

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

Dmitriy Traytel and Tobias Nipkow

June 11, 2019

## 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

<b>1</b>	<b>Regular Sets</b>	<b>3</b>
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 . . . . .	10
1.7	Lists of Fixed Length . . . . .	10
<b>2</b>	<b>II-Extended Regular Expressions</b>	<b>10</b>
2.1	Syntax of regular expressions . . . . .	10
2.2	ACI normalization . . . . .	11

2.3	Finality . . . . .	14
2.4	Wellformedness w.r.t. an alphabet . . . . .	15
2.5	Language . . . . .	16
<b>3</b>	<b>Derivatives of <math>\Pi</math>-Extended Regular Expressions</b>	<b>17</b>
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 . . . . .	21
<b>4</b>	<b>Some Useful Regular Operators</b>	<b>21</b>
4.1	Quotienting by the same letter . . . . .	24
4.2	Suffix and Prefix Languages . . . . .	27
<b>5</b>	<b><math>\Pi</math>-Extended Dual Regular Expressions</b>	<b>28</b>
5.1	Syntax of regular expressions . . . . .	28
<b>6</b>	<b>Deciding Equivalence of <math>\Pi</math>-Extended Regular Expressions</b>	<b>33</b>
<b>7</b>	<b>Initial Normalization of the Input</b>	<b>41</b>
<b>8</b>	<b>Partial Derivatives-like Normalization</b>	<b>46</b>
<b>9</b>	<b>Monadic Second-Order Logic Formulas</b>	<b>48</b>
9.1	Interpretations and Encodings . . . . .	48
9.2	Syntax and Semantics of MSO . . . . .	48
9.3	ENC . . . . .	50
<b>10</b>	<b>M2L</b>	<b>52</b>
10.1	Encodings . . . . .	52
10.2	Welldefinedness of enc wrt. Models . . . . .	55
10.3	From M2L to Regular expressions . . . . .	57
<b>11</b>	<b>Normalization of M2L Formulas</b>	<b>60</b>
<b>12</b>	<b>Deciding Equivalence of M2L Formulas</b>	<b>61</b>
<b>13</b>	<b>WS1S</b>	<b>64</b>
13.1	Encodings . . . . .	64
13.2	Welldefinedness of enc wrt. Models . . . . .	70
13.3	From WS1S to Regular expressions . . . . .	72
<b>14</b>	<b>Normalization of WS1S Formulas</b>	<b>76</b>
<b>15</b>	<b>Deciding Equivalence of WS1S Formulas</b>	<b>77</b>

# 1 Regular Sets

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

**definition** conc :: 'a lang  $\Rightarrow$  'a lang  $\Rightarrow$  'a lang (infixr @@ 75) **where**  
 $A @@ B = \{xs@ys \mid xs \text{ ys. } xs:A \ \& \ ys:B\}$

**lemma** [code]:

$A @@ B = (\%(xs, ys). xs @ ys) ' (A \times B)$   
(proof)

**overloading** word-pow == compow :: nat  $\Rightarrow$  'a list  $\Rightarrow$  'a list

**begin**

**primrec** word-pow :: nat  $\Rightarrow$  'a list  $\Rightarrow$  'a list **where**

word-pow 0 w = [] |

word-pow (Suc n) w = w @ word-pow n w

**end**

**overloading** lang-pow == compow :: nat  $\Rightarrow$  'a lang  $\Rightarrow$  'a lang

**begin**

**primrec** lang-pow :: nat  $\Rightarrow$  'a lang  $\Rightarrow$  'a lang **where**

lang-pow 0 A = {[]} |

lang-pow (Suc n) A = A @@ (lang-pow n A)

**end**

**lemma** word-pow-alt: compow n w = concat (replicate n w)

(proof)

**definition** star :: 'a lang  $\Rightarrow$  'a lang **where**

star A = ( $\bigcup n. A \hat{\ } n$ )

## 1.1 Concatenation of Languages

**lemma** concI[simp,intro]:  $u : A \Longrightarrow v : B \Longrightarrow u@v : A @@ B$

(proof)

**lemma** concE[elim]:

**assumes**  $w \in A @@ B$

**obtains**  $u v$  **where**  $u \in A \ v \in B \ w = u@v$

(proof)

**lemma** conc-mono:  $A \subseteq C \Longrightarrow B \subseteq D \Longrightarrow A @@ B \subseteq C @@ D$

(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**  $A @@ (B \cup C) = A @@ B \cup A @@ C$   
**and**  $(A \cup B) @@ C = A @@ C \cup B @@ C$   
 ⟨proof⟩

**lemma conc-UNION-distrib:**  
**shows**  $A @@ \bigcup (M \text{ ' } I) = \bigcup ((\%i. A @@ M i) \text{ ' } I)$   
**and**  $\bigcup (M \text{ ' } I) @@ A = \bigcup ((\%i. M i @@ A) \text{ ' } I)$   
 ⟨proof⟩

**lemma hom-image-conc:**  $[\bigwedge xs\ ys. f (xs @ ys) = f xs @ f ys] \implies f \text{ ' } (A @@ B) = f \text{ ' } A @@ f \text{ ' } B$   
 ⟨proof⟩

**lemma map-image-conc[simp]:**  $map f \text{ ' } (A @@ B) = map f \text{ ' } A @@ map f \text{ ' } B$   
 ⟨proof⟩

**lemma conc-subset-lists:**  $A \subseteq lists\ S \implies B \subseteq lists\ S \implies A @@ B \subseteq lists\ S$   
 ⟨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*  $w : A. length\ w \leq k \implies w : A ^^ n \implies length\ w \leq k * n$   
 ⟨proof⟩

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

**lemma** *lang-pow-subset-lists*:  $A \subseteq \text{lists } S \implies A^{\wedge n} \subseteq \text{lists } S$   
*<proof>*

**lemma** *star-subset-lists*:  $A \subseteq \text{lists } S \implies \text{star } A \subseteq \text{lists } S$   
*<proof>*

**lemma** *star-if-lang-pow[simp]*:  $w : A^{\wedge n} \implies w : \text{star } A$   
*<proof>*

**lemma** *Nil-in-star[iff]*:  $[] : \text{star } A$   
*<proof>*

**lemma** *star-if-lang[simp]*: **assumes**  $w : A$  **shows**  $w : \text{star } A$   
*<proof>*

**lemma** *append-in-starI[simp]*:  
**assumes**  $u : \text{star } A$  **and**  $v : \text{star } A$  **shows**  $u@v : \text{star } A$   
*<proof>*

**lemma** *conc-star-star*:  $\text{star } A @@ \text{star } A = \text{star } A$   
*<proof>*

**lemma** *conc-star-comm*:  
**shows**  $A @@ \text{star } A = \text{star } A @@ A$   
*<proof>*

**lemma** *star-induct[consumes 1, case-names Nil append, induct set: star]*:  
**assumes**  $w : \text{star } A$   
**and**  $P []$   
**and** *step*:  $!!u v. u : A \implies v : \text{star } A \implies P v \implies P (u@v)$   
**shows**  $P w$   
*<proof>*

**lemma** *star-empty[simp]*:  $\text{star } \{\} = \{[]\}$   
*<proof>*

**lemma** *star-epsilon[simp]*:  $\text{star } \{[]\} = \{[]\}$   
*<proof>*

**lemma** *star-idemp[simp]*:  $\text{star } (\text{star } A) = \text{star } A$   
*<proof>*

**lemma** *star-unfold-left*:  $\text{star } A = A @@ \text{star } A \cup \{[]\}$  (**is**  $?L = ?R$ )  
*<proof>*

**lemma** *concat-in-star*:  $\text{set } ws \subseteq A \implies \text{concat } ws : \text{star } A$   
*<proof>*

**lemma** *in-star-iff-concat*:

$w : \text{star } A = (\text{EX } ws. \text{set } ws \subseteq A \ \& \ w = \text{concat } ws \ \& \ [] \notin \text{set } ws)$

(**is**  $- = (\text{EX } ws. ?R \ w \ ws)$ )

$\langle \text{proof} \rangle$

**lemma** *star-conv-concat*:  $\text{star } A = \{\text{concat } ws \mid ws. \text{set } ws \subseteq A \ \& \ [] \notin \text{set } ws\}$

$\langle \text{proof} \rangle$

**lemma** *star-insert-eps[simp]*:  $\text{star } (\text{insert } [] \ A) = \text{star } (A)$

$\langle \text{proof} \rangle$

**lemma** *star-decom*:

**assumes**  $a: x \in \text{star } A \ x \neq []$

**shows**  $\exists a \ b. x = a \ @ \ b \ \wedge \ a \neq [] \ \wedge \ a \in A \ \wedge \ b \in \text{star } A$

$\langle \text{proof} \rangle$

**lemma** *Ball-starI*:  $\forall a \in \text{set } as. [a] \in A \implies as \in \text{star } A$

$\langle \text{proof} \rangle$

**lemma** *map-image-star[simp]*:  $\text{map } f \ ' \ \text{star } A = \text{star } (\text{map } f \ ' \ A)$

$\langle \text{proof} \rangle$

### 1.3 Left-Quotients of Languages

**definition** *lQuot* ::  $'a \Rightarrow 'a \ \text{lang} \Rightarrow 'a \ \text{lang}$

**where**  $lQuot \ x \ A = \{ xs. x \# xs \in A \}$

**definition** *lQuots* ::  $'a \ \text{list} \Rightarrow 'a \ \text{lang} \Rightarrow 'a \ \text{lang}$

**where**  $lQuots \ xs \ A = \{ ys. xs \ @ \ ys \in A \}$

**abbreviation**

$lQuotss \ :: \ 'a \ \text{list} \Rightarrow 'a \ \text{lang} \ \text{set} \Rightarrow 'a \ \text{lang}$

**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]*:  $lQuot \ a \ \{[b]\} = (\text{if } a = b \ \text{then } \{\} \ \text{else } \{\})$

**and** *lQuot-chars[simp]*:  $lQuot \ a \ \{[b] \mid b. P \ b\} = (\text{if } P \ a \ \text{then } \{\} \ \text{else } \{\})$

**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]*:  $lQuot \ a \ (-A) = - \ lQuot \ a \ A$

$\langle \text{proof} \rangle$

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

$\langle \text{proof} \rangle$

**lemma** *lQuot-conc [simp]*:  $lQuot \ c \ (A \ @ \ B) = (lQuot \ c \ A) \ @ \ B \cup (\text{if } [] \in A$

then  $lQuot\ c\ B$  else  $\{\}$   
 ⟨proof⟩

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

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

**lemma**  $lQuot\text{-}lists$  [simp]:  $c : S \implies lQuot\ c\ (lists\ S) = lists\ S$   
 ⟨proof⟩

**lemma**  $lQuots\text{-}simps$  [simp]:  
 shows  $lQuots\ []\ A = A$   
 and  $lQuots\ (c\ \#\ s)\ A = lQuots\ s\ (lQuot\ c\ A)$   
 and  $lQuots\ (s1\ @\ s2)\ A = lQuots\ s2\ (lQuots\ s1\ A)$   
 ⟨proof⟩

**lemma**  $lQuots\text{-}append$  [iff]:  $v \in lQuots\ w\ A \longleftrightarrow w\ @\ v \in A$   
 ⟨proof⟩

## 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**  $rQuot\text{-}rev\text{-}lQuot$ :  $rQuot\ x\ A = rev\ `lQuot\ x\ (rev\ `A)$   
 ⟨proof⟩

**lemma**  $rQuots\text{-}rev\text{-}lQuots$ :  $rQuots\ x\ A = rev\ `lQuots\ x\ (rev\ `A)$   
 ⟨proof⟩

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

**lemma**  $lQuot\text{-}rQuot$ :  $lQuot\ a\ (rQuot\ b\ A) = rQuot\ b\ (lQuot\ a\ A)$

*<proof>*

**lemma** *rQuot-lQuot*:  $rQuot\ a\ (lQuot\ b\ A) = lQuot\ b\ (rQuot\ a\ A)$   
*<proof>*

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

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

**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]*:  $c : S \implies rQuot\ c\ (lists\ S) = lists\ S$   
*<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]*:  $v \in \text{biQuots } u \ w \ A \longleftrightarrow u \ @ \ v \ @ \ \text{rev } w \in A$   
*<proof>*

## 1.6 Arden's Lemma

**lemma** *arden-helper*:

**assumes** *eq*:  $X = A \ @@ \ X \cup B$

**shows**  $X = (A \ ^{\wedge} \text{Suc } n) \ @@ \ X \cup (\bigcup m \leq n. (A \ ^{\wedge} m) \ @@ \ B)$   
*<proof>*

**lemma** *Arden*:

**assumes**  $\square \notin A$

**shows**  $X = A \ @@ \ X \cup B \longleftrightarrow X = \text{star } A \ @@ \ B$   
*<proof>*

**lemma** *reversed-arden-helper*:

**assumes** *eq*:  $X = X \ @@ \ A \cup B$

**shows**  $X = X \ @@ \ (A \ ^{\wedge} \text{Suc } n) \cup (\bigcup m \leq n. B \ @@ \ (A \ ^{\wedge} m))$   
*<proof>*

**theorem** *reversed-Arden*:

**assumes** *nemp*:  $\square \notin A$

**shows**  $X = X \ @@ \ A \cup B \longleftrightarrow X = B \ @@ \ \text{star } A$   
*<proof>*

## 1.7 Lists of Fixed Length

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

**lemma** *tl-listsN*:  $A \subseteq \text{listsN } (n + 1) \ S \implies \text{tl } 'A \subseteq \text{listsN } n \ S$   
*<proof>*

**lemma** *map-tl-listsN*:  $A \subseteq \text{lists } (\text{listsN } (n + 1) \ S) \implies \text{map } \text{tl } 'A \subseteq \text{lists } (\text{listsN } n \ S)$   
*<proof>*

## 2 $\Pi$ -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 OPERA-
TION N xs)

```

**abbreviation** PLUS  $\equiv$  rexp-of-list Plus Zero

**abbreviation** TIMES  $\equiv$  rexp-of-list Times One

**abbreviation** INTERSECT  $\equiv$  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:  $\bigwedge x y xs. P (y \# xs) \implies P (x \# (y \# xs))$ 
  shows P xs
  <proof>

```

## 2.2 ACI normalization

```

fun toplevel-summands :: 'a rexp  $\Rightarrow$  'a rexp set where
  toplevel-summands (Plus r s) = toplevel-summands r  $\cup$  toplevel-summands s
| toplevel-summands r = {r}

```

**abbreviation** (input) flatten LISTOP X  $\equiv$  LISTOP (sorted-list-of-set X)

**lemma** toplevel-summands-nonempty[simp]:

```

  toplevel-summands r  $\neq$  {}
  <proof>

```

**lemma** toplevel-summands-finite[simp]:

```

  finite (toplevel-summands r)
  <proof>

```

**primrec** ACI-norm :: ('a::linorder) rexp  $\Rightarrow$  'a rexp ( $\ll\!-\!\gg$ ) **where**

```

   $\ll\!Zero\gg$  = Zero
|  $\ll\!Full\gg$  = Full
|  $\ll\!One\gg$  = One
|  $\ll\!Atom\ a\gg$  = Atom a
|  $\ll\!Plus\ r\ s\gg$  = flatten PLUS (toplevel-summands (Plus  $\ll\!r\gg$   $\ll\!s\gg$ ))
|  $\ll\!Times\ r\ s\gg$  = Times  $\ll\!r\gg$   $\ll\!s\gg$ 
|  $\ll\!Star\ r\gg$  = Star  $\ll\!r\gg$ 

```

|  $\langle\langle \text{Not } r \rangle\rangle = \text{Not } \langle r \rangle$   
|  $\langle\langle \text{Inter } r \ s \rangle\rangle = \text{Inter } \langle r \rangle \ \langle s \rangle$   
|  $\langle\langle \text{Pr } r \rangle\rangle = \text{Pr } \langle r \rangle$

**lemma** *Plus-toplevel-summands*:

$\text{Plus } r \ s \in \text{toplevel-summands } t \implies \text{False}$   
 $\langle \text{proof} \rangle$

**lemma** *toplevel-summands-not-Plus[simp]*:

$(\forall r \ s. x \neq \text{Plus } r \ s) \implies \text{toplevel-summands } x = \{x\}$   
 $\langle \text{proof} \rangle$

**lemma** *toplevel-summands-PLUS-strong*:

$\llbracket xs \neq []; \text{list-all } (\lambda x. \neg(\exists r \ s. x = \text{Plus } r \ s)) \ xs \rrbracket \implies \text{toplevel-summands } (\text{PLUS } xs) = \text{set } xs$   
 $\langle \text{proof} \rangle$

**lemma** *toplevel-summands-flatten*:

$\llbracket X \neq \{\}; \text{finite } X; \forall x \in X. \neg(\exists r \ s. x = \text{Plus } r \ s) \rrbracket \implies \text{toplevel-summands } (\text{flatten } \text{PLUS } X) = X$   
 $\langle \text{proof} \rangle$

**lemma** *ACI-norm-Plus*:

$\langle r \rangle = \text{Plus } s \ t \implies \exists s \ t. r = \text{Plus } s \ t$   
 $\langle \text{proof} \rangle$

**lemma** *toplevel-summands-flatten-ACI-norm-image*:

$\text{toplevel-summands } (\text{flatten } \text{PLUS } (\text{ACI-norm } \text{'toplevel-summands } r)) = \text{ACI-norm } \text{'toplevel-summands } r$   
 $\langle \text{proof} \rangle$

**lemma** *toplevel-summands-flatten-ACI-norm-image-Union*:

$\text{toplevel-summands } (\text{flatten } \text{PLUS } (\text{ACI-norm } \text{'toplevel-summands } r \cup \text{ACI-norm } \text{'toplevel-summands } s)) =$   
 $\text{ACI-norm } \text{'toplevel-summands } r \cup \text{ACI-norm } \text{'toplevel-summands } s$   
 $\langle \text{proof} \rangle$

**lemma** *toplevel-summands-ACI-norm*:

$\text{toplevel-summands } \langle r \rangle = \text{ACI-norm } \text{'toplevel-summands } r$   
 $\langle \text{proof} \rangle$

**lemma** *ACI-norm-flatten*:

$\langle r \rangle = \text{flatten } \text{PLUS } (\text{ACI-norm } \text{'toplevel-summands } r)$   
 $\langle \text{proof} \rangle$

**theorem** *ACI-norm-idem[simp]*:

$\langle\langle r \rangle\rangle = \langle r \rangle$   
 $\langle \text{proof} \rangle$

**fun** *ACI-nPlus* :: 'a::linorder rexp  $\Rightarrow$  'a rexp  $\Rightarrow$  'a rexp

**where**

*ACI-nPlus* (*Plus* r1 r2) s = *ACI-nPlus* r1 (*ACI-nPlus* r2 s)  
| *ACI-nPlus* r (*Plus* s1 s2) =  
  (if r = s1 then *Plus* s1 s2  
  else if r < s1 then *Plus* r (*Plus* s1 s2)  
  else *Plus* s1 (*ACI-nPlus* r s2))  
| *ACI-nPlus* r s =  
  (if r = s then r  
  else if r < s then *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  $\implies \exists s t. r = \text{Plus } s t$   
⟨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*:  $\llbracket \text{ACI-norm-alt } r \rrbracket = \llbracket r \rrbracket$

⟨proof⟩

**lemma** *ACI-nPlus-singleton-PLUS*:

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

**lemma** *ACI-nPlus-PLUS*:

$\llbracket xs1 \neq []; xs2 \neq []; \forall x \in \text{set } (xs1 @ xs2). \neg(\exists r s. x = \text{Plus } r s); \text{sorted } xs2; \text{distinct } xs2 \rrbracket \implies$   
 $\text{ACI-nPlus } (\text{PLUS } xs1) (\text{PLUS } xs2) = \text{flatten PLUS } (\text{set } (xs1 @ xs2))$   
 ⟨proof⟩

