigma\ a\ r))$ *id n*
⟨*proof*⟩

definition *check-eqv* **where**
$\textit{check-eqv}\ n\ \varphi\ \psi \longleftrightarrow \textit{wf-formula}\ n\ (\textit{FOr}\ \varphi\ \psi) \wedge$
$\textit{dual.check-eqvRE}\ \textit{Enum.enum}\ n\ (\textit{rexp-of}''\ n\ (\textit{norm}\ \varphi))\ (\textit{rexp-of}''\ n\ (\textit{norm}\ \psi))$

definition *counterexample* **where**
$\textit{counterexample}\ n\ \varphi\ \psi =$
$\textit{map-option}\ (\lambda w.\ \textit{dec-interp}\ n\ (\textit{FOV}\ (\textit{FOr}\ \varphi\ \psi))\ (w\ @-\ \textit{sconst}\ (\textit{any}, \textit{replicate}\ n\ \textit{False})))$
$(\textit{dual.counterexampleRE}\ \textit{Enum.enum}\ n\ (\textit{rexp-of}''\ n\ (\textit{norm}\ \varphi))\ (\textit{rexp-of}''\ n\ (\textit{norm}\ \psi)))$

lemma *soundness*: $\textit{dual.check-eqv}\ n\ \varphi\ \psi \Longrightarrow \Phi.\textit{lang}_{WS1S}\ n\ \varphi = \Phi.\textit{lang}_{WS1S}\ n\ \psi$
⟨*proof*⟩

⟨*ML*⟩

References

[1] A. Krauss and T. Nipkow. Regular sets and expressions. *Archive of Formal Proofs*, 2010. http://isa-afp.org/entries/Regular-Sets.shtml, Formal proof development.

[2] D. Traytel and T. Nipkow. Verified decision procedures for MSO on words based on derivatives of regular expressions. In G. Morrisett and T. Uustalu, editors, *Proc. Int. Conf. Functional Programming, ICFP 2013*, pages 3–12. ACM, 2013.