**lemma** *ACI-nPlus-flatten-PLUS*:

$\llbracket X1 \neq \{\}; X2 \neq \{\}; \text{finite } X1; \text{finite } X2; \forall x \in X1 \cup X2. \neg(\exists r s. x = \text{Plus } r s) \rrbracket \implies$   
 $\text{ACI-nPlus } (\text{flatten PLUS } X1) (\text{flatten PLUS } X2) = \text{flatten PLUS } (X1 \cup X2)$   
 ⟨proof⟩

**lemma** *ACI-nPlus-ACI-norm[simp]*:  $\text{ACI-nPlus } \langle r \rangle \langle s \rangle = \langle \text{Plus } r s \rangle$   
 ⟨proof⟩

**lemma** *ACI-norm-alt*:

$\text{ACI-norm-alt } r = \langle r \rangle$   
 ⟨proof⟩

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

## 2.3 Finality

**primrec** *final* :: 'a rexp  $\Rightarrow$  bool

**where**

$\text{final Zero} = \text{False}$   
 $|\ \text{final Full} = \text{True}$   
 $|\ \text{final One} = \text{True}$   
 $|\ \text{final (Atom -)} = \text{False}$   
 $|\ \text{final (Plus } r\ s) = (\text{final } r \vee \text{final } s)$   
 $|\ \text{final (Times } r\ s) = (\text{final } r \wedge \text{final } s)$   
 $|\ \text{final (Star -)} = \text{True}$   
 $|\ \text{final (Not } r) = (\sim \text{final } r)$   
 $|\ \text{final (Inter } r1\ r2) = (\text{final } r1 \wedge \text{final } r2)$   
 $|\ \text{final (Pr } r) = \text{final } r$

**lemma** *toplevel-summands-final*:

$\text{final } s = (\exists r \in \text{toplevel-summands } s. \text{final } r)$   
 ⟨proof⟩

**lemma** *final-PLUS*:

$\text{final } (\text{PLUS } xs) = (\exists r \in \text{set } xs. \text{final } r)$   
 ⟨proof⟩

**theorem** *ACI-norm-final[simp]*:

$\text{final } \langle r \rangle = \text{final } r$   
 ⟨proof⟩

## 2.4 Wellformedness w.r.t. an alphabet

**locale** *alphabet* =

**fixes**  $\Sigma :: \text{nat} \Rightarrow 'a \text{ set } (\Sigma \ -)$

**and** *wf-atom* ::  $\text{nat} \Rightarrow 'b :: \text{linorder} \Rightarrow \text{bool}$

**begin**

**primrec** *wf* ::  $\text{nat} \Rightarrow 'b \text{ rexp} \Rightarrow \text{bool}$

**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*  $\wedge$  *wf n s*) |

*wf n (Times r s)* = (*wf n r*  $\wedge$  *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*  $\wedge$  *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*  $\in$   $\Sigma$  *n*)  $\wedge$  *wf-word n ws*)

**lemma** *wf-word-snoc[simp]*: *wf-word n (ws @ [w])* = ((*w*  $\in$   $\Sigma$  *n*)  $\wedge$  *wf-word n ws*)  
*<proof>*

**lemma** *wf-word-append[simp]*: *wf-word n (ws @ vs)* = (*wf-word n ws*  $\wedge$  *wf-word n vs*)  
*<proof>*

**lemma** *wf-word*: *wf-word n w* = (*w*  $\in$  *lists* ( $\Sigma$  *n*))  
*<proof>*

**lemma** *toplevel-summands-wf*:

*wf n s* = ( $\forall r \in \text{toplevel-summands } s. \text{wf } n \ r$ )

*<proof>*

**lemma** *wf-PLUS[simp]*:

*wf n (PLUS xs)* = ( $\forall r \in \text{set } xs. \text{wf } n \ r$ )

*<proof>*

**lemma** *wf-TIMES[simp]*:

*wf n (TIMES xs)* = ( $\forall r \in \text{set } xs. \text{wf } n \ r$ )

*<proof>*

**lemma** *wf-flatten-PLUS[simp]*:

*finite X*  $\implies$  *wf n (flatten PLUS X)* = ( $\forall r \in X. \text{wf } n \ r$ )

*<proof>*

**theorem** *ACI-norm-wf*[simp]:

$wf\ n\ \ll r \gg = wf\ n\ r$   
*<proof>*

**lemma** *wf-INTERSECT*[simp]:

$wf\ n\ (INTERSECT\ xs) = (\forall r \in set\ xs.\ wf\ n\ r)$   
*<proof>*

**lemma** *wf-flatten-INTERSECT*[simp]:

$finite\ X \implies wf\ n\ (flatten\ INTERSECT\ X) = (\forall r \in X.\ wf\ n\ r)$   
*<proof>*

**end**

## 2.5 Language

**locale** *project* =

*alphabet*  $\Sigma$  *wf-atom* **for**  $\Sigma :: nat \Rightarrow 'a\ set$  **and** *wf-atom*  $:: nat \Rightarrow 'b :: linorder$   
 $\Rightarrow bool +$

**fixes** *project*  $:: 'a \Rightarrow 'a$

**and** *lookup*  $:: 'b \Rightarrow 'a \Rightarrow bool$

**assumes** *project*:  $\bigwedge a.\ a \in \Sigma\ (Suc\ n) \implies project\ a \in \Sigma\ n$

**begin**

**primrec** *lang*  $:: nat \Rightarrow 'b\ rexp \Rightarrow 'a\ lang$  **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]:  $wf\ word\ (Suc\ n)\ ws \implies wf\ word\ n\ (map\ project\ ws)$

*<proof>*

**lemma** *wf-lang-wf-word*:  $wf\ n\ r \implies \forall w \in lang\ n\ r.\ wf\ word\ n\ w$

*<proof>*

**lemma** *lang-subset-lists*:  $wf\ n\ r \implies lang\ n\ r \subseteq lists\ (\Sigma\ n)$

*<proof>*

**lemma** *toplevel-summands-lang*:

$r \in toplevel\ summands\ s \implies lang\ n\ r \subseteq lang\ n\ s$

*<proof>*



**lemma** *toplevel-summands-lang-UN*:

$$\text{lang } n \ s = (\bigcup r \in \text{toplevel-summands } s. \text{lang } n \ r)$$

*<proof>*

**lemma** *toplevel-summands-in-lang*:

$$w \in \text{lang } n \ s = (\exists r \in \text{toplevel-summands } s. w \in \text{lang } n \ r)$$

*<proof>*

**lemma** *lang-PLUS[simp]*:

$$\text{lang } n \ (\text{PLUS } xs) = (\bigcup r \in \text{set } xs. \text{lang } n \ r)$$

*<proof>*

**lemma** *lang-TIMES[simp]*:

$$\text{lang } n \ (\text{TIMES } xs) = \text{foldr } (@@) \ (\text{map } (\text{lang } n) \ xs) \ {\ [] }$$

*<proof>*

**lemma** *lang-flatten-PLUS*:

$$\text{finite } X \implies \text{lang } n \ (\text{flatten } \text{PLUS } X) = (\bigcup r \in X. \text{lang } n \ r)$$

*<proof>*

**theorem** *ACI-norm-lang[simp]*:

$$\text{lang } n \ \langle\langle r \rangle\rangle = \text{lang } n \ r$$

*<proof>*

**lemma** *lang-final*: *final*  $r = (\ [] \in \text{lang } n \ r)$

*<proof>*

**lemma** *in-lang-INTERSECT*:

$$\text{wf-word } n \ w \implies w \in \text{lang } n \ (\text{INTERSECT } xs) = (\forall r \in \text{set } xs. w \in \text{lang } n \ r)$$

*<proof>*

**lemma** *lang-INTERSECT*:

$$\text{lang } n \ (\text{INTERSECT } xs) = (\text{if } xs = [] \text{ then lists } (\Sigma \ n) \text{ else } \bigcap r \in \text{set } xs. \text{lang } n \ r)$$

*<proof>*

**lemma** *lang-flatten-INTERSECT[simp]*:

**assumes** *finite*  $X \ X \neq \{\}$   $\forall r \in X. \text{wf } n \ r$

**shows**  $w \in \text{lang } n \ (\text{flatten } \text{INTERSECT } X) = (\forall r \in X. w \in \text{lang } n \ r)$  (**is**  $?L = ?R$ )

*<proof>*

**end**

### 3 Derivatives of $\Pi$ -Extended Regular Expressions

```

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

```

### 3.1 Syntactic Derivatives

```

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

```

```

primrec lderivs where
  lderivs [] r = r
| lderivs (w#ws) r = lderivs ws (lderv w r)

```

### 3.2 Finiteness of ACI-Equivalent Derivatives

**lemma** toplevel-summands-lderv:  
 toplevel-summands (lderv as r) = ( $\bigcup s \in \text{toplevel-summands } r. \text{toplevel-summands } (lderv as s)$ )  
*<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  $\in \{\text{Zero}, \text{One}\}$   
*<proof>*

**lemma** lderivs-Atom: lderivs xs (Atom as)  $\in \{\text{Zero}, \text{One}, \text{Atom } as\}$   
*<proof>*

**lemma** lderivs-Plus: lderivs xs (Plus r s) = Plus (ldervs xs r) (ldervs xs s)

$\langle \text{proof} \rangle$

**lemma** *lderivs-PLUS*:  $\text{lderivs } xs \ (\text{PLUS } ys) = \text{PLUS } (\text{map } (\text{lderivs } xs) \ ys)$   
 $\langle \text{proof} \rangle$

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

**lemma** *toplevel-summands-lderivs-Star-nonempty*:  
 $xs \neq [] \implies \text{toplevel-summands } (\text{lderivs } xs \ (\text{Star } r)) \subseteq$   
 $\{\text{Times } (\text{lderivs } ys \ r) \ (\text{Star } r) \mid ys. \exists zs. ys \neq [] \wedge zs \ @ \ ys = xs\}$   
 $\langle \text{proof} \rangle$

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

**lemma** *ex-lderivs-Pr*:  $\exists s. \text{lderivs } \text{ass } (\text{Pr } r) = \text{Pr } s$   
 $\langle \text{proof} \rangle$

**lemma** *toplevel-summands-PLUS*:  
 $xs \neq [] \implies \text{toplevel-summands } (\text{PLUS } (\text{map } f \ xs)) = (\bigcup r \in \text{set } xs. \text{toplevel-summands } (f \ r))$   
 $\langle \text{proof} \rangle$

**lemma** *lderiv-toplevel-summands-Zero*:  
 $\llbracket \text{lderivs } xs \ (\text{Pr } r) = \text{Pr } s; \text{toplevel-summands } r = \{\text{Zero}\} \rrbracket \implies \text{toplevel-summands } s = \{\text{Zero}\}$   
 $\langle \text{proof} \rangle$

**lemma** *toplevel-summands-lderivs-Pr*:  
 $\llbracket xs \neq []; \text{lderivs } xs \ (\text{Pr } r) = \text{Pr } s \rrbracket \implies$   
 $\text{toplevel-summands } s \subseteq \{\text{Zero}\} \vee \text{toplevel-summands } s \subseteq (\bigcup xs. \text{toplevel-summands } (\text{lderivs } xs \ r))$   
 $\langle \text{proof} \rangle$

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

**lemma** *ACI-norm-toplevel-summands-Zero*:  $\text{toplevel-summands } r \subseteq \{\text{Zero}\} \implies \llbracket r \rrbracket = \text{Zero}$

*<proof>*

**lemma** *ACI-norm-ldervis-Pr*:

$ACI\text{-norm} \text{ ' } \{lderiv\ xs\ (Pr\ r) \mid xs.\ True\} \subseteq$   
 $\{Pr\ Zero\} \cup \{Pr\ \ll s \gg \mid s.\ toplevel\text{-summands}\ s \subseteq (\bigcup xs.\ toplevel\text{-summands}$   
 $\ll lderiv\ xs\ r \gg)\}$   
*<proof>*

**lemma** *finite-ACI-norm-toplevel-summands*:  $finite\ B \implies finite\ \{f\ \ll s \gg \mid s.\ toplevel\text{-summands}\ s \subseteq B\}$   
*<proof>*

**lemma** *ldervis-Not*:  $ldervis\ xs\ (Not\ r) = Not\ (ldervis\ xs\ r)$   
*<proof>*

**lemma** *ldervis-Inter*:  $ldervis\ xs\ (Inter\ r\ s) = Inter\ (ldervis\ xs\ r)\ (ldervis\ xs\ s)$   
*<proof>*

**theorem** *finite-ldervis*:  $finite\ \{\ll lderiv\ xs\ r \gg \mid xs.\ True\}$   
*<proof>*

### 3.3 Wellformedness and language of derivatives

**lemma** *wf-ldervis[simp]*:  $wf\ n\ r \implies wf\ n\ (ldervis\ w\ r)$   
*<proof>*

**lemma** *wf-ldervis[simp]*:  $wf\ n\ r \implies wf\ n\ (ldervis\ ws\ r)$   
*<proof>*

**lemma** *lQuot-map-project*:

**assumes**  $as \in \Sigma\ n\ A \subseteq lists\ (\Sigma\ (Suc\ n))$

**shows**  $lQuot\ as\ (map\ project\ \text{' } A) = map\ project\ \text{' } (\bigcup a \in set\ (embed\ as).\ lQuot\ a\ A)$  (**is**  $?L = ?R$ )  
*<proof>*

**lemma** *lang-ldervis*:  $\ll wf\ n\ r; w \in \Sigma\ n \gg \implies lang\ n\ (ldervis\ w\ r) = lQuot\ w\ (lang\ n\ r)$   
*<proof>*

**lemma** *lang-ldervis*:  $\ll wf\ n\ r; wf\text{-word}\ n\ ws \gg \implies lang\ n\ (ldervis\ ws\ r) = lQuots\ ws\ (lang\ n\ r)$   
*<proof>*

**corollary** *ldervis-final*:

**assumes**  $wf\ n\ r\ wf\text{-word}\ n\ ws$

**shows**  $final\ (ldervis\ ws\ r) \longleftrightarrow ws \in lang\ n\ r$   
*<proof>*

**abbreviation** *ldervis-set*  $n\ r\ s \equiv \{(\ll lderiv\ w\ r \gg, \ll lderiv\ w\ s \gg) \mid w.\ wf\text{-word}\ n\}$

$w\}$

### 3.4 Deriving preserves ACI-equivalence

**lemma** *ACI-norm-PLUS*:

$list-all2 (\lambda r s. \langle r \rangle = \langle s \rangle) xs ys \implies \langle PLUS xs \rangle = \langle PLUS ys \rangle$   
 $\langle proof \rangle$

**lemma** *toplevel-summands-ACI-norm-ldderiv*:

$(\bigcup a \in toplevel-summands r. toplevel-summands \langle lderiv as \langle a \rangle \rangle) = toplevel-summands \langle lderiv as \langle r \rangle \rangle$   
 $\langle proof \rangle$

**theorem** *ACI-norm-ldderiv*:

$\langle lderiv as \langle r \rangle \rangle = \langle lderiv as r \rangle$   
 $\langle proof \rangle$

**corollary** *ldderiv-preserves*:  $\langle r \rangle = \langle s \rangle \implies \langle lderiv as r \rangle = \langle lderiv as s \rangle$

$\langle proof \rangle$

**lemma** *ldderivs-snoc[simp]*:  $ldderivs (ws @ [w]) r = (ldderiv w (ldderivs ws r))$

$\langle proof \rangle$

**theorem** *ACI-norm-ldderivs*:

$\langle lderivs ws \langle r \rangle \rangle = \langle lderivs ws r \rangle$   
 $\langle proof \rangle$

**lemma** *ldderivs-alt*:  $\langle lderivs w r \rangle = fold (\lambda a r. \langle lderiv a r \rangle) w \langle r \rangle$

$\langle proof \rangle$

**lemma** *finite-fold-ldderiv*:  $finite \{fold (\lambda a r. \langle lderiv a r \rangle) w \langle s \rangle \mid w. True\}$

$\langle proof \rangle$

**end**

## 4 Some Useful Regular Operators

**primrec** *REV* ::  $'a \text{ rexp} \Rightarrow 'a \text{ rexp}$  **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]*:  $REV (REV\ r) = r$   
⟨proof⟩

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

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

**lemma** (in *alphabet*) *wf-REV[simp]*:  $wf\ n\ r \implies wf\ n\ (REV\ r)$   
⟨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* - *Zero* = *Zero*  
| *rderiv* - *Full* = *Full*  
| *rderiv* - *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-ldderiv*:  $rderiv\ as\ r = REV\ (ldderiv\ as\ (REV\ r))$   
⟨proof⟩

**lemma** *rderivs-ldderivs*:  $rderivs\ w\ r = REV\ (ldderivs\ w\ (REV\ r))$

*<proof>*

**lemma** *wf-rderiv[simp]*:  $wf\ n\ r \implies wf\ n\ (rderiv\ w\ r)$   
*<proof>*

**lemma** *wf-rderivs[simp]*:  $wf\ n\ r \implies wf\ n\ (rderivs\ ws\ r)$   
*<proof>*

**lemma** *lang-rderiv*:  $\llbracket wf\ n\ r; as \in \Sigma\ n \rrbracket \implies lang\ n\ (rderiv\ as\ r) = rQuot\ as\ (lang\ n\ r)$   
*<proof>*

**lemma** *lang-rderivs*:  $\llbracket wf\ n\ r; wf\text{-word}\ n\ w \rrbracket \implies lang\ n\ (rderivs\ w\ r) = rQuots\ w\ (lang\ n\ r)$   
*<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]*:  $toplevel\text{-summands}\ (REV\ r) = REV\ \text{'}\ toplevel\text{-summands}\ r$   
*<proof>*

**lemma** *ACI-norm-REV*:  $\langle\langle REV\ \langle r \rangle \rangle\rangle = \langle\langle REV\ r \rangle\rangle$   
*<proof>*

**lemma** *ACI-norm-rderiv*:  $\langle\langle rderiv\ as\ \langle r \rangle \rangle\rangle = \langle\langle rderiv\ as\ r \rangle\rangle$   
*<proof>*

**lemma** *ACI-norm-rderivs*:  $\langle\langle rderivs\ w\ \langle r \rangle \rangle\rangle = \langle\langle rderivs\ w\ r \rangle\rangle$   
*<proof>*

**theorem** *finite-rderivs*:  $finite\ \{\langle\langle rderivs\ xs\ r \rangle\rangle \mid xs.\ True\}$   
*<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]*:  $\llbracket wf\ n\ r; wf\text{-word}\ n\ w; a \in \Sigma\ n \rrbracket \implies$   
 $lang\ n\ (rderiv\ a\ (lderivs\ w\ r)) = lang\ n\ (lderivs\ w\ (rderiv\ a\ r))$   
*<proof>*

**lemma** *lang-lderiv-rderivs[simp]*:  $\llbracket wf\ n\ r; wf\text{-word}\ n\ w; a \in \Sigma\ n \rrbracket \implies$   
 $lang\ n\ (lderiv\ a\ (rderivs\ w\ r)) = lang\ n\ (rderivs\ w\ (lderiv\ a\ r))$   
*<proof>*

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

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

**lemma** *wf-biderivs[simp]*:  $wf\ n\ r \implies wf\ n\ (biderivs\ w1\ w2\ r)$   
*<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*:  $\ll biderivs\ w1\ w2\ \ll r \gg = \ll biderivs\ w1\ w2\ r \gg$   
*<proof>*

**lemma** *finite*  $\{\ll biderivs\ w1\ w2\ r \gg \mid w1\ w2\ .\ True\}$   
*<proof>*

**end**

## 4.1 Quotienting by the same letter

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

**lemma** *fin-cut-same-Nil[simp]*:  $fin\text{-cut-same}\ x\ [] = []$   
*<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*:  $length\ xs = m \implies m \leq length\ (takeWhile\ P\ xs) \longleftrightarrow$   
 $Ball\ (set\ xs)\ P$



*<proof>*

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

**lemma** *fin-cut-same-singleton[simp]*: *fin-cut-same*  $x$  ( $xs @ [x]$ ) = *fin-cut-same*  $x$   $xs$   
*<proof>*

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

**lemma** *fin-cut-sameE*: *fin-cut-same*  $x$   $xs = ys \implies \exists m. xs = ys @ \text{replicate } m \text{ } x$   
*<proof>*

**definition** *SAMEQUOT*  $a$   $A = \{\text{fin-cut-same } a \text{ } x @ \text{replicate } m \text{ } x \mid x m. x \in A\}$

**lemma** *SAMEQUOT-mono*:  $A \subseteq B \implies \text{SAMEQUOT } a \text{ } A \subseteq \text{SAMEQUOT } a \text{ } B$   
*<proof>*

**locale** *embed2* = *embed*  $\Sigma$  *wf-atom* *project* *lookup* *embed*  
for  $\Sigma :: \text{nat} \Rightarrow 'a \text{ set}$   
and *wf-atom* ::  $\text{nat} \Rightarrow 'b :: \text{linorder} \Rightarrow \text{bool}$   
and *project* ::  $'a \Rightarrow 'a$   
and *lookup* ::  $'b \Rightarrow 'a \Rightarrow \text{bool}$   
and *embed* ::  $'a \Rightarrow 'a \text{ list} +$   
**fixes** *singleton* ::  $'a \Rightarrow 'b$   
**assumes** *wf-singleton[simp]*:  $a \in \Sigma \text{ } n \implies \text{wf-atom } n \text{ } (\text{singleton } a)$   
**assumes** *lookup-singleton[simp]*: *lookup* (*singleton*  $a$ )  $a' = (a = a')$   
**begin**

**lemma** *finite-rderivs-same*: *finite*  $\{\llbracket \text{rderivs } (\text{replicate } m \text{ } a) \text{ } r \rrbracket \mid m . \text{True}\}$   
*<proof>*

**lemma** *wf-word-replicate[simp]*:  $a \in \Sigma \text{ } n \implies \text{wf-word } n \text{ } (\text{replicate } m \text{ } a)$   
*<proof>*

**lemma** *star-singleton[simp]*: *star*  $\{[x]\} = \{\text{replicate } m \text{ } x \mid m . \text{True}\}$   
*<proof>*

**definition** *samequot*  $a$   $r = \text{Times } (\text{flatten } \text{PLUS } \{\llbracket \text{rderivs } (\text{replicate } m \text{ } a) \text{ } r \rrbracket \mid m . \text{True}\}) (\text{Star } (\text{Atom } (\text{singleton } a)))$

**lemma** *wf-samequot*:  $\llbracket \text{wf } n \text{ } r; a \in \Sigma \text{ } n \rrbracket \implies \text{wf } n \text{ } (\text{samequot } a \text{ } r)$   
*<proof>*

**lemma** *lang-samequot*:  $\llbracket \text{wf } n \text{ } r; a \in \Sigma \text{ } n \rrbracket \implies$

$\text{lang } n \text{ (samequot } a \text{ } r) = \text{SAMEQUOT } a \text{ (lang } n \text{ } r)$   
 ⟨proof⟩

**fun** *rderiv-and-add* **where**  
*rderiv-and-add* as ( $- :: \text{bool}, rs$ ) =  
 (let  
    $r = \llbracket \text{rderiv as (hd } rs) \rrbracket$   
 in if  $r \in \text{set } rs$  then ( $\text{False}, rs$ ) else ( $\text{True}, r \# rs$ )

**definition** *invar-rderiv-and-add* as  $r \text{ } brs \equiv$   
 ( $\text{if } \text{fst } brs \text{ then } \text{True} \text{ else } \llbracket \text{rderiv as (hd (snd } brs)) \rrbracket \in \text{set (snd } brs) \rrbracket \wedge$   
 $\text{snd } brs \neq [] \wedge \text{distinct (snd } brs) \wedge$   
 $(\forall i < \text{length (snd } brs). \text{snd } brs ! i = \llbracket \text{rderivs (replicate (length (snd } brs) - 1$   
 $- i) \text{ as) } r \rrbracket$ )

**lemma** *invar-rderiv-and-add-init*: *invar-rderiv-and-add* as  $r$  ( $\text{True}, [\llbracket r \rrbracket]$ )  
 ⟨proof⟩

**lemma** *invar-rderiv-and-add-step*: *invar-rderiv-and-add* as  $r \text{ } brs \implies \text{fst } brs \implies$   
*invar-rderiv-and-add* as  $r$  (*rderiv-and-add* as  $brs$ )  
 ⟨proof⟩

**lemma** *rderivs-replicate-mult*:  $\llbracket \llbracket \text{rderivs (replicate } i \text{ as) } r \rrbracket = \llbracket r \rrbracket; i > 0 \rrbracket \implies$   
 $\llbracket \text{rderivs (replicate (} m * i \text{) as) } r \rrbracket = \llbracket r \rrbracket$   
 ⟨proof⟩

**lemma** *rderivs-replicate-mult-rest*:  
**assumes**  $\llbracket \text{rderivs (replicate } i \text{ as) } r \rrbracket = \llbracket r \rrbracket \text{ } k < i$   
**shows**  $\llbracket \text{rderivs (replicate (} m * i + k \text{) as) } r \rrbracket = \llbracket \text{rderivs (replicate } k \text{ as) } r \rrbracket$  (**is**  
 $?L = ?R$ )  
 ⟨proof⟩

**lemma** *rderivs-replicate-mod*:  
**assumes**  $\llbracket \text{rderivs (replicate } i \text{ as) } r \rrbracket = \llbracket r \rrbracket \text{ } i > 0$   
**shows**  $\llbracket \text{rderivs (replicate } m \text{ as) } r \rrbracket = \llbracket \text{rderivs (replicate (} m \bmod i \text{) as) } r \rrbracket$  (**is**  
 $?L = ?R$ )  
 ⟨proof⟩

**lemma** *rderivs-replicate-diff*:  $\llbracket \llbracket \text{rderivs (replicate } i \text{ as) } r \rrbracket = \llbracket \text{rderivs (replicate } j$   
 $\text{ as) } r \rrbracket; i > j \rrbracket \implies$   
 $\llbracket \text{rderivs (replicate (} i - j \text{) as) (rderivs (replicate } j \text{ as) } r) \rrbracket = \llbracket \text{rderivs (replicate}$   
 $j \text{ as) } r \rrbracket$   
 ⟨proof⟩

**lemma** *samequot-wf*:  
**assumes**  $\text{wf } n \text{ } r \text{ while-option fst (rderiv-and-add as) (True, [\llbracket r \rrbracket])} = \text{Some (} b,$   
 $rs)$   
**shows**  $\text{wf } n \text{ (PLUS } rs)$   
 ⟨proof⟩

**lemma** *samequot-soundness*:

**assumes** *while-option fst (rderiv-and-add as) (True, [«r»]) = Some (b, rs)*  
**shows**  $\text{lang } n \text{ (PLUS } rs) = \bigcup (\text{lang } n \text{ ' } \{\llbracket rderivs \text{ (replicate } m \text{ as) } r \rrbracket \mid m. \text{True}\})$   
*<proof>*

**lemma** *length-subset-card*:  $\llbracket \text{finite } X; \text{distinct } (x \# xs); \text{set } (x \# xs) \subseteq X \rrbracket \implies \text{length } xs < \text{card } X$   
*<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*:  $\llbracket \text{wf } n \text{ } r; \text{as} \in \Sigma \text{ } n \rrbracket \implies \text{wf } n \text{ (samequot-exec as } r)$   
*<proof>*

**lemma** *samequot-exec-samequot*:  $\text{lang } n \text{ (samequot-exec as } r) = \text{lang } n \text{ (samequot as } r)$   
*<proof>*

**lemma** *lang-samequot-exec*:

$\llbracket \text{wf } n \text{ } r; \text{as} \in \Sigma \text{ } n \rrbracket \implies \text{lang } n \text{ (samequot-exec as } r) = \text{SAMEQUOT as (lang } n \text{ } r)$   
*<proof>*

**end**

## 4.2 Suffix and Prefix Languages

**definition** *Suffix* :: *'a lang*  $\Rightarrow$  *'a lang* **where**

*Suffix L = {w.  $\exists u. u @ w \in L$ }*

**definition** *Prefix* :: *'a lang*  $\Rightarrow$  *'a lang* **where**

*Prefix L = {w.  $\exists u. w @ u \in L$ }*

**lemma** *Prefix-Suffix*: *Prefix L = rev ' Suffix (rev ' L)*

*<proof>*

**definition** *Root* :: *'a lang*  $\Rightarrow$  *'a lang* **where**

*Root L = {x .  $\exists n > 0. x ^{\wedge n} \in L$ }*

**definition** *Cycle* :: 'a lang  $\Rightarrow$  'a lang **where**  
*Cycle*  $L = \{u @ w \mid u w. w @ u \in L\}$

**context** *embed*  
**begin**

**context**  
**fixes**  $n :: \text{nat}$   
**begin**

**definition** *SUFFIX* :: 'b rexp  $\Rightarrow$  'b rexp **where**  
*SUFFIX*  $r = \text{flatten PLUS } \{\ll\text{lderivs } w r\gg \mid w. \text{wf-word } n w\}$

**lemma** *finite-lderivs-wf*: *finite*  $\{\ll\text{lderivs } w r\gg \mid w. \text{wf-word } n w\}$   
 $\langle \text{proof} \rangle$

**definition** *PREFIX* :: 'b rexp  $\Rightarrow$  'b rexp **where**  
*PREFIX*  $r = \text{REV } (\text{SUFFIX } (\text{REV } r))$

**lemma** *wf-SUFFIX[simp]*:  $\text{wf } n r \Longrightarrow \text{wf } n (\text{SUFFIX } r)$   
 $\langle \text{proof} \rangle$

**lemma** *lang-SUFFIX[simp]*:  $\text{wf } n r \Longrightarrow \text{lang } n (\text{SUFFIX } r) = \text{Suffix } (\text{lang } n r)$   
 $\langle \text{proof} \rangle$

**lemma** *wf-PREFIX[simp]*:  $\text{wf } n r \Longrightarrow \text{wf } n (\text{PREFIX } r)$   
 $\langle \text{proof} \rangle$

**lemma** *lang-PREFIX[simp]*:  $\text{wf } n r \Longrightarrow \text{lang } n (\text{PREFIX } r) = \text{Prefix } (\text{lang } n r)$   
 $\langle \text{proof} \rangle$

**end**

**lemma** *take-drop-CycleI[intro!]*:  $x \in L \Longrightarrow \text{drop } i x @ \text{take } i x \in \text{Cycle } L$   
 $\langle \text{proof} \rangle$

**lemma** *take-drop-CycleI'[intro!]*:  $\text{drop } i x @ \text{take } i x \in L \Longrightarrow x \in \text{Cycle } L$   
 $\langle \text{proof} \rangle$

**end**

## 5 $\Pi$ -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  $\equiv$  *rexp-of-list* (CoPlus b) (CoZero b)

**abbreviation** *bool-unop-dual* b  $\equiv$  (if b then id else HOL.Not)

**abbreviation** *bool-binop-dual* b  $\equiv$  (if b then ( $\vee$ ) else ( $\wedge$ ))

**abbreviation** *set-binop-dual* b  $\equiv$  (if b then ( $\cup$ ) else ( $\cap$ ))

**primrec** *final-dual* :: 'a rexp-dual  $\Rightarrow$  bool

**where**

*final-dual* (CoZero b) = ( $\neg$  b)  
| *final-dual* (CoOne b) = b  
| *final-dual* (CoAtom b -) = ( $\neg$  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* ( $\neg$  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  $\Rightarrow$  'b rexp-dual  $\Rightarrow$  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  $\wedge$  *wf-dual* n s) |  
*wf-dual* n (CoTimes - r s) = (*wf-dual* n r  $\wedge$  *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) = ( $\forall$  r  $\in$  set xs. *wf-dual* n r)  
 <proof>

**abbreviation** *set-unop-dual* n b A  $\equiv$  if b then A else lists ( $\Sigma$  n) - A

**end**

**context** *project*

**begin**

**primrec** *lang-dual* :: nat  $\Rightarrow$  'b rexp-dual  $\Rightarrow$  'a lang **where**

*lang-dual* n (CoZero b) = *set-unop-dual* n b {} |  
*lang-dual* n (CoOne b) = *set-unop-dual* n b {[]} |

$lang\text{-}dual\ n\ (CoAtom\ b\ a) = set\text{-}unop\text{-}dual\ n\ b\ \{[x] \mid x.\ lookup\ a\ x \wedge x \in \Sigma\ n\} \mid$   
 $lang\text{-}dual\ n\ (CoPlus\ b\ r\ s) = set\text{-}binop\text{-}dual\ b\ (lang\text{-}dual\ n\ r)\ (lang\text{-}dual\ n\ s) \mid$   
 $lang\text{-}dual\ n\ (CoTimes\ b\ r\ s) = set\text{-}unop\text{-}dual\ n\ b$   
 $\quad (set\text{-}unop\text{-}dual\ n\ b\ (lang\text{-}dual\ n\ r)\ @@\ set\text{-}unop\text{-}dual\ n\ b\ (lang\text{-}dual\ n\ s)) \mid$   
 $lang\text{-}dual\ n\ (CoStar\ b\ r) = set\text{-}unop\text{-}dual\ n\ b\ (star\ (set\text{-}unop\text{-}dual\ n\ b\ (lang\text{-}dual\ n\ r))) \mid$   
 $lang\text{-}dual\ n\ (CoPr\ b\ r) = set\text{-}unop\text{-}dual\ n\ b\ (map\ project\ ' (set\text{-}unop\text{-}dual\ (n + 1)$   
 $b\ (lang\text{-}dual\ (n + 1)\ r)))$

**lemma** *wf-dual-lang-dual-wf-word*:  $wf\text{-}dual\ n\ r \implies \forall w \in lang\text{-}dual\ n\ r.\ wf\text{-}word\ n\ w$   
 $\langle proof \rangle$

**lemma** *lang-dual-subset-lists*:  $wf\text{-}dual\ n\ r \implies lang\text{-}dual\ n\ r \subseteq lists\ (\Sigma\ n)$   
 $\langle proof \rangle$

**lemma** *lang-dual-final-dual*:  $final\text{-}dual\ r = (\square \in lang\text{-}dual\ n\ r)$   
 $\langle proof \rangle$

**lemma** *lang-dual-PLUS-dual[simp]*:  
 $lang\text{-}dual\ n\ (CoPLUS\text{-}dual\ True\ xs) = (\bigcup r \in set\ xs.\ lang\text{-}dual\ n\ r)$   
 $\langle proof \rangle$

**lemma** *lang-dual-CoPLUS-dual[simp]*:  
 $lang\text{-}dual\ n\ (CoPLUS\text{-}dual\ False\ xs) = (if\ xs = \square\ then\ lists\ (\Sigma\ n)\ else\ \bigcap r \in set\ xs.\ lang\text{-}dual\ n\ r)$   
 $\langle proof \rangle$

**end**

**context** *embed*  
**begin**

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

**primrec** *lderivs-dual* **where**  
 $lderivs\text{-}dual\ \square\ r = r$   
 $\mid lderivs\text{-}dual\ (w\#ws)\ r = lderivs\text{-}dual\ ws\ (lderiv\text{-}dual\ w\ r)$

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

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

**lemma** *lang-dual-lderiv-dual*:  $\llbracket wf\text{-dual } n \ r; w \in \Sigma \ n \rrbracket \implies$   
 $lang\text{-dual } n \ (lderiv\text{-dual } w \ r) = lQuot \ w \ (lang\text{-dual } n \ r)$   
 ⟨proof⟩

**lemma** *lang-dual-lderivs-dual*:  $\llbracket wf\text{-dual } n \ r; wf\text{-word } n \ ws \rrbracket \implies$   
 $lang\text{-dual } n \ (lderivs\text{-dual } ws \ r) = lQuots \ ws \ (lang\text{-dual } n \ r)$   
 ⟨proof⟩

**corollary** *lderivs-dual-final-dual*:

**assumes**  $wf\text{-dual } n \ r \ wf\text{-word } n \ ws$

**shows**  $final\text{-dual } (lderivs\text{-dual } ws \ r) \longleftrightarrow ws \in lang\text{-dual } n \ r$

⟨proof⟩

**end**

**fun** *pnCoPlus* ::  $bool \Rightarrow 'a::linorder \ rexp\text{-dual} \Rightarrow 'a \ rexp\text{-dual} \Rightarrow 'a \ rexp\text{-dual}$

**where**

$pnCoPlus \ b1 \ (CoZero \ b2) \ r = (if \ b1 = b2 \ then \ r \ else \ CoZero \ b2)$   
 $| \ pnCoPlus \ b1 \ r \ (CoZero \ b2) = (if \ b1 = b2 \ then \ r \ else \ CoZero \ b2)$   
 $| \ pnCoPlus \ b1 \ (CoPlus \ b2 \ r \ s) \ t =$   
 $\quad (if \ b1 = b2 \ then \ pnCoPlus \ b2 \ r \ (pnCoPlus \ b2 \ s \ t) \ else \ CoPlus \ b1 \ (CoPlus \ b2 \ r \ s) \ t)$   
 $| \ pnCoPlus \ b1 \ r \ (CoPlus \ b2 \ s \ t) =$   
 $\quad (if \ b1 = b2 \ then$   
 $\quad\quad (if \ r = s \ then \ (CoPlus \ b2 \ s \ t)$   
 $\quad\quad\quad else \ if \ r \leq s \ then \ CoPlus \ b2 \ r \ (CoPlus \ b2 \ s \ t)$   
 $\quad\quad\quad\quad else \ CoPlus \ b2 \ s \ (pnCoPlus \ b2 \ r \ t))$   
 $\quad\quad else \ CoPlus \ b1 \ r \ (CoPlus \ b2 \ s \ t))$   
 $| \ pnCoPlus \ b \ r \ s =$   
 $\quad (if \ r = s \ then \ r$   
 $\quad\quad else \ if \ r \leq s \ then \ CoPlus \ b \ r \ s$   
 $\quad\quad\quad else \ CoPlus \ b \ s \ r)$

**lemma** (**in** *alphabet*) *wf-dual-pnCoPlus[simp]*:  $\llbracket wf\text{-dual } n \ r; wf\text{-dual } n \ s \rrbracket \implies wf\text{-dual } n \ (pnCoPlus \ b \ r \ s)$   
 ⟨proof⟩

**lemma** (**in** *project*) *lang-dual-pnCoPlus[simp]*:  $\llbracket wf\text{-dual } n \ r; wf\text{-dual } n \ s \rrbracket \implies$   
 $lang\text{-dual } n \ (pnCoPlus \ b \ r \ s) = lang\text{-dual } n \ (CoPlus \ b \ r \ s)$   
 ⟨proof⟩

**fun** *pnCoTimes* ::  $bool \Rightarrow 'a::linorder \ rexp\text{-dual} \Rightarrow 'a \ rexp\text{-dual} \Rightarrow 'a \ rexp\text{-dual}$

**where**

$pnCoTimes\ b1\ (CoZero\ b2)\ r = (if\ b1 = b2\ then\ CoZero\ b1\ else\ CoTimes\ b1\ (CoZero\ b2)\ r)$   
 $| pnCoTimes\ b1\ (CoOne\ b2)\ r = (if\ b1 = b2\ then\ r\ else\ CoTimes\ b1\ (CoOne\ b2)\ r)$   
 $| pnCoTimes\ b1\ (CoPlus\ b2\ r\ s)\ t = (if\ b1 = b2\ then\ pnCoPlus\ b2\ (pnCoTimes\ b2\ r\ t)\ (pnCoTimes\ b2\ s\ t)$   
 $\quad else\ CoTimes\ b1\ (CoPlus\ b2\ r\ s)\ t)$   
 $| pnCoTimes\ b\ r\ s = CoTimes\ b\ r\ s$

**lemma** (in *alphabet*) *wf-dual-pnCoTimes[simp]*:  $\llbracket wf\text{-dual}\ n\ r; wf\text{-dual}\ n\ s \rrbracket \implies wf\text{-dual}\ n\ (pnCoTimes\ b\ r\ s)$   
 $\langle proof \rangle$

**lemma** (in *project*) *lang-dual-pnCoTimes[simp]*:  $\llbracket wf\text{-dual}\ n\ r; wf\text{-dual}\ n\ s \rrbracket \implies lang\text{-dual}\ n\ (pnCoTimes\ b\ r\ s) = lang\text{-dual}\ n\ (CoTimes\ b\ r\ s)$   
 $\langle proof \rangle$

**fun** *pnCoPr* ::  $bool \Rightarrow 'a::linorder\ rexp\text{-dual} \Rightarrow 'a\ rexp\text{-dual}$  **where**  
 $pnCoPr\ b1\ (CoZero\ b2) = (if\ b1 = b2\ then\ CoZero\ b2\ else\ CoPr\ b1\ (CoZero\ b2))$   
 $| pnCoPr\ b1\ (CoOne\ b2) = (if\ b1 = b2\ then\ CoOne\ b2\ else\ CoPr\ b1\ (CoOne\ b2))$   
 $| pnCoPr\ b1\ (CoPlus\ b2\ r\ s) = (if\ b1 = b2\ then\ pnCoPlus\ b2\ (pnCoPr\ b2\ r)\ (pnCoPr\ b2\ s)$   
 $\quad else\ CoPr\ b1\ (CoPlus\ b2\ r\ s))$   
 $| pnCoPr\ b\ r = CoPr\ b\ r$

**lemma** (in *alphabet*) *wf-dual-pnCoPr[simp]*:  $wf\text{-dual}\ (Suc\ n)\ r \implies wf\text{-dual}\ n\ (pnCoPr\ b\ r)$   
 $\langle proof \rangle$

**lemma** (in *project*) *lang-dual-pnCoPr[simp]*:  $wf\text{-dual}\ (Suc\ n)\ r \implies lang\text{-dual}\ n\ (pnCoPr\ b\ r) = lang\text{-dual}\ n\ (CoPr\ b\ r)$   
 $\langle proof \rangle$

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

**lemma** (in *alphabet*) *wf-dual-pnorm-dual[simp]*:  $wf\text{-dual}\ n\ r \implies wf\text{-dual}\ n\ (pnorm\text{-dual}\ r)$   
 $\langle proof \rangle$

**lemma** (in *project*) *lang-dual-pnorm-dual[simp]*:  $wf\text{-dual}\ n\ r \implies lang\text{-dual}\ n\ (pnorm\text{-dual}\ r) = lang\text{-dual}\ n\ r$



*<proof>*

**primrec** *CoNot* **where**

*CoNot* (*CoZero* *b*) = *CoZero* ( $\neg$  *b*)  
| *CoNot* (*CoOne* *b*) = *CoOne* ( $\neg$  *b*)  
| *CoNot* (*CoAtom* *b* *a*) = *CoAtom* ( $\neg$  *b*) *a*  
| *CoNot* (*CoPlus* *b* *r* *s*) = *CoPlus* ( $\neg$  *b*) (*CoNot* *r*) (*CoNot* *s*)  
| *CoNot* (*CoTimes* *b* *r* *s*) = *CoTimes* ( $\neg$  *b*) (*CoNot* *r*) (*CoNot* *s*)  
| *CoNot* (*CoStar* *b* *r*) = *CoStar* ( $\neg$  *b*) (*CoNot* *r*)  
| *CoNot* (*CoPr* *b* *r*) = *CoPr* ( $\neg$  *b*) (*CoNot* *r*)

**primrec** *rexp-dual-of* **where**

*rexp-dual-of* *Zero* = *CoZero* *True*  
| *rexp-dual-of* *Full* = *CoZero* *False*  
| *rexp-dual-of* *One* = *CoOne* *True*  
| *rexp-dual-of* (*Atom* *a*) = *CoAtom* *True* *a*  
| *rexp-dual-of* (*Plus* *r* *s*) = *CoPlus* *True* (*rexp-dual-of* *r*) (*rexp-dual-of* *s*)  
| *rexp-dual-of* (*Times* *r* *s*) = *CoTimes* *True* (*rexp-dual-of* *r*) (*rexp-dual-of* *s*)  
| *rexp-dual-of* (*Star* *r*) = *CoStar* *True* (*rexp-dual-of* *r*)  
| *rexp-dual-of* (*Not* *r*) = *CoNot* (*rexp-dual-of* *r*)  
| *rexp-dual-of* (*Inter* *r* *s*) = *CoPlus* *False* (*rexp-dual-of* *r*) (*rexp-dual-of* *s*)  
| *rexp-dual-of* (*Pr* *r*) = *CoPr* *True* (*rexp-dual-of* *r*)

**lemma** (**in** *alphabet*) *wf-dual-CoNot[simp]*: *wf-dual* *n* *r*  $\implies$  *wf-dual* *n* (*CoNot* *r*)  
*<proof>*

**lemma** (**in** *project*) *lang-dual-CoNot[simp]*: *wf-dual* *n* *r*  $\implies$  *lang-dual* *n* (*CoNot* *r*)  
= *lists* ( $\Sigma$  *n*) – *lang-dual* *n* *r*  
*<proof>*

**lemma** (**in** *alphabet*) *wf-dual-rexp-dual-of[simp]*: *wf* *n* *r*  $\implies$  *wf-dual* *n* (*rexp-dual-of* *r*)  
*<proof>*

**lemma** (**in** *project*) *lang-dual-rexp-dual-of[simp]*: *wf* *n* *r*  $\implies$  *lang-dual* *n* (*rexp-dual-of* *r*)  
= *lang* *n* *r*  
*<proof>*

**end**

## 6 Deciding Equivalence of $\Pi$ -Extended Regular Expressions

**lemma** *image2p-in-rel*: *BNF-Greatest-Fixpoint.image2p* *f* *g* (*in-rel* *R*) = *in-rel* (*map-prod* *f* *g* ‘ *R*)  
*<proof>*

**lemma** *image2p-apply*: *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** *rtrancf-fold-product*:

**shows**  $\{((r, s), (f a r, f a s)) \mid r s a. a \in A\}^* = \{((r, s), (fold f w r, fold f w s)) \mid r s w. w \in lists A\}$  (**is**  $?L = ?R$ )

*<proof>*

**lemma** *in-fold-lQuot*:  $v \in fold lQuot w L \longleftrightarrow w @ v \in L$

*<proof>*

**lemma** (**in project**) *lang-eq-ext*:  $\llbracket wf n r; wf n s \rrbracket \Longrightarrow (lang n r = lang n s) = (\forall w \in lists(\Sigma n). w \in lang n r \longleftrightarrow w \in lang n s)$

*<proof>*

**lemma** (**in project**) *lang-eq-ext-Nil-fold-Deriv*:

**fixes**  $r s n$

**assumes** *WF*:  $wf n r wf n s$

**defines**  $\mathfrak{B} \equiv \{(fold lQuot w (lang n r), fold lQuot w (lang n s)) \mid w. w \in lists(\Sigma n)\}$

**shows**  $lang n r = lang n s \longleftrightarrow (\forall (K, L) \in \mathfrak{B}. [] \in K \longleftrightarrow [] \in L)$

*<proof>*

**locale** *rexp-DA = project set o  $\sigma$  wf-atom project lookup*

**for**  $\sigma :: nat \Rightarrow 'a list$

**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]*:  $\llbracket a \in set(\sigma n); wf-state s \rrbracket \Longrightarrow L (delta a s) = lQuot a (L s)$

**assumes** *final-iff-Nil[simp]*:  $final s \longleftrightarrow [] \in L s$

**assumes** *L-wf-state[dest]*:  $wf-state s \Longrightarrow L s \subseteq lists(set(\sigma n))$

**assumes** *init-wf-state[simp]*:  $wf n r \Longrightarrow wf-state (init r)$

**assumes** *delta-wf-state[simp]*:  $\llbracket a \in set(\sigma n); wf-state s \rrbracket \Longrightarrow wf-state (delta a s)$

**assumes** *L-post[simp]*:  $wf-state s \Longrightarrow L (post s) = L s$

**assumes** *wf-state-post[simp]*:  $wf-state s \Longrightarrow wf-state (post s)$

**begin**

**lemma** *L-deltas[simp]*:  $\llbracket wf-word n w; wf-state s \rrbracket \Longrightarrow L (fold delta w s) = fold$

*lQuot w (L s)*  
*<proof>*

**definition** *progression* (**infix**  $\rightarrow$  60) **where**

$R \rightarrow S = (\forall s1\ s2. R\ s1\ s2 \longrightarrow wf\text{-state}\ s1 \wedge wf\text{-state}\ s2 \wedge final\ s1 = final\ s2 \wedge$   
 $(\forall x \in set\ (\sigma\ n). BNF\text{-Greatest-Fixpoint.image2p}\ post\ post\ S\ (post\ (\delta x\ s1))\ (post\ (\delta x\ s2))))$

**lemma** *SUPR-progression*[*intro!*]:  $\forall n. \exists m. X\ n \rightarrow Y\ m \implies (SUP\ n. X\ n) \rightarrow$   
 $(SUP\ n. Y\ n)$   
*<proof>*

**definition** *bisimulation* **where**

*bisimulation*  $R = R \rightarrow R$

**definition** *bisimulation-upto* **where**

*bisimulation-upto*  $R\ f = R \rightarrow f\ R$

**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*  $f = (mono\ f \wedge (\forall R\ S. R \rightarrow S \longrightarrow f\ R \rightarrow f\ S))$

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

**lemma** *bisimulation-upto-bisimulation*:

**assumes** *compatible*  $f$  *bisimulation-upto*  $R\ f$

**obtains**  $S$  **where** *bisimulation*  $S\ R \leq S$

*<proof>*

**lemma** *bisimulation-eqL*: *bisimulation*  $(\lambda s1\ s2. wf\text{-state}\ s1 \wedge wf\text{-state}\ s2 \wedge L\ s1 = L\ s2)$

*<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* ( $ws, - :: 's \times 's$ ) *list*,  $- :: 's$  *rel*) = (case  $ws$  of []  $\Rightarrow$  *False* | ( $w :: 'a$  *list*,  $p, q$ )#-  $\Rightarrow$  *final p = final q*)  
**fun** *test* **where** *test* ( $ws, - :: 's$  *rel*) = (case  $ws$  of []  $\Rightarrow$  *False* | ( $p, q$ )#-  $\Rightarrow$  *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* ( $\lambda a.$   
     let  $r' = \text{delta } a \ r$ ;  $s' = \text{delta } a \ s$   
     in (( $a \# w, r', s'$ ), (*post r'*, *post s'*)) ( $\sigma \ n$ );  
     *new* = *remdups'* *snd* (*filter* ( $\lambda(-, rs). rs \notin N$ ) *succs*);  
      $ws' = \text{tl } ws \ @ \ \text{map } \text{fst } \text{new}$ ;  
      $N' = \text{set } (\text{map } \text{snd } \text{new}) \cup N$   
   in ( $ws', ps', N'$ )
 )

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

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

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

**definition** *invariant* **where**

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

**lemma** *invariant-start*:

$\llbracket \text{wf-state } r; \text{wf-state } s \rrbracket \Longrightarrow \text{invariant } r \ s \ ([([], r, s)], [], \{(\text{post } r, \text{post } s)\})$   
 <proof>

**lemma** *step-invariant-mono*:

**assumes** *step-invariant* ( $ws, ps, N$ ) = ( $ws', ps', N'$ )  
**shows**  $\text{snd } ' \ \text{set } ws \cup \text{set } ps \subseteq \text{snd } ' \ \text{set } ws' \cup \text{set } ps'$   
 <proof>

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

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

**lemma** *step-commute*:  $ws \neq [] \implies$   
 $(case \ step\text{-invariant } (ws, ps, N) \ of \ (ws', ps', N') \Rightarrow (map \ snd \ ws', N')) = step$   
 $(map \ snd \ ws, N)$   
 $\langle proof \rangle$

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

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

**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**  $lang \ n \ r = lang \ n \ s$   
 $\langle proof \rangle$

**definition** *check-equiv* **where**

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

**lemma** *check-equiv-sound*:  
**assumes** *check-equiv*  $r \ s$  **and** *WF*:  $wf \ n \ r \ wf \ n \ s$   
**shows**  $lang \ n \ r = lang \ n \ s$

*<proof>*

**definition** *counterexample where*

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

**lemma** *counterexample-sound:*

**assumes** *result:* *counterexample*  $r\ s = \text{Some } w$  **and** *WF:*  $wf\ n\ r\ wf\ n\ s$

**shows**  $w \in lang\ n\ r \longleftrightarrow w \notin lang\ n\ s$

*<proof>*

Auxiliary executable functions:

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

*reachable*  $s = \text{snd } (the\ (rtrancl\text{-while } (\lambda\cdot. \text{True})\ (\lambda s. \text{map } (\lambda a. \text{post } (delta\ a\ s))\ (\sigma\ n))\ (\text{init } s)))$

**definition** *automaton*  $:: 'b\ rexp \Rightarrow (('s * 'a) * 's)$  **set where**

*automaton*  $s =$   
*snd* (*the*  
(let  $i = \text{init } s$ ;  
*start*  $= (([i], \{post\ i\}), \{\})$ ;  
*test-invariant*  $= \lambda((ws, Z), A). ws \neq \[]$ ;  
*step-invariant*  $= \lambda((ws, Z), A).$   
(let  $s = \text{hd } ws$ ;  
*new-edges*  $= \text{map } (\lambda a. ((s, a), delta\ a\ s))\ (\sigma\ n)$ ;  
*new*  $= \text{remdups } (filter\ (\lambda ss. \text{post } ss \notin Z)\ (\text{map } \text{snd } \text{new-edges}))$   
*in*  $((\text{new } @\ \text{tl } ws, \text{post } 'set\ \text{new } \cup\ Z), \text{set } \text{new-edges } \cup\ A)$   
*in while-option test-invariant step-invariant start*))

**definition** *match*  $:: 'b\ rexp \Rightarrow 'a\ list \Rightarrow bool$  **where**

*match*  $s\ w = \text{final } (fold\ delta\ w\ (\text{init } s))$

**lemma** *match-correct:*  $\llbracket wf\text{-word } n\ w; wf\ n\ s \rrbracket \Longrightarrow \text{match } s\ w \longleftrightarrow w \in lang\ n\ s$

*<proof>*

**end**

**locale** *rexp-DFA*  $= rexp\text{-DA } \sigma\ wf\text{-atom } project\ lookup\ \text{init } delta\ \text{final } wf\text{-state } post$   
 $L\ n$

**for**  $\sigma :: nat \Rightarrow 'a\ list$   
**and**  $wf\text{-atom} :: nat \Rightarrow 'b :: linorder \Rightarrow bool$   
**and**  $project :: 'a \Rightarrow 'a$   
**and**  $lookup :: 'b \Rightarrow 'a \Rightarrow bool$   
**and**  $\text{init} :: 'b\ rexp \Rightarrow 's$   
**and**  $delta :: 'a \Rightarrow 's \Rightarrow 's$   
**and**  $\text{final} :: 's \Rightarrow bool$   
**and**  $wf\text{-state} :: 's \Rightarrow bool$

```

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-equiv-complete:
  assumes lang n r = lang n s wf n r wf n s
  shows check-equiv 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  $\sigma :: \text{nat} \Rightarrow 'a \text{ list}$ 
  and  $\text{wf-atom} :: \text{nat} \Rightarrow 'b :: \text{linorder} \Rightarrow \text{bool}$ 
  and  $\text{project} :: 'a \Rightarrow 'a$ 
  and  $\text{lookup} :: 'b \Rightarrow 'a \Rightarrow \text{bool}$ 
  and  $\text{init} :: 'b \text{ rexp} \Rightarrow 's$ 
  and  $\text{delta} :: 'a \Rightarrow 's \Rightarrow 's$ 
  and  $\text{final} :: 's \Rightarrow \text{bool}$ 
  and  $\text{wf-state} :: 's \Rightarrow \text{bool}$ 
  and  $L :: 's \Rightarrow 'a \text{ lang}$ 
  and  $n :: \text{nat}$ 
begin

lemma step-efficient[code]:  $\text{step} (ws, N) =$ 
  (let
    ( $r, s$ ) =  $\text{hd } ws$ ;
     $\text{new} = \text{remdups} (\text{filter} (\lambda(r,s). (r,s) \notin N) (\text{map} (\lambda a. (\text{delta } a r, \text{delta } a s)) (\sigma$ 
 $n)))$ 
  in ( $\text{tl } ws @ \text{new}, \text{set } \text{new} \cup N$ )
  <proof>

end

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

sublocale rexp-DA-no-post <proof>

end

locale rexp-DA-sim = project set o  $\sigma$  wf-atom project lookup
  for  $\sigma :: \text{nat} \Rightarrow 'a \text{ list}$ 
  and  $\text{wf-atom} :: \text{nat} \Rightarrow 'b :: \text{linorder} \Rightarrow \text{bool}$ 
  and  $\text{project} :: 'a \Rightarrow 'a$ 
  and  $\text{lookup} :: 'b \Rightarrow 'a \Rightarrow \text{bool} +$ 
  fixes  $\text{init} :: 'b \text{ rexp} \Rightarrow 's$ 
  fixes  $\text{sim-delta} :: 's \Rightarrow 's \text{ list}$ 
  fixes  $\text{final} :: 's \Rightarrow \text{bool}$ 
  fixes  $\text{wf-state} :: 's \Rightarrow \text{bool}$ 

```



```

fixes L :: 's ⇒ 'a lang
fixes post :: 's ⇒ 's
fixes n :: nat
assumes L-init[simp]: wf n r ⇒ L (init r) = lang n r
assumes final-iff-Nil[simp]: final s ⇔ [] ∈ L s
assumes L-wf-state[dest]: wf-state s ⇒ L s ⊆ lists (set (σ n))
assumes init-wf-state[simp]: wf n r ⇒ wf-state (init r)
assumes L-post[simp]: wf-state s ⇒ L (post s) = L s
assumes wf-state-post[simp]: wf-state s ⇒ wf-state (post s)
assumes L-sim-delta[simp]: wf-state s ⇒ map L (sim-delta s) = map (λa. lQuot a (L s)) (σ n)
assumes sim-delta-wf-state[simp]: wf-state s ⇒ ∀ s' ∈ set (sim-delta s). wf-state s'
begin

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

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

lemma L-delta[simp]: [a ∈ set (σ n); wf-state s] ⇒ L (delta a s) = lQuot a (L s)
  ⟨proof⟩

lemma delta-wf-state[simp]: [a ∈ set (σ n); wf-state s] ⇒ 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 ∪ toplevel-inters s
| toplevel-inters r = {r}

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

```

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

**context** *alphabet*  
**begin**

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

**end**

**context** *project*  
**begin**

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

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

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

**lemma** *lang-flatten-INTERSECT-finite*[simp]:  
 $\text{finite } X \implies w \in \text{lang } n \ (\text{flatten } \text{INTERSECT } X) =$   
 (*if*  $X = \{\}$  *then*  $w \in \text{lists } (\Sigma \ n)$  *else* ( $\forall r \in X. w \in \text{lang } n \ r$ ))  
 ⟨*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]:  $\text{set } (\text{merge-distinct } xs \ ys) = \text{set } xs \cup \text{set } ys$   
 ⟨*proof*⟩

**lemma** *sorted-merge-distinct*[simp]:  $\llbracket \text{sorted } xs; \text{sorted } ys \rrbracket \implies \text{sorted } (\text{merge-distinct } xs \ ys)$   
 ⟨*proof*⟩

**lemma** *distinct-merge-distinct*[simp]:  $\llbracket \text{sorted } xs; \text{ distinct } xs; \text{ sorted } ys; \text{ distinct } ys \rrbracket$   
 $\implies$   
*distinct* (*merge-distinct* *xs* *ys*)  
 ⟨*proof*⟩

**lemma** *sorted-list-of-set-merge-distinct*[simp]:  $\llbracket \text{sorted } xs; \text{ distinct } xs; \text{ sorted } ys; \text{ distinct } ys \rrbracket \implies$   
*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-with-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-with-option* *merge-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-summands* *r*  
 ⟨*proof*⟩

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

**fun** *Inters* **where**  
*Inters* (*Inter* *r* *s*) = *zip-with-option* *merge-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-inters* *r*  
 ⟨*proof*⟩

**lemma** *Inters-Some*: *Inters* *r* = *Some* *xs*  $\longleftrightarrow$   
 (*Zero*  $\notin$  *set* *xs*  $\wedge$  *xs* = *sorted-list-of-set* (*toplevel-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*:  $inPlus\ r\ s = (let\ X = toplevel-summands\ (Plus\ r\ s) - \{Zero\}$   
*in*  
 $\quad flatten\ PLUS\ (if\ Full \in X\ then\ \{Full\}\ else\ X))$   
*<proof>*

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

**fun** *inStar* **where**  
 $\quad inStar\ Zero = One$   
 $| inStar\ Full = Full$   
 $| inStar\ One = One$   
 $| inStar\ (Star\ r) = Star\ r$   
 $| inStar\ r = Star\ r$

**definition** *inInter* **where**  
 $\quad inInter\ r\ s = (case\ Inters\ (Inter\ r\ s)\ of\ None \Rightarrow Zero\ | Some\ rs \Rightarrow INTERSECT$   
 $\quad rs)$

**lemma** *inInter-alt*:  $inInter\ r\ s = (let\ X = toplevel-inters\ (Inter\ r\ s) - \{Full\}$   
*in*  
 $\quad flatten\ INTERSECT\ (if\ Zero \in X\ then\ \{Zero\}\ else\ X))$   
*<proof>*

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

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

**primrec** *inorm* **where**  
 $\quad inorm\ Zero = Zero$   
 $| inorm\ Full = Full$   
 $| inorm\ One = One$   
 $| inorm\ (Atom\ a) = Atom\ a$   
 $| inorm\ (Plus\ r\ s) = Plus\ (inorm\ r)\ (inorm\ s)$   
 $| inorm\ (Times\ r\ s) = Times\ (inorm\ r)\ (inorm\ s)$

|  $\text{inorm } (\text{Star } r) = \text{inStar } (\text{inorm } r)$   
|  $\text{inorm } (\text{Not } r) = \text{inNot } (\text{inorm } r)$   
|  $\text{inorm } (\text{Inter } r \ s) = \text{inInter } (\text{inorm } r) (\text{inorm } s)$   
|  $\text{inorm } (\text{Pr } r) = \text{inPr } (\text{inorm } r)$

**context** *alphabet* **begin**

**lemma** *wf-inPlus[simp]*:  $\llbracket \text{wf } n \ r; \ \text{wf } n \ s \rrbracket \implies \text{wf } n \ (\text{inPlus } r \ s)$   
 $\langle \text{proof} \rangle$

**lemma** *wf-inTimes[simp]*:  $\llbracket \text{wf } n \ r; \ \text{wf } n \ s \rrbracket \implies \text{wf } n \ (\text{inTimes } r \ s)$   
 $\langle \text{proof} \rangle$

**lemma** *wf-inStar[simp]*:  $\text{wf } n \ r \implies \text{wf } n \ (\text{inStar } r)$   
 $\langle \text{proof} \rangle$

**lemma** *wf-inInter[simp]*:  $\llbracket \text{wf } n \ r; \ \text{wf } n \ s \rrbracket \implies \text{wf } n \ (\text{inInter } r \ s)$   
 $\langle \text{proof} \rangle$

**lemma** *wf-inNot[simp]*:  $\text{wf } n \ r \implies \text{wf } n \ (\text{inNot } r)$   
 $\langle \text{proof} \rangle$

**lemma** *wf-inPr[simp]*:  $\text{wf } (\text{Suc } n) \ r \implies \text{wf } n \ (\text{inPr } r)$   
 $\langle \text{proof} \rangle$

**lemma** *wf-inorm[simp]*:  $\text{wf } n \ r \implies \text{wf } n \ (\text{inorm } r)$   
 $\langle \text{proof} \rangle$

**end**

**context** *project* **begin**

**lemma** *lang-inPlus[simp]*:  $\llbracket \text{wf } n \ r; \ \text{wf } n \ s \rrbracket \implies \text{lang } n \ (\text{inPlus } r \ s) = \text{lang } n \ (\text{Plus } r \ s)$   
 $\langle \text{proof} \rangle$

**lemma** *lang-inTimes[simp]*:  $\llbracket \text{wf } n \ r; \ \text{wf } n \ s \rrbracket \implies \text{lang } n \ (\text{inTimes } r \ s) = \text{lang } n \ (\text{Times } r \ s)$   
 $\langle \text{proof} \rangle$

**lemma** *lang-inStar[simp]*:  $\text{wf } n \ r \implies \text{lang } n \ (\text{inStar } r) = \text{lang } n \ (\text{Star } r)$   
 $\langle \text{proof} \rangle$

**lemma** *Zero-toplevel-inters[dest]*:  $\text{Zero} \in \text{toplevel-inters } r \implies \text{lang } n \ r = \{\}$   
 $\langle \text{proof} \rangle$

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

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

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

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

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

**lemma** *lang-inorm*[simp]:  $\text{wf } n \ r \implies \text{lang } n \ (\text{inorm } r) = \text{lang } n \ r$   
 ⟨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  $r$   $s$ )  $t = \text{pnPlus } r \ (\text{pnPlus } s \ t)$   
 | *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]:  $\llbracket \text{wf } n \ r; \text{wf } n \ s \rrbracket \implies \text{wf } n \ (\text{pnPlus } r \ s)$   
 ⟨proof⟩

**lemma** (in *project*) *lang-pnPlus*[simp]:  $\llbracket \text{wf } n \ r; \text{wf } n \ s \rrbracket \implies \text{lang } n \ (\text{pnPlus } r \ s) = \text{lang } n \ (\text{Plus } r \ s)$   
 ⟨proof⟩

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

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

**lemma** (in *project*) *lang-pnTimes*[simp]:  $\llbracket wf\ n\ r; wf\ n\ s \rrbracket \implies lang\ n\ (pnTimes\ r\ s) = lang\ n\ (Times\ r\ s)$   
 ⟨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*  $\leq$  *s* then Inter *r* (Inter *s* *t*)  
   else Inter *s* (*pnInter* *r* *t*))  
 | *pnInter* *r* *s* =  
   (if *r* = *s* then *s*  
   else if *r*  $\leq$  *s* then Inter *r* *s*  
   else Inter *s* *r*)

**lemma** (in *alphabet*) *wf-pnInter*[simp]:  $\llbracket wf\ n\ r; wf\ n\ s \rrbracket \implies wf\ n\ (pnInter\ r\ s)$   
 ⟨proof⟩

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

**fun** *pnNot* :: 'a::linorder rexp  $\Rightarrow$  '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]:  $wf\ n\ r \implies wf\ n\ (pnNot\ r)$   
 ⟨proof⟩

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

**fun** *pnPr* :: 'a::linorder rexp  $\Rightarrow$  '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*)  $wf\text{-}pnPr[simp]$ :  $wf (Suc\ n)\ r \implies wf\ n (pnPr\ r)$   
⟨*proof*⟩

**lemma** (in *project*)  $lang\text{-}pnPr[simp]$ :  $wf (Suc\ n)\ r \implies lang\ n (pnPr\ r) = lang\ n (Pr\ r)$   
⟨*proof*⟩

**primrec**  $pnorm :: 'a::linorder\ rexp \Rightarrow 'a\ rexp$  **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*)  $wf\text{-}pnorm[simp]$ :  $wf\ n\ r \implies wf\ n (pnorm\ r)$   
⟨*proof*⟩

**lemma** (in *project*)  $lang\text{-}pnorm[simp]$ :  $wf\ n\ r \implies lang\ n (pnorm\ r) = lang\ n\ r$   
⟨*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  $\Rightarrow$  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  $\Rightarrow$  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**  $\sigma = (\lambda \Sigma n. \text{concat} (\text{map} (\lambda bs. \text{map} (\lambda a. (a, bs)) \Sigma) (\text{List.n-lists } n$   
[*True*, *False*])))

**definition**  $\pi = (\lambda (a, bs). (a, \text{tl } bs))$

**definition**  $\varepsilon = (\lambda \Sigma (a::'a, bs). \text{if } a \in \text{set } \Sigma \text{ then } [(a, \text{True} \# bs), (a, \text{False} \# bs)]$   
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-atom*  $\Sigma$  n (*Singleton* a bs) = (a  $\in$  set  $\Sigma \wedge$  length bs = n)  
| *wf-atom*  $\Sigma$  n (*AQ* m a) = (a  $\in$  set  $\Sigma \wedge$  m < n)  
| *wf-atom*  $\Sigma$  n (*Arbitrary-Except* m -) = (m < n)  
| *wf-atom*  $\Sigma$  n (*Arbitrary-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-Except* m b) (-, bs) = (bs ! m = b)  
| *lookup* (*Arbitrary-Except2* m1 m2) (-, bs) = (bs ! m1  $\wedge$  bs ! m2)

**lemma**  $\pi \cdot \sigma: \pi$  ' (set o  $\sigma$   $\Sigma$ ) (n + 1) = (set o  $\sigma$   $\Sigma$ ) n  
{proof}

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

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

**primrec** *pre-wf-formula* ::  $\text{nat} \Rightarrow 'a \text{ formula} \Rightarrow \text{bool}$  **where**  
*pre-wf-formula*  $n$  (*FQ*  $a$   $m$ ) = ( $a \in \text{set } \Sigma \wedge m < n$ )  
| *pre-wf-formula*  $n$  (*FLess*  $m1$   $m2$ ) = ( $m1 < n \wedge m2 < n$ )  
| *pre-wf-formula*  $n$  (*FIn*  $m$   $M$ ) = ( $m < n \wedge M < n$ )  
| *pre-wf-formula*  $n$  (*FNot*  $\varphi$ ) = *pre-wf-formula*  $n$   $\varphi$   
| *pre-wf-formula*  $n$  (*FOr*  $\varphi_1$   $\varphi_2$ ) = (*pre-wf-formula*  $n$   $\varphi_1 \wedge$  *pre-wf-formula*  $n$   $\varphi_2$ )  
| *pre-wf-formula*  $n$  (*FAnd*  $\varphi_1$   $\varphi_2$ ) = (*pre-wf-formula*  $n$   $\varphi_1 \wedge$  *pre-wf-formula*  $n$   $\varphi_2$ )  
| *pre-wf-formula*  $n$  (*FExists*  $\varphi$ ) = (*pre-wf-formula* ( $n + 1$ )  $\varphi \wedge 0 \in \text{FOV } \varphi \wedge 0 \notin \text{SOV } \varphi$ )  
| *pre-wf-formula*  $n$  (*FEXISTS*  $\varphi$ ) = (*pre-wf-formula* ( $n + 1$ )  $\varphi \wedge 0 \notin \text{FOV } \varphi \wedge 0 \in \text{SOV } \varphi$ )

**abbreviation** *closed*  $\equiv$  *pre-wf-formula*  $0$

**definition** [*simp*]: *wf-formula*  $n$   $\varphi \equiv$  *pre-wf-formula*  $n$   $\varphi \wedge \text{FOV } \varphi \cap \text{SOV } \varphi = \{\}$

**lemma** *max-idx-vars*: *pre-wf-formula*  $n$   $\varphi \implies \forall p \in \text{FOV } \varphi \cup \text{SOV } \varphi. p < n$   
*<proof>*

**lemma** *finite-FOV*: *finite* ( $\text{FOV } \varphi$ )  
*<proof>*

### 9.3 ENC

**definition** *valid-ENC* ::  $\text{nat} \Rightarrow \text{nat} \Rightarrow ('a \text{ atom}) \text{ rexp}$  **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*:  $n = 0 \vee p < n \implies \text{wf } n$  (*valid-ENC*  $n$   $p$ )  
*<proof>*

**definition** *ENC* ::  $\text{nat} \Rightarrow \text{nat set} \Rightarrow ('a \text{ atom}) \text{ rexp}$  **where**  
*ENC*  $n$   $V =$  *flatten INTERSECT* (*valid-ENC*  $n$  '  $V$ )

**lemma** *wf-rexp-ENC*:  $\llbracket \text{finite } V; n = 0 \vee (\forall v \in V. v < n) \rrbracket \implies \text{wf } n$  (*ENC*  $n$   $V$ )

$\langle \text{proof} \rangle$

**lemma** *enc-atom- $\sigma$ -eq*:  $i < \text{length } w \implies$   
 $(\text{length } I = n \wedge p \in \text{set } \Sigma) \iff \text{enc-atom } I \ i \ p \in \text{set } (\sigma \ \Sigma \ n)$   
 $\langle \text{proof} \rangle$

**lemmas** *enc-atom- $\sigma$*  = *iffD1*[*OF enc-atom- $\sigma$ -eq*, *OF - conjI*]

**lemma** *enc-atom-bool-take-drop-True*:  
 $\llbracket r < \text{length } I; \text{ case } I \ ! \ r \text{ of } \text{Inl } p' \Rightarrow p = p' \mid \text{Inr } P \Rightarrow p \in P \rrbracket \implies$   
 $\text{enc-atom-bool } I \ p = \text{take } r \ (\text{enc-atom-bool } I \ p) \ @ \ \text{True} \ \# \ \text{drop} \ (\text{Suc } r)$   
 $(\text{enc-atom-bool } I \ p)$   
 $\langle \text{proof} \rangle$

**lemma** *enc-atom-bool-take-drop-True2*:  
 $\llbracket r < \text{length } I; \text{ case } I \ ! \ r \text{ of } \text{Inl } p' \Rightarrow p = p' \mid \text{Inr } P \Rightarrow p \in P;$   
 $s < \text{length } I; \text{ case } I \ ! \ s \text{ of } \text{Inl } p' \Rightarrow p = p' \mid \text{Inr } P \Rightarrow p \in P; r < s \rrbracket \implies$   
 $\text{enc-atom-bool } I \ p = \text{take } r \ (\text{enc-atom-bool } I \ p) \ @ \ \text{True} \ \#$   
 $\text{take } (s - \text{Suc } r) \ (\text{drop} \ (\text{Suc } r) \ (\text{enc-atom-bool } I \ p)) \ @ \ \text{True} \ \#$   
 $\text{drop} \ (\text{Suc } s) \ (\text{enc-atom-bool } I \ p)$   
 $\langle \text{proof} \rangle$

**lemma** *enc-atom-bool-take-drop-False*:  
 $\llbracket r < \text{length } I; \text{ case } I \ ! \ r \text{ of } \text{Inl } p' \Rightarrow p \neq p' \mid \text{Inr } P \Rightarrow p \notin P \rrbracket \implies$   
 $\text{enc-atom-bool } I \ p = \text{take } r \ (\text{enc-atom-bool } I \ p) \ @ \ \text{False} \ \# \ \text{drop} \ (\text{Suc } r)$   
 $(\text{enc-atom-bool } I \ p)$   
 $\langle \text{proof} \rangle$

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

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

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

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

**lemma** *AQ-D*:

**assumes**  $v \in \text{lang } n \text{ (Atom (AQ } m \ a))$   $m < n$   $a \in \text{set } \Sigma$

**shows**  $\exists x. v = [x] \wedge \text{fst } x = a \wedge \text{snd } x ! m$

*<proof>*

**lemma** *Arbitrary-ExceptD*:

**assumes**  $v \in \text{lang } n \text{ (Atom (Arbitrary-Except } r \ b))$   $r < n$

**shows**  $\exists x. v = [x] \wedge \text{snd } x ! r = b$

*<proof>*

**lemma** *Arbitrary-Except2D*:

**assumes**  $v \in \text{lang } n \text{ (Atom (Arbitrary-Except2 } r \ s))$   $r < n$   $s < n$

**shows**  $\exists x. v = [x] \wedge \text{snd } x ! r \wedge \text{snd } x ! s$

*<proof>*

**lemma** *star-Arbitrary-ExceptD*:

$\llbracket v \in \text{star (lang } n \text{ (Atom (Arbitrary-Except } r \ b)))$ ;  $r < n$ ;  $i < \text{length } v \rrbracket \implies$   
 $\text{snd } (v ! i) ! r = b$

*<proof>*

**end**

**end**

## 10 M2L

### 10.1 Encodings

**context** *formula*

**begin**

**fun** *enc* ::  $'a \text{ interp} \Rightarrow ('a \times \text{bool list}) \text{ list}$  **where**

$\text{enc } (w, I) = \text{map-index (enc-atom } I) w$

**abbreviation** *wf-interp*  $w \ I \equiv (\text{length } w > 0 \wedge$

$(\forall a \in \text{set } w. a \in \text{set } \Sigma) \wedge$

$(\forall x \in \text{set } I. \text{case } x \text{ of Inl } p \Rightarrow p < \text{length } w \mid \text{Inr } P \Rightarrow \forall p \in P. p < \text{length } w))$

**fun** *wf-interp-for-formula* ::  $'a \text{ interp} \Rightarrow 'a \text{ formula} \Rightarrow \text{bool}$  **where**

*wf-interp-for-formula*  $(w, I) \ \varphi =$

$(\text{wf-interp } w \ I \wedge$

$(\forall n \in \text{FOV } \varphi. \text{case } I ! n \text{ of Inl } - \Rightarrow \text{True} \mid - \Rightarrow \text{False}) \wedge$

$(\forall n \in \text{SOV } \varphi. \text{case } I ! n \text{ of Inl } - \Rightarrow \text{False} \mid - \Rightarrow \text{True}))$

**fun** *satisfies* ::  $'a \text{ interp} \Rightarrow 'a \text{ formula} \Rightarrow \text{bool}$  (**infix**  $\models$  50) **where**

$(w, I) \models \text{FQ } a \ m = (w ! (\text{case } I ! m \text{ of Inl } p \Rightarrow p) = a)$

$\mid (w, I) \models \text{FLess } m1 \ m2 = ((\text{case } I ! m1 \text{ of Inl } p \Rightarrow p) < (\text{case } I ! m2 \text{ of Inl } p \Rightarrow p))$

$\mid (w, I) \models \text{FIn } m \ M = ((\text{case } I ! m \text{ of Inl } p \Rightarrow p) \in (\text{case } I ! M \text{ of Inr } P \Rightarrow P))$

$| (w, I) \models FNot \varphi = (\neg (w, I) \models \varphi)$   
 $| (w, I) \models (FOr \varphi_1 \varphi_2) = ((w, I) \models \varphi_1 \vee (w, I) \models \varphi_2)$   
 $| (w, I) \models (FAnd \varphi_1 \varphi_2) = ((w, I) \models \varphi_1 \wedge (w, I) \models \varphi_2)$   
 $| (w, I) \models (FExists \varphi) = (\exists p. p \in \{0 .. length w - 1\} \wedge (w, Inl p \# I) \models \varphi)$   
 $| (w, I) \models (FEXISTS \varphi) = (\exists P. P \subseteq \{0 .. length w - 1\} \wedge (w, Inr P \# I) \models \varphi)$

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

**definition**  $dec\text{-word} \equiv map \text{fst}$

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

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

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

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

**lemma**  $positions\text{-in-row-length}$ :  $\exists ! p. p < length w \wedge \text{snd} (w ! p) ! i \implies$   
 $\text{the-elem} (\text{positions-in-row } w i) < length w$   
 $\langle proof \rangle$

**lemma**  $dec\text{-interp-Inl}$ :  $\llbracket i \in FO; i < n \rrbracket \implies \exists p. dec\text{-interp} n FO x ! i = Inl p$   
 $\langle proof \rangle$

**lemma**  $dec\text{-interp-not-Inr}$ :  $\llbracket dec\text{-interp} n FO x ! i = Inr P; i \in FO; i < n \rrbracket \implies$   
 $False$   
 $\langle proof \rangle$

**lemma**  $dec\text{-interp-Inr}$ :  $\llbracket i \notin FO; i < n \rrbracket \implies \exists P. dec\text{-interp} n FO x ! i = Inr P$   
 $\langle proof \rangle$

**lemma**  $dec\text{-interp-not-Inl}$ :  $\llbracket dec\text{-interp} n FO x ! i = Inl p; i \notin FO; i < n \rrbracket \implies$   
 $False$   
 $\langle proof \rangle$

**lemma**  $Inl\text{-dec-interp-length}$ :

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

**lemma** *Inr-dec-interp-length*:  $\llbracket \text{Inr } P \in \text{set } (\text{dec-interp } n \text{ FO } w); p \in P \rrbracket \implies p < \text{length } w$   
 ⟨proof⟩

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

**lemma** *enc-atom-dec*:  
 $\llbracket \text{wf-word } n \ w; \forall i \in FO. i < n \longrightarrow (\exists ! p. p < \text{length } w \wedge \text{snd } (w ! p) ! i); p < \text{length } w \rrbracket \implies$   
 $\text{enc-atom } (\text{dec-interp } n \text{ FO } w) \ p \ (\text{fst } (w ! p)) = w ! p$   
 ⟨proof⟩

**lemma** *enc-dec*:  
 $\llbracket \text{wf-word } n \ w; \forall i \in FO. i < n \longrightarrow (\exists ! p. p < \text{length } w \wedge \text{snd } (w ! p) ! i) \rrbracket \implies$   
 $\text{enc } (\text{dec-word } w, \text{dec-interp } n \text{ FO } w) = w$   
 ⟨proof⟩

**lemma** *dec-word-enc*:  $\text{dec-word } (\text{enc } (w, I)) = w$   
 ⟨proof⟩

**lemma** *enc-unique*:  
**assumes**  $\text{wf-interp } w \ I \ i < \text{length } I$   
**shows**  $\exists p. I ! i = \text{Inl } p \implies \exists ! p. p < \text{length } (\text{enc } (w, I)) \wedge \text{snd } (\text{enc } (w, I) ! p) ! i$   
 ⟨proof⟩

**lemma** *dec-interp-enc-Inl*:  
 $\llbracket \text{dec-interp } n \text{ FO } (\text{enc } (w, I)) ! i = \text{Inl } p'; I ! i = \text{Inl } p; i \in FO; i < n; \text{length } I = n; p < \text{length } w; \text{wf-interp } w \ I \rrbracket \implies$   
 $p = p'$   
 ⟨proof⟩

**lemma** *dec-interp-enc-Inr*:  
 $\llbracket \text{dec-interp } n \text{ FO } (\text{enc } (w, I)) ! i = \text{Inr } P'; I ! i = \text{Inr } P; i \notin FO; i < n; \text{length } I = n; \forall p \in P. p < \text{length } w \rrbracket \implies$   
 $P = P'$   
 ⟨proof⟩

**lemma** *length-dec-interp[simp]*:  $\text{length } (\text{dec-interp } n \text{ FO } x) = n$   
 ⟨proof⟩

**lemma** *nth-dec-interp[simp]*:  $i < n \implies \text{dec-interp } n \{x \mid i = \text{Inr } (\text{positions-in-row } x \ i)\}$

*<proof>*

**lemma** *set- $\sigma D$ [simp]*:  $(a, bs) \in \text{set } (\sigma \Sigma n) \implies a \in \text{set } \Sigma$

*<proof>*

**lemma** *lang-ENC*:

**assumes**  $FO \subseteq \{0 \dots n\}$   $SO \subseteq \{0 \dots n\} - FO$

**shows**  $\text{lang } n \ (\text{ENC } n \ FO) - \{\square\} = \{\text{enc } (w, I) \mid w \ I \ . \ \text{length } I = n \wedge \text{wf-interp } w \ I \wedge$

$(\forall i \in FO. \text{case } I \ ! \ i \ \text{of } \text{Inl } - \Rightarrow \text{True} \mid \text{Inr } - \Rightarrow \text{False}) \wedge$

$(\forall i \in SO. \text{case } I \ ! \ i \ \text{of } \text{Inl } - \Rightarrow \text{False} \mid \text{Inr } - \Rightarrow \text{True})\}$

**(is ?L = ?R)**

*<proof>*

**lemma** *lang-ENC-formula*:

**assumes** *wf-formula*  $n \ \varphi$

**shows**  $\text{lang } n \ (\text{ENC } n \ (\text{FOV } \varphi)) - \{\square\} = \{\text{enc } (w, I) \mid w \ I \ . \ \text{length } I = n \wedge \text{wf-interp-for-formula } (w, I) \ \varphi\}$

*<proof>*

## 10.2 Welldefinedness of enc wrt. Models

**lemma** *enc-alt-def*:

$\text{enc } (w, x \# I) = \text{map-index } (\lambda n \ (a, bs). (a, (\text{case } x \ \text{of } \text{Inl } p \Rightarrow n = p \mid \text{Inr } P \Rightarrow n \in P) \# bs)) (\text{enc } (w, I))$

*<proof>*

**lemma** *enc-extend-interp*:  $\text{enc } (w, I) = \text{enc } (w', I') \implies \text{enc } (w, x \# I) = \text{enc } (w', x \# I')$

*<proof>*

**lemma** *wf-interp-for-formula-FExists*:

$\llbracket \text{wf-formula } (\text{length } I) \ (\text{FExists } \varphi); w \neq \square \rrbracket \implies$

$\text{wf-interp-for-formula } (w, I) \ (\text{FExists } \varphi) \longleftrightarrow$

$(\forall p < \text{length } w. \text{wf-interp-for-formula } (w, \text{Inl } p \# I) \ \varphi)$

*<proof>*

**lemma** *wf-interp-for-formula-any-Inl*:  $\text{wf-interp-for-formula } (w, \text{Inl } p \# I) \ \varphi \implies \forall p < \text{length } w. \text{wf-interp-for-formula } (w, \text{Inl } p \# I) \ \varphi$

*<proof>*

**lemma** *wf-interp-for-formula-FEXISTS*:

$\llbracket \text{wf-formula } (\text{length } I) \ (\text{FEXISTS } \varphi); w \neq \square \rrbracket \implies$

$\text{wf-interp-for-formula } (w, I) \ (\text{FEXISTS } \varphi) \longleftrightarrow (\forall P \subseteq \{0 \dots \text{length } w - 1\}. \text{wf-interp-for-formula } (w, \text{Inr } P \# I) \ \varphi)$

*<proof>*

**lemma** *wf-interp-for-formula-any-Inr*:  $wf\text{-interp-for-formula } (w, \text{Inr } P \# I) \varphi \implies$   
 $\forall P \subseteq \{0 \dots \text{length } w - 1\}. wf\text{-interp-for-formula } (w, \text{Inr } P \# I) \varphi$   
 $\langle \text{proof} \rangle$

**lemma** *enc-word-length*:  $enc (w, I) = enc (w', I') \implies \text{length } w = \text{length } w'$   
 $\langle \text{proof} \rangle$

**lemma** *enc-length*:  
**assumes**  $w \neq []$   $enc (w, I) = enc (w', I')$   
**shows**  $\text{length } I = \text{length } I'$   
 $\langle \text{proof} \rangle$

**lemma** *wf-interp-for-formula-FOr*:  
 $wf\text{-interp-for-formula } (w, I) (FOr \varphi_1 \varphi_2) =$   
 $(wf\text{-interp-for-formula } (w, I) \varphi_1 \wedge wf\text{-interp-for-formula } (w, I) \varphi_2)$   
 $\langle \text{proof} \rangle$

**lemma** *wf-interp-for-formula-FAnd*:  
 $wf\text{-interp-for-formula } (w, I) (FAnd \varphi_1 \varphi_2) =$   
 $(wf\text{-interp-for-formula } (w, I) \varphi_1 \wedge wf\text{-interp-for-formula } (w, I) \varphi_2)$   
 $\langle \text{proof} \rangle$

**lemma** *enc-wf-interp*:  
**assumes**  $wf\text{-formula } (\text{length } I) \varphi$   $wf\text{-interp-for-formula } (w, I) \varphi$   
**shows**  $wf\text{-interp-for-formula } (\text{dec-word } (enc (w, I)), \text{dec-interp } (\text{length } I) (FOV$   
 $\varphi) (enc (w, I))) \varphi$   
**(is**  $wf\text{-interp-for-formula } (-, ?dec) \varphi$   
 $\langle \text{proof} \rangle$

**lemma** *enc-welldef*:  $\llbracket enc (w, I) = enc (w', I'); wf\text{-formula } (\text{length } I) \varphi;$   
 $wf\text{-interp-for-formula } (w, I) \varphi; wf\text{-interp-for-formula } (w', I') \varphi \rrbracket \implies$   
 $\text{satisfies } (w, I) \varphi \longleftrightarrow \text{satisfies } (w', I') \varphi$   
 $\langle \text{proof} \rangle$

**lemma** *lang<sub>M2L</sub>-FOr*:  
**assumes**  $wf\text{-formula } n (FOr \varphi_1 \varphi_2)$   
**shows**  $lang_{M2L} n (FOr \varphi_1 \varphi_2) \subseteq$   
 $(lang_{M2L} n \varphi_1 \cup lang_{M2L} n \varphi_2) \cap \{enc (w, I) \mid w \text{ I. length } I = n \wedge$   
 $wf\text{-interp-for-formula } (w, I) (FOr \varphi_1 \varphi_2)\}$   
**(is**  $\subseteq (?L1 \cup ?L2) \cap ?ENC$   
 $\langle \text{proof} \rangle$

**lemma** *lang<sub>M2L</sub>-FAnd*:  
**assumes**  $wf\text{-formula } n (FAnd \varphi_1 \varphi_2)$   
**shows**  $lang_{M2L} n (FAnd \varphi_1 \varphi_2) \subseteq$   
 $lang_{M2L} n \varphi_1 \cap lang_{M2L} n \varphi_2 \cap \{enc (w, I) \mid w \text{ I. length } I = n \wedge wf\text{-interp-for-formula}$   
 $(w, I) (FAnd \varphi_1 \varphi_2)\}$   
**(is**  $\subseteq ?L1 \cap ?L2 \cap ?ENC$   
 $\langle \text{proof} \rangle$



### 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  $\varphi$ ) = Inter (rexp.Not (rexp-of n  $\varphi$ )) (ENC n (FOV (FNot  $\varphi$ )))  
| *rexp-of* n (FOr  $\varphi_1$   $\varphi_2$ ) = Inter (Plus (rexp-of n  $\varphi_1$ ) (rexp-of n  $\varphi_2$ )) (ENC n  
(FOV (FOr  $\varphi_1$   $\varphi_2$ )))  
| *rexp-of* n (FAnd  $\varphi_1$   $\varphi_2$ ) = INTERSECT [rexp-of n  $\varphi_1$ , rexp-of n  $\varphi_2$ , ENC n  
(FOV (FAnd  $\varphi_1$   $\varphi_2$ ))] )  
| *rexp-of* n (FExists  $\varphi$ ) = Pr (rexp-of (n + 1)  $\varphi$ )  
| *rexp-of* n (FEXISTS  $\varphi$ ) = Pr (rexp-of (n + 1)  $\varphi$ )

**fun** *rexp-of-alt* :: nat  $\Rightarrow$  'a formula  $\Rightarrow$  ('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  $\varphi$ ) = rexp.Not (rexp-of-alt n  $\varphi$ )  
| *rexp-of-alt* n (FOr  $\varphi_1$   $\varphi_2$ ) = Plus (rexp-of-alt n  $\varphi_1$ ) (rexp-of-alt n  $\varphi_2$ )  
| *rexp-of-alt* n (FAnd  $\varphi_1$   $\varphi_2$ ) = Inter (rexp-of-alt n  $\varphi_1$ ) (rexp-of-alt n  $\varphi_2$ )  
| *rexp-of-alt* n (FExists  $\varphi$ ) = Pr (Inter (rexp-of-alt (n + 1)  $\varphi$ ) (ENC (n + 1)  
(FOV  $\varphi$ )))  
| *rexp-of-alt* n (FEXISTS  $\varphi$ ) = Pr (Inter (rexp-of-alt (n + 1)  $\varphi$ ) (ENC (n + 1)  
(FOV  $\varphi$ )))

**definition** *rexp-of' n  $\varphi$*  = Inter (rexp-of-alt n  $\varphi$ ) (ENC n (FOV  $\varphi$ ))

**fun** *rexp-of-alt'* :: nat  $\Rightarrow$  'a formula  $\Rightarrow$  ('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  $\varphi$ ) = rexp.Not (rexp-of-alt' n  $\varphi$ )  
| *rexp-of-alt'* n (FOr  $\varphi_1$   $\varphi_2$ ) = Plus (rexp-of-alt' n  $\varphi_1$ ) (rexp-of-alt' n  $\varphi_2$ )  
| *rexp-of-alt'* n (FAnd  $\varphi_1$   $\varphi_2$ ) = Inter (rexp-of-alt' n  $\varphi_1$ ) (rexp-of-alt' n  $\varphi_2$ )  
| *rexp-of-alt'* n (FExists  $\varphi$ ) = Pr (Inter (rexp-of-alt' (n + 1)  $\varphi$ ) (ENC (n + 1)  
{0}))  
| *rexp-of-alt'* n (FEXISTS  $\varphi$ ) = Pr (rexp-of-alt' (n + 1)  $\varphi$ )

**definition** *rexp-of'' n  $\varphi$*  = Inter (rexp-of-alt' n  $\varphi$ ) (ENC n (FOV  $\varphi$ ))

**theorem** *lang<sub>M2L</sub>-rexp-of*: wf-formula n  $\varphi \implies \text{lang}_{M2L} n \varphi = \text{lang} n (\text{rexp-of } n$

$\varphi) - \{\square\}$   
**(is**  $- \implies - = ?L n \varphi)$   
 $\langle proof \rangle$

**lemma** *wf-rexp-of*:  $wf\text{-formula } n \varphi \implies wf\ n (rexp\text{-of } n \varphi)$   
 $\langle proof \rangle$

**lemma** *wf-rexp-of-alt*:  $wf\text{-formula } n \varphi \implies wf\ n (rexp\text{-of-alt } n \varphi)$   
 $\langle proof \rangle$

**lemma** *wf-rexp-of'*:  $wf\text{-formula } n \varphi \implies wf\ n (rexp\text{-of}' n \varphi)$   
 $\langle proof \rangle$

**lemma** *wf-rexp-of-alt'*:  $wf\text{-formula } n \varphi \implies wf\ n (rexp\text{-of-alt}' n \varphi)$   
 $\langle proof \rangle$

**lemma** *wf-rexp-of''*:  $wf\text{-formula } n \varphi \implies wf\ n (rexp\text{-of}'' n \varphi)$   
 $\langle proof \rangle$

**lemma** *ENC-Not*:  $ENC\ n (FOV (FNot \varphi)) = ENC\ n (FOV \varphi)$   
 $\langle proof \rangle$

**lemma** *ENC-And*:  
 $wf\text{-formula } n (FAnd \varphi \psi) \implies lang\ n (ENC\ n (FOV (FAnd \varphi \psi))) - \{\square\} \subseteq lang\ n (ENC\ n (FOV \varphi)) \cap lang\ n (ENC\ n (FOV \psi)) - \{\square\}$   
 $\langle proof \rangle$

**lemma** *ENC-Or*:  
 $wf\text{-formula } n (FOr \varphi \psi) \implies lang\ n (ENC\ n (FOV (FOr \varphi \psi))) - \{\square\} \subseteq lang\ n (ENC\ n (FOV \varphi)) \cap lang\ n (ENC\ n (FOV \psi)) - \{\square\}$   
 $\langle proof \rangle$

**lemma** *project-enc*:  $map\ \pi (enc\ (w, x \# I)) = enc\ (w, I)$   
 $\langle proof \rangle$

**lemma** *list-list-eqI*:  
**assumes**  $\forall (-, x) \in set\ xs. x \neq \square \ \forall (-, y) \in set\ ys. y \neq \square$   
 $map\ (\lambda(-, x). hd\ x)\ xs = map\ (\lambda(-, x). hd\ x)\ ys\ map\ \pi\ xs = map\ \pi\ ys$   
**shows**  $xs = ys$   
 $\langle proof \rangle$

**lemma** *project-enc-extend*:  
**assumes**  $map\ \pi\ x = enc\ (w, I) \ \forall (-, x) \in set\ x. x \neq \square$   
**shows**  $x = enc\ (w, Inr\ (positions\text{-in-row } x\ 0) \# I)$   
 $\langle proof \rangle$

**lemma** *ENC-Exists*:  
 $wf\text{-formula } n (FExists \varphi) \implies lang\ n (ENC\ n (FOV (FExists \varphi))) - \{\square\} = map\ \pi \text{ ' } lang\ (Suc\ n) (ENC\ (Suc\ n) (FOV \varphi)) - \{\square\}$

*<proof>*

**lemma** *ENC-EXISTS:*

*wf-formula*  $n$  (*FEXISTS*  $\varphi$ )  $\implies$   $\text{lang } n$  ( $\text{ENC } n$  ( $\text{FOV}$  (*FEXISTS*  $\varphi$ ))) -  $\{\emptyset\}$   
 $=$   $\text{map } \pi$  '  $\text{lang}$  ( $\text{Suc } n$ ) ( $\text{ENC}$  ( $\text{Suc } n$ ) ( $\text{FOV}$   $\varphi$ )) -  $\{\emptyset\}$   
*<proof>*

**lemma** *map-project-empty:*  $\text{map } \pi$  '  $A - \{\emptyset\} = \text{map } \pi$  ' ( $A - \{\emptyset\}$ )

*<proof>*

**lemma** *lang<sub>M2L</sub>-rexp-of-rexp-of':*

*wf-formula*  $n$   $\varphi \implies \text{lang } n$  ( $\text{rexp-of } n$   $\varphi$ ) -  $\{\emptyset\} = \text{lang } n$  ( $\text{rexp-of}'$   $n$   $\varphi$ ) -  $\{\emptyset\}$   
*<proof>*

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

*<proof>*

**lemma** *lang-ENC-split:*

**assumes** *finite*  $X$   $X = Y1 \cup Y2$   $n = 0 \vee (\forall p \in X. p < n)$

**shows**  $\text{lang } n$  ( $\text{ENC } n$   $X$ ) =  $\text{lang } n$  ( $\text{ENC } n$   $Y1$ )  $\cap$   $\text{lang } n$  ( $\text{ENC } n$   $Y2$ )

*<proof>*

**lemma** *map-project-Int-ENC:*

**assumes**  $0 \notin X$   $X \subseteq \{0 ..< n + 1\}$   $Z \subseteq \text{lists } ((\text{set } o \sigma \Sigma) (n + 1))$

**shows**  $\text{map } \pi$  ' ( $Z \cap \text{lang } (n + 1)$  ( $\text{ENC } (n + 1)$   $X$ )) -  $\{\emptyset\} =$

$\text{map } \pi$  '  $Z \cap \text{lang } n$  ( $\text{ENC } n$  ( $(\lambda x. x - 1)$  '  $X$ )) -  $\{\emptyset\}$

*<proof>*

**lemma** *map-project-ENC:*

**assumes**  $X \subseteq \{0 ..< n + 1\}$   $Z \subseteq \text{lists } ((\text{set } o \sigma \Sigma) (n + 1))$

**shows**  $\text{map } \pi$  ' ( $Z \cap \text{lang } (n + 1)$  ( $\text{ENC } (n + 1)$   $X$ )) -  $\{\emptyset\} =$

(*if*  $0 \in X$

*then*  $\text{map } \pi$  ' ( $Z \cap \text{lang } (n + 1)$  ( $\text{ENC } (n + 1)$   $\{0\}$ ))  $\cap$   $\text{lang } n$  ( $\text{ENC } n$  ( $(\lambda x.$   
 $x - 1)$  ' ( $X - \{0\}$ ))) -  $\{\emptyset\}$

*else*  $\text{map } \pi$  '  $Z \cap \text{lang } n$  ( $\text{ENC } n$  ( $(\lambda x. x - 1)$  ' ( $X - \{0\}$ ))) -  $\{\emptyset\}$ )

(**is**  $?L = (\text{if - then ?R1 else ?R2})$ )

*<proof>*

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

**lemma** *lang<sub>M2L</sub>-rexp-of'-rexp-of'':*

*wf-formula*  $n$   $\varphi \implies \text{lang } n$  ( $\text{rexp-of}'$   $n$   $\varphi$ ) -  $\{\emptyset\} = \text{lang } n$  ( $\text{rexp-of}''$   $n$   $\varphi$ ) -  $\{\emptyset\}$   
*<proof>*

**theorem** *lang<sub>M2L</sub>-rexp-of':* *wf-formula*  $n$   $\varphi \implies \text{lang}_{M2L} n \varphi = \text{lang } n$  ( $\text{rexp-of}'$   
 $n$   $\varphi$ ) -  $\{\emptyset\}$

*<proof>*

**theorem** *lang<sub>M2L</sub>-rexp-of''*: wf-formula  $n \varphi \implies \text{lang}_{M2L} n \varphi = \text{lang } n (\text{rexp-of'' } n \varphi) - \{\square\}$   
 <proof>

**end**

## 11 Normalization of M2L Formulas

**fun** *nNot* **where**

*nNot* (*FNot*  $\varphi$ ) =  $\varphi$   
 | *nNot* (*FAnd*  $\varphi1 \varphi2$ ) = *FOr* (*nNot*  $\varphi1$ ) (*nNot*  $\varphi2$ )  
 | *nNot* (*FOr*  $\varphi1 \varphi2$ ) = *FAnd* (*nNot*  $\varphi1$ ) (*nNot*  $\varphi2$ )  
 | *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*  $n$   $\varphi$

$n \varphi$   
 ⟨proof⟩

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

**lemma** *lang<sub>M2L</sub>-norm[simp]*: *lang<sub>M2L</sub>*  $n$  (*norm*  $\varphi$ ) = *lang<sub>M2L</sub>*  $n$   $\varphi$   
 ⟨proof⟩

**end**

## 12 Deciding Equivalence of M2L Formulas

**global-interpretation** *embed set o*  $\sigma \Sigma$  *wf-atom*  $\Sigma \pi$  *lookup*  $\varepsilon \Sigma$   
**for**  $\Sigma :: 'a :: \text{linorder}$  *list*  
**defines**  
 $\mathfrak{D} = \text{embed.lderiv lookup } (\varepsilon \Sigma)$   
**and**  $\text{Co}\mathfrak{D} = \text{embed.lderiv-dual lookup } (\varepsilon \Sigma)$   
 ⟨proof⟩

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

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

**lemma** *lang-Plus-Zero*: *lang*  $\Sigma n$  (*Plus r One*) = *lang*  $\Sigma n$  (*Plus s One*)  $\longleftrightarrow$  *lang*  
 $\Sigma n r - \{\}\} = \text{lang } \Sigma n s - \{\}\}$   
 ⟨proof⟩

**lemmas**  $\text{lang}_{M2L}\text{-rexp-of-norm} = \text{trans}[\text{OF sym}[\text{OF } \Phi.\text{lang}_{M2L}\text{-norm}] \Phi.\text{lang}_{M2L}\text{-rexp-of}]$   
**lemmas**  $\text{lang}_{M2L}\text{-rexp-of}'\text{-norm} = \text{trans}[\text{OF sym}[\text{OF } \Phi.\text{lang}_{M2L}\text{-norm}] \Phi.\text{lang}_{M2L}\text{-rexp-of}']$   
**lemmas**  $\text{lang}_{M2L}\text{-rexp-of}''\text{-norm} = \text{trans}[\text{OF sym}[\text{OF } \Phi.\text{lang}_{M2L}\text{-norm}] \Phi.\text{lang}_{M2L}\text{-rexp-of}'']$

⟨ML⟩

**global-interpretation**  $D$ :  $\text{rexp-DFA } \sigma \Sigma \text{ wf-atom } \Sigma \pi \text{ lookup } \lambda x. \llbracket \text{pnorm } (inorm x) \rrbracket$   
 $\lambda a r. \llbracket \mathfrak{D} \Sigma a r \rrbracket \text{ final alphabet.wf } (wf\text{-atom } \Sigma) n \text{ pnorm lang } \Sigma n n$   
**for**  $\Sigma :: 'a :: \text{linorder list}$  **and**  $n :: \text{nat}$   
**defines**  
 $\text{test} = \text{rexp-DA.test } (final :: 'a \text{ atom rexp} \Rightarrow \text{bool})$   
**and**  $\text{step} = \text{rexp-DA.step } (\sigma \Sigma) (\lambda a r. \llbracket \mathfrak{D} \Sigma a r \rrbracket) \text{ pnorm } n$   
**and**  $\text{closure} = \text{rexp-DA.closure } (\sigma \Sigma) (\lambda a r. \llbracket \mathfrak{D} \Sigma a r \rrbracket) \text{ final pnorm } n$   
**and**  $\text{check-equivRE} = \text{rexp-DA.check-equiv } (\sigma \Sigma) (\lambda x. \llbracket \text{pnorm } (inorm x) \rrbracket) (\lambda a r. \llbracket \mathfrak{D} \Sigma a r \rrbracket) \text{ final pnorm } n$   
**and**  $\text{test-invariant} = \text{rexp-DA.test-invariant } (final :: 'a \text{ atom rexp} \Rightarrow \text{bool}) ::$   
 $(('a \times \text{bool list}) \text{ list} \times -) \text{ list} \times - \Rightarrow \text{bool}$   
**and**  $\text{step-invariant} = \text{rexp-DA.step-invariant } (\sigma \Sigma) (\lambda a r. \llbracket \mathfrak{D} \Sigma a r \rrbracket) \text{ pnorm } n$   
**and**  $\text{closure-invariant} = \text{rexp-DA.closure-invariant } (\sigma \Sigma) (\lambda a r. \llbracket \mathfrak{D} \Sigma a r \rrbracket)$   
 $\text{final pnorm } n$   
**and**  $\text{counterexampleRE} = \text{rexp-DA.counterexample } (\sigma \Sigma) (\lambda x. \llbracket \text{pnorm } (inorm x) \rrbracket) (\lambda a r. \llbracket \mathfrak{D} \Sigma a r \rrbracket) \text{ final pnorm } n$   
**and**  $\text{reachable} = \text{rexp-DA.reachable } (\sigma \Sigma) (\lambda x. \llbracket \text{pnorm } (inorm x) \rrbracket) (\lambda a r. \llbracket \mathfrak{D} \Sigma a r \rrbracket) \text{ pnorm } n$   
**and**  $\text{automaton} = \text{rexp-DA.automaton } (\sigma \Sigma) (\lambda x. \llbracket \text{pnorm } (inorm x) \rrbracket) (\lambda a r. \llbracket \mathfrak{D} \Sigma a r \rrbracket) \text{ pnorm } n$   
 $\langle \text{proof} \rangle$

**definition**  $\text{check-equiv where}$

$\text{check-equiv } n \varphi \psi \longleftrightarrow \text{wf-formula } n (FOr \varphi \psi) \wedge$   
 $\text{slow.check-equivRE Enum.enum } n (\text{Plus } (\text{rexp-of'' } n (\text{norm } \varphi)) \text{ One}) (\text{Plus } (\text{rexp-of'' } n (\text{norm } \psi)) \text{ One})$

**definition**  $\text{counterexample where}$

$\text{counterexample } n \varphi \psi =$   
 $\text{map-option } (\lambda w. \text{dec-interp } n (FOV (FOr \varphi \psi)) w)$   
 $(\text{slow.counterexampleRE Enum.enum } n (\text{Plus } (\text{rexp-of'' } n (\text{norm } \varphi)) \text{ One}) (\text{Plus } (\text{rexp-of'' } n (\text{norm } \psi)) \text{ One}))$

**lemma**  $\text{soundness: slow.check-equiv } n \varphi \psi \Longrightarrow \Phi.\text{lang}_{M2L} n \varphi = \Phi.\text{lang}_{M2L} n \psi$   
 $\langle \text{proof} \rangle$

**lemma**  $\text{completeness:}$

**assumes**  $\Phi.\text{lang}_{M2L} n \varphi = \Phi.\text{lang}_{M2L} n \psi \text{ wf-formula } n (FOr \varphi \psi)$   
**shows**  $\text{slow.check-equiv } n \varphi \psi$   
 $\langle \text{proof} \rangle$

$\langle ML \rangle$

**global-interpretation**  $D$ :  $\text{rexp-DA-no-post } \sigma \Sigma \text{ wf-atom } \Sigma \pi \text{ lookup } \lambda x. \text{ pnorm } (inorm x)$

$\lambda a r. \text{ pnorm } (\mathfrak{D} \Sigma a r) \text{ final alphabet.wf } (wf\text{-atom } \Sigma) n \text{ lang } \Sigma n n$   
**for**  $\Sigma :: 'a :: \text{linorder list}$  **and**  $n :: \text{nat}$   
**defines**

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

**definition** *check-eqv* **where**

$check\text{-}eqv \ n \ \varphi \ \psi \longleftrightarrow wf\text{-}formula \ n \ (FOr \ \varphi \ \psi) \wedge$   
 $fast.check\text{-}eqvRE \ Enum.enum \ n \ (Plus \ (rexp\text{-}of'' \ n \ (norm \ \varphi)) \ One) \ (Plus \ (rexp\text{-}of'' \ n \ (norm \ \psi)) \ One)$

**definition** *counterexample* **where**

$counterexample \ n \ \varphi \ \psi =$   
 $map\text{-}option \ (\lambda w. dec\text{-}interp \ n \ (FOV \ (FOr \ \varphi \ \psi)) \ w)$   
 $(fast.counterexampleRE \ Enum.enum \ n \ (Plus \ (rexp\text{-}of'' \ n \ (norm \ \varphi)) \ One) \ (Plus \ (rexp\text{-}of'' \ n \ (norm \ \psi)) \ One))$

**lemma** *soundness*:  $fast.check\text{-}eqv \ n \ \varphi \ \psi \Longrightarrow \Phi.lang_{M2L} \ n \ \varphi = \Phi.lang_{M2L} \ n \ \psi$   
 $\langle proof \rangle$

$\langle ML \rangle$

**global-interpretation** *D*:  $rexp-DA\text{-}no\text{-}post \ \sigma \Sigma \ wf\text{-}atom \ \Sigma \ \pi \ lookup$

$\lambda x. pnorm\text{-}dual \ (rexp\text{-}dual\text{-}of \ (inorm \ x)) \ \lambda a r. pnorm\text{-}dual \ (Co\mathfrak{D} \ \Sigma \ a \ r) \ \text{final}\text{-}dual$   
 $alphabet.wf\text{-}dual \ (wf\text{-}atom \ \Sigma) \ n \ lang\text{-}dual \ \Sigma \ n \ n$

**for**  $\Sigma :: 'a :: \text{linorder list}$  **and**  $n :: \text{nat}$

**defines**

$test = rexp-DA.test (final\text{-}dual :: 'a \text{ atom } rexp\text{-}dual \Rightarrow bool)$   
**and**  $step = rexp-DA.step (\sigma \Sigma) (\lambda a r. pnorm\text{-}dual (Co\mathfrak{D} \Sigma a r)) \text{ id } n$   
**and**  $closure = rexp-DA.closure (\sigma \Sigma) (\lambda a r. pnorm\text{-}dual (Co\mathfrak{D} \Sigma a r)) \text{ final}\text{-}dual \text{ id } n$   
**and**  $check\text{-}eqvRE = rexp-DA.check\text{-}eqv (\sigma \Sigma) (\lambda x. pnorm\text{-}dual (rexp\text{-}dual\text{-}of (inorm x))) (\lambda a r. pnorm\text{-}dual (Co\mathfrak{D} \Sigma a r)) \text{ final}\text{-}dual \text{ id } n$   
**and**  $test\text{-}invariant = rexp-DA.test\text{-}invariant (final\text{-}dual :: 'a \text{ atom } rexp\text{-}dual \Rightarrow bool) :: (( 'a \times bool \text{ list} ) \text{ list } \times -) \text{ list } \times - \Rightarrow bool$

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

**definition** *check-eqv where*

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

**definition** *counterexample where*

*counterexample n  $\varphi \psi =$*   
*map-option ( $\lambda w. \text{dec-interp } n (\text{FOV } (\text{FOr } \varphi \psi)) w$ )*  
*(dual.counterexampleRE Enum.enum n (Plus (rexp-of'' n (norm  $\varphi$ )) One) (Plus*  
*(rexp-of'' n (norm  $\psi$ )) One))*

**lemma** *soundness: dual.check-eqv n  $\varphi \psi \implies \Phi.\text{lang}_{M2L} n \varphi = \Phi.\text{lang}_{M2L} n \psi$*   
*<proof>*

*<ML>*

## 13 WS1S

### 13.1 Encodings

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

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

**declare** *smap-sconst[simp]*

**lemma** *(in wellorder) min-Least:*

$\llbracket \exists n. P n; \exists n. Q n \rrbracket \implies \text{min } (\text{Least } P) (\text{Least } Q) = (\text{LEAST } n. P n \vee Q n)$   
*<proof>*

**lemma** *sconst-collapse: y ## sconst y = sconst y*  
*<proof>*

**lemma** *shift-sconst-inj:  $\llbracket \text{length } x = \text{length } y; x @- \text{sconst } z = y @- \text{sconst } z \rrbracket$*   
 $\implies x = y$   
*<proof>*



**context** *formula*  
**begin**

**definition** *any*  $\equiv$  *hd*  $\Sigma$

**lemma** *any- $\Sigma$ [simp]*: *any*  $\in$  *set*  $\Sigma$   
 $\langle$ *proof* $\rangle$

**lemma** *any- $\sigma$ [simp]*: *length* *bs* = *n*  $\implies$  (*any*, *bs*)  $\in$  *set* ( $\sigma$   $\Sigma$  *n*)  
 $\langle$ *proof* $\rangle$

**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*  $\pi$  (*stream-enc* (*w*, *x* # *I*)) = *stream-enc* (*w*, *I*)  
 $\langle$ *proof* $\rangle$

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

**lemma** *ex-Loop-stream-enc*:

**assumes**  $\forall x \in \text{set } I. \text{ case } x \text{ of } \text{Inr } P \Rightarrow \text{finite } P \mid - \Rightarrow \text{True}$

**shows**  $\exists n. \text{sdrop } n$  (*stream-enc* (*w*, *I*)) = *sconst* (*any*, *replicate* (*length* *I*) *False*)  
 $\langle$ *proof* $\rangle$

**lemma** *length-snth-enc[simp]*: *length* (*snd* (*stream-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*:  $\llbracket \text{drop } n$  *w* @- *sconst* *y* = *sconst* *y*; *p* < *length* *w*;  $\neg$  *p* < *n*  $\rrbracket$   
 $\implies$  *w* ! *p* = *y*  
 $\langle$ *proof* $\rangle$

**lemma** *less-length-cut-same*:

$\llbracket (w$  @- *sconst* *y*) !! *p* = *a*  $\rrbracket \implies a = y \vee (p < \text{length } (\text{cut-same } y (w$  @- *sconst* *y*)) \wedge w ! *p* = *a*)  
 $\langle$ *proof* $\rangle$

**lemma** *less-length-cut-same-Inl*:

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

**lemma** *less-length-cut-same-Inr*:

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

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

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

**lemma** *cut-same-stop[simp]*:  
**assumes**  $x \neq y$   
**shows**  $\text{cut-same } x (xs @- y \#\# \text{sconst } x) = xs @ [y]$  (**is**  $\text{cut-same } x ?s = -$ )  
 $\langle \text{proof} \rangle$

**lemma** *cut-same-shift-sconst*:  $\exists n. w = \text{cut-same } x (w @- \text{sconst } x) @ \text{replicate } n$   
 $x$   
 $\langle \text{proof} \rangle$

**lemma** *set-cut-same*:  $\text{set } (\text{cut-same } x (w @- \text{sconst } x)) \subseteq \text{set } w$   
 $\langle \text{proof} \rangle$

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

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

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

**lemma** *enc-unique-length*:  $v \in \text{enc } (w, I) \Longrightarrow \forall v'. \text{length } v' = \text{length } v \wedge v' \in \text{enc } (w, I) \longrightarrow v = v'$   
 $\langle \text{proof} \rangle$

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

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

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

**lemma** *encD*:  
 $\llbracket v \in enc\ (w, I); (\forall x \in set\ I. case\ x\ of\ Inr\ P \Rightarrow finite\ P \mid - \Rightarrow True) \rrbracket \implies$   
 $v = map\ (case-prod\ (enc-atom\ I))\ (zip\ [0 ..< length\ v]\ (stake\ (length\ v)\ (w @-sconst\ any)))$   
 ⟨proof⟩

**lemma** *enc-Inl*:  $\llbracket x \in enc\ (w, I); (\forall x \in set\ I. case\ x\ of\ Inr\ P \Rightarrow finite\ P \mid - \Rightarrow True);$   
 $m < length\ I; I ! m = Inl\ p \rrbracket \implies p < length\ x \wedge snd\ (x ! p) ! m$   
 ⟨proof⟩

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

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

**lemma** *enc-stream-enc*:  
 $\llbracket (\forall x \in set\ I. case\ x\ of\ Inr\ P \Rightarrow finite\ P \mid - \Rightarrow True);$   
 $(\forall x \in set\ I'. case\ x\ of\ Inr\ P \Rightarrow finite\ P \mid - \Rightarrow True);$   
 $enc\ (w, I) = enc\ (w', I') \rrbracket \implies stream-enc\ (w, I) = stream-enc\ (w', I')$   
 ⟨proof⟩

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

**fun** *wf-interp-for-formula*  $:: 'a\ interp \Rightarrow 'a\ formula \Rightarrow bool$  **where**  
 $wf-interp-for-formula\ (w, I)\ \varphi =$   
 $(wf-interp\ w\ I \wedge$

$(\forall n \in FOV \varphi. \text{case } I ! n \text{ of } Inl - \Rightarrow True \mid - \Rightarrow False) \wedge$   
 $(\forall n \in SOV \varphi. \text{case } I ! n \text{ of } Inl - \Rightarrow False \mid Inr - \Rightarrow True))$

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

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

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

**lemma** *enc-set- $\sigma$* :  $\llbracket x \in \text{enc } (w, I); (\forall x \in \text{set } I. \text{case } x \text{ of } Inr \ P \Rightarrow \text{finite } P \mid - \Rightarrow True);$   
 $\text{length } I = n; a \in \text{set } x; \text{set } w \subseteq \text{set } \Sigma \rrbracket \Longrightarrow a \in \text{set } (\sigma \ \Sigma \ n)$   
*<proof>*

**definition** *positions-in-row*  $s \ i =$   
*Option.these (sset (smap2 ( $\lambda p \ (-, \text{bs})$ ). if nth bs i then Some p else None) nats s))*

**lemma** *positions-in-row*:  $\text{positions-in-row } s \ i = \{p. \text{snd } (s !! p) ! i\}$   
*<proof>*

**lemma** *positions-in-row-unique*:  $\exists ! p. \text{snd } (s !! p) ! i \Longrightarrow$   
 $\text{the-elm } (\text{positions-in-row } s \ i) = (\text{THE } p. \text{snd } (s !! p) ! i)$   
*<proof>*

**lemma** *positions-in-row-nth*:  $\exists ! p. \text{snd } (s !! p) ! i \Longrightarrow$   
 $\text{snd } (s !! \text{the-elm } (\text{positions-in-row } s \ i)) ! i$   
*<proof>*

**definition** *dec-word*  $s = \text{cut-same any } (\text{smap } \text{fst } s)$

**lemma** *dec-word-stream-enc*:  $\text{dec-word } (\text{stream-enc } (w, I)) = \text{cut-same any } (w \ @- \ \text{sconst any})$

$\langle \text{proof} \rangle$

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

**lemma** *stream-dec-Inl*:  $\llbracket i \in \text{FO}; i < n \rrbracket \implies \exists p. \text{stream-dec } n \ \text{FO } s \ ! \ i = \text{Inl } p$   
 $\langle \text{proof} \rangle$

**lemma** *stream-dec-not-Inr*:  $\llbracket \text{stream-dec } n \ \text{FO } s \ ! \ i = \text{Inr } P; i \in \text{FO}; i < n \rrbracket \implies$   
 $\text{False}$   
 $\langle \text{proof} \rangle$

**lemma** *stream-dec-Inr*:  $\llbracket i \notin \text{FO}; i < n \rrbracket \implies \exists P. \text{stream-dec } n \ \text{FO } s \ ! \ i = \text{Inr } P$   
 $\langle \text{proof} \rangle$

**lemma** *stream-dec-not-Inl*:  $\llbracket \text{stream-dec } n \ \text{FO } s \ ! \ i = \text{Inl } p; i \notin \text{FO}; i < n \rrbracket \implies$   
 $\text{False}$   
 $\langle \text{proof} \rangle$

**lemma** *Inr-dec-finite*:  $\llbracket \forall i < n. \text{finite } \{p. \text{snd } (s \ ! \ p) \ ! \ i\}; \text{Inr } P \in \text{set } (\text{stream-dec } n \ \text{FO } s) \rrbracket \implies$   
 $\text{finite } P$   
 $\langle \text{proof} \rangle$

**lemma** *enc-atom-dec*:  
 $\llbracket \forall p. \text{length } (\text{snd } (s \ ! \ p)) = n; \forall i \in \text{FO}. i < n \longrightarrow (\exists ! p. \text{snd } (s \ ! \ p) \ ! \ i); a = \text{fst } (s \ ! \ p) \rrbracket \implies$   
 $\text{enc-atom } (\text{stream-dec } n \ \text{FO } s) \ p \ a = s \ ! \ p$   
 $\langle \text{proof} \rangle$

**lemma** *length-stream-dec[simp]*:  $\text{length } (\text{stream-dec } n \ \text{FO } x) = n$   
 $\langle \text{proof} \rangle$

**lemma** *stream-enc-dec*:  
 $\llbracket \exists n. \text{sdrop } n \ (\text{smap } \text{fst } s) = \text{sconst } \text{any};$   
 $\text{stream-all } (\lambda x. \text{length } (\text{snd } x) = n) \ s; \forall i \in \text{FO}. (\exists ! p. \text{snd } (s \ ! \ p) \ ! \ i) \rrbracket \implies$   
 $\text{stream-enc } (\text{dec-word } s, \text{stream-dec } n \ \text{FO } s) = s$   
 $\langle \text{proof} \rangle$

**lemma** *stream-enc-unique*:  
 $i < \text{length } I \implies \exists p. I \ ! \ i = \text{Inl } p \implies \exists ! p. \text{snd } (\text{stream-enc } (w, I) \ ! \ p) \ ! \ i$   
 $\langle \text{proof} \rangle$

**lemma** *stream-dec-enc-Inl*:  
 $\llbracket \text{stream-dec } n \ \text{FO } (\text{stream-enc } (w, I)) \ ! \ i = \text{Inl } p'; I \ ! \ i = \text{Inl } p; i \in \text{FO}; i < n; \text{length } I = n \rrbracket \implies$   
 $p = p'$

$\langle \text{proof} \rangle$

**lemma** *stream-dec-enc-Inr*:

$\llbracket \text{stream-dec } n \text{ } FO \text{ (stream-enc } (w, I)) ! i = \text{Inr } P'; I ! i = \text{Inr } P; i \notin FO; i < n; \text{length } I = n \rrbracket \implies$   
 $P = P'$   
 $\langle \text{proof} \rangle$

**lemma** *Collect-snth*:  $\{p. P ((x \#\# s) !! p)\} \subseteq \{0\} \cup \text{Suc } ' \{p. P (s !! p)\}$

$\langle \text{proof} \rangle$

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

$\langle \text{proof} \rangle$

**lemma** *lang-ENC*:

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

**lemma** *lang-ENC-formula*:

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

## 13.2 Welldefinedness of enc wrt. Models

**lemma** *wf-interp-for-formula-FExists*:

$\llbracket \text{wf-formula } (\text{length } I) \text{ (FExists } \varphi) \rrbracket \implies$   
 $\text{wf-interp-for-formula } (w, I) \text{ (FExists } \varphi) \longleftrightarrow (\forall p. \text{wf-interp-for-formula } (w, \text{Inl } p \# I) \ \varphi)$   
 $\langle \text{proof} \rangle$

**lemma** *wf-interp-for-formula-any-Inl*:  $\text{wf-interp-for-formula } (w, \text{Inl } p \# I) \ \varphi \implies$

$\forall p. \text{wf-interp-for-formula } (w, \text{Inl } p \# I) \ \varphi$   
 $\langle \text{proof} \rangle$

**lemma** *wf-interp-for-formula-FEXISTS*:

$\llbracket \text{wf-formula } (\text{length } I) \text{ (FEXISTS } \varphi) \rrbracket \implies$   
 $\text{wf-interp-for-formula } (w, I) \text{ (FEXISTS } \varphi) \longleftrightarrow (\forall P. \text{finite } P \longrightarrow \text{wf-interp-for-formula } (w, \text{Inr } P \# I) \ \varphi)$   
 $\langle \text{proof} \rangle$

**lemma** *wf-interp-for-formula-any-Inr*:  $\text{wf-interp-for-formula } (w, \text{Inr } P \# I) \ \varphi \implies$

$\forall P. \text{finite } P \longrightarrow \text{wf-interp-for-formula } (w, \text{Inr } P \# I) \varphi$   
 ⟨proof⟩

**lemma** *wf-interp-for-formula-FOr*:

$\text{wf-interp-for-formula } (w, I) (\text{FOr } \varphi_1 \varphi_2) =$   
 $(\text{wf-interp-for-formula } (w, I) \varphi_1 \wedge \text{wf-interp-for-formula } (w, I) \varphi_2)$   
 ⟨proof⟩

**lemma** *wf-interp-for-formula-FAnd*:

$\text{wf-interp-for-formula } (w, I) (\text{FAnd } \varphi_1 \varphi_2) =$   
 $(\text{wf-interp-for-formula } (w, I) \varphi_1 \wedge \text{wf-interp-for-formula } (w, I) \varphi_2)$   
 ⟨proof⟩

**lemma** *enc-wf-interp*:

$\llbracket \text{wf-formula } (\text{length } I) \varphi; \text{wf-interp-for-formula } (w, I) \varphi; x \in \text{enc } (w, I) \rrbracket \Longrightarrow$   
 $\text{wf-interp-for-formula } (\text{dec-word } (x @- \text{sconst } (\text{any}, \text{replicate } (\text{length } I) \text{False})),$   
 $\text{stream-dec } (\text{length } I) (\text{FOV } \varphi) (x @- \text{sconst } (\text{any}, \text{replicate } (\text{length } I) \text{False})))$   
 $\varphi$   
 ⟨proof⟩

**lemma** *enc-atom-welldef*:  $\forall x a. \text{enc-atom } I x a = \text{enc-atom } I' x a \Longrightarrow m < \text{length } I \Longrightarrow$

$(\text{case } (I ! m, I' ! m) \text{ of } (\text{Inl } p, \text{Inl } q) \Rightarrow p = q \mid (\text{Inr } P, \text{Inr } Q) \Rightarrow P = Q \mid - \Rightarrow \text{True})$   
 ⟨proof⟩

**lemma** *stream-enc-welldef*:  $\llbracket \text{stream-enc } (w, I) = \text{stream-enc } (w', I'); \text{wf-formula } (\text{length } I) \varphi;$

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

**lemma** *lang<sub>WS1S</sub>-FOr*:

**assumes**  $\text{wf-formula } n (\text{FOr } \varphi_1 \varphi_2)$   
**shows**  $\text{lang}_{\text{WS1S}} n (\text{FOr } \varphi_1 \varphi_2) \subseteq$   
 $(\text{lang}_{\text{WS1S}} n \varphi_1 \cup \text{lang}_{\text{WS1S}} n \varphi_2) \cap \bigcup \{ \text{enc } (w, I) \mid w I. \text{length } I = n \wedge$   
 $\text{wf-interp-for-formula } (w, I) (\text{FOr } \varphi_1 \varphi_2) \}$   
**(is -  $\subseteq$  (?L1  $\cup$  ?L2)  $\cap$  ?ENC)**  
 ⟨proof⟩

**lemma** *lang<sub>WS1S</sub>-FAnd*:

**assumes**  $\text{wf-formula } n (\text{FAnd } \varphi_1 \varphi_2)$   
**shows**  $\text{lang}_{\text{WS1S}} n (\text{FAnd } \varphi_1 \varphi_2) \subseteq$   
 $\text{lang}_{\text{WS1S}} n \varphi_1 \cap \text{lang}_{\text{WS1S}} n \varphi_2 \cap \bigcup \{ \text{enc } (w, I) \mid w I. \text{length } I = n \wedge$   
 $\text{wf-interp-for-formula } (w, I) (\text{FAnd } \varphi_1 \varphi_2) \}$   
 ⟨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  $\varphi$* ) = *Inter* (*rexp.Not* (*rexp-of n  $\varphi$* )) (*ENC n* (*FOV* (*FNot  $\varphi$* )))  
| *rexp-of* *n* (*FOr  $\varphi_1 \varphi_2$* ) = *Inter* (*Plus* (*rexp-of n  $\varphi_1$* ) (*rexp-of n  $\varphi_2$* )) (*ENC n*  
(*FOV* (*FOr  $\varphi_1 \varphi_2$* )))  
| *rexp-of* *n* (*FAnd  $\varphi_1 \varphi_2$* ) = *INTERSECT* [*rexp-of n  $\varphi_1$* , *rexp-of n  $\varphi_2$* , *ENC n*  
(*FOV* (*FAnd  $\varphi_1 \varphi_2$* ))]  
| *rexp-of* *n* (*FExists  $\varphi$* ) = *samequot-exec* (*any*, *replicate n False*) (*Pr* (*rexp-of* (*n*  
+ 1)  *$\varphi$* ))  
| *rexp-of* *n* (*FEXISTS  $\varphi$* ) = *samequot-exec* (*any*, *replicate n False*) (*Pr* (*rexp-of*  
(*n* + 1)  *$\varphi$* ))

**fun** *rexp-of-alt* :: *nat*  $\Rightarrow$  '*a formula*  $\Rightarrow$  ('*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  $\varphi$* ) = *rexp.Not* (*rexp-of-alt n  $\varphi$* )  
| *rexp-of-alt* *n* (*FOr  $\varphi_1 \varphi_2$* ) = *Plus* (*rexp-of-alt n  $\varphi_1$* ) (*rexp-of-alt n  $\varphi_2$* )  
| *rexp-of-alt* *n* (*FAnd  $\varphi_1 \varphi_2$* ) = *Inter* (*rexp-of-alt n  $\varphi_1$* ) (*rexp-of-alt n  $\varphi_2$* )  
| *rexp-of-alt* *n* (*FExists  $\varphi$* ) = *samequot-exec* (*any*, *replicate n False*) (*Pr* (*Inter*  
(*rexp-of-alt* (*n* + 1)  *$\varphi$* ) (*ENC* (*Suc n*) (*FOV  $\varphi$* ))))  
| *rexp-of-alt* *n* (*FEXISTS  $\varphi$* ) = *samequot-exec* (*any*, *replicate n False*) (*Pr* (*Inter*  
(*rexp-of-alt* (*n* + 1)  *$\varphi$* ) (*ENC* (*Suc n*) (*FOV  $\varphi$* ))))

**definition** *rexp-of' n  $\varphi$*  = *Inter* (*rexp-of-alt n  $\varphi$* ) (*ENC n* (*FOV  $\varphi$* ))

**fun** *rexp-of-alt'* :: *nat*  $\Rightarrow$  '*a formula*  $\Rightarrow$  ('*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  $\varphi$* ) = *rexp.Not* (*rexp-of-alt' n  $\varphi$* )  
| *rexp-of-alt'* *n* (*FOr  $\varphi_1 \varphi_2$* ) = *Plus* (*rexp-of-alt' n  $\varphi_1$* ) (*rexp-of-alt' n  $\varphi_2$* )



$| \text{rexp-of-alt}' n (\text{FAnd } \varphi_1 \varphi_2) = \text{Inter } (\text{rexp-of-alt}' n \varphi_1) (\text{rexp-of-alt}' n \varphi_2)$   
 $| \text{rexp-of-alt}' n (\text{FExists } \varphi) = \text{samequot-exec } (\text{any, replicate } n \text{ False}) (\text{Pr } (\text{Inter } (\text{rexp-of-alt}' (n + 1) \varphi) (\text{ENC } (n + 1) \{0\})))$   
 $| \text{rexp-of-alt}' n (\text{FEXISTS } \varphi) = \text{samequot-exec } (\text{any, replicate } n \text{ False}) (\text{Pr } (\text{rexp-of-alt}' (n + 1) \varphi))$

**definition**  $\text{rexp-of}'' n \varphi = \text{Inter } (\text{rexp-of-alt}' n \varphi) (\text{ENC } n (\text{FOV } \varphi))$

**lemma** *enc-eqI*:

**assumes**  $x \in \text{enc } (w, I) \ x \in \text{enc } (w', I')$  *wf-interp-for-formula*  $(w, I) \ \varphi$   
*wf-interp-for-formula*  $(w', I') \ \varphi$   
 $\text{length } I = \text{length } I'$   
**shows**  $\text{enc } (w, I) = \text{enc } (w', I')$   
 $\langle \text{proof} \rangle$

**lemma** *enc-eq-welldef*:

$\llbracket \text{enc } (w, I) = \text{enc } (w', I'); \text{wf-formula } (\text{length } I) \ \varphi; \text{wf-interp-for-formula } (w, I) \ \varphi$   
 $;\text{wf-interp-for-formula } (w', I') \ \varphi \rrbracket \implies$   
 $(w, I) \models \varphi \longleftrightarrow (w', I') \models \varphi$   
 $\langle \text{proof} \rangle$

**lemma** *enc-welldef*:

$\llbracket x \in \text{enc } (w, I); x \in \text{enc } (w', I'); \text{length } I = \text{length } I'; \text{wf-formula } (\text{length } I) \ \varphi;$   
 $\text{wf-interp-for-formula } (w, I) \ \varphi; \text{wf-interp-for-formula } (w', I') \ \varphi \rrbracket \implies$   
 $(w, I) \models \varphi \longleftrightarrow (w', I') \models \varphi$   
 $\langle \text{proof} \rangle$

**lemma** *wf-rexp-of*:  $\text{wf-formula } n \ \varphi \implies \text{wf } n \ (\text{rexp-of } n \ \varphi)$

$\langle \text{proof} \rangle$

**theorem** *lang<sub>WS1S</sub>-rexp-of*:  $\text{wf-formula } n \ \varphi \implies \text{lang}_{\text{WS1S}} n \ \varphi = \text{lang } n \ (\text{rexp-of } n \ \varphi)$

(**is**  $- \implies - = ?L \ n \ \varphi$ )

$\langle \text{proof} \rangle$

**lemma** *wf-rexp-of-alt*:  $\text{wf-formula } n \ \varphi \implies \text{wf } n \ (\text{rexp-of-alt } n \ \varphi)$

$\langle \text{proof} \rangle$

**lemma** *wf-rexp-of'*:  $\text{wf-formula } n \ \varphi \implies \text{wf } n \ (\text{rexp-of}' n \ \varphi)$

$\langle \text{proof} \rangle$

**lemma** *wf-rexp-of-alt'*:  $\text{wf-formula } n \ \varphi \implies \text{wf } n \ (\text{rexp-of-alt}' n \ \varphi)$

$\langle \text{proof} \rangle$

**lemma** *wf-rexp-of''*:  $\text{wf-formula } n \ \varphi \implies \text{wf } n \ (\text{rexp-of}'' n \ \varphi)$

$\langle \text{proof} \rangle$

**lemma** *ENC-FNot*:  $\text{ENC } n \ (\text{FOV } (\text{FNot } \varphi)) = \text{ENC } n \ (\text{FOV } \varphi)$

$\langle \text{proof} \rangle$

**lemma** *ENC-FAnd*:

$\text{wf-formula } n \text{ (FAnd } \varphi \psi) \implies \text{lang } n \text{ (ENC } n \text{ (FOV (FAnd } \varphi \psi)) \subseteq \text{lang } n \text{ (ENC } n \text{ (FOV } \varphi)) \cap \text{lang } n \text{ (ENC } n \text{ (FOV } \psi))$   
 $\langle \text{proof} \rangle$

**lemma** *ENC-FOr*:

$\text{wf-formula } n \text{ (FOr } \varphi \psi) \implies \text{lang } n \text{ (ENC } n \text{ (FOV (FOr } \varphi \psi)) \subseteq \text{lang } n \text{ (ENC } n \text{ (FOV } \varphi)) \cap \text{lang } n \text{ (ENC } n \text{ (FOV } \psi))$   
 $\langle \text{proof} \rangle$

**lemma** *ENC-FExists*:

$\text{wf-formula } n \text{ (FExists } \varphi) \implies \text{lang } n \text{ (ENC } n \text{ (FOV (FExists } \varphi)) =$   
 $\text{SAMEQUOT (any, replicate } n \text{ False) (map } \pi \text{ ' lang (Suc } n \text{) (ENC (Suc } n \text{) (FOV } \varphi)) \text{ (is - } \implies ?L = ?R)$   
 $\langle \text{proof} \rangle$

**lemma** *ENC-FEXISTS*:

$\text{wf-formula } n \text{ (FEXISTS } \varphi) \implies \text{lang } n \text{ (ENC } n \text{ (FOV (FEXISTS } \varphi)) =$   
 $\text{SAMEQUOT (any, replicate } n \text{ False) (map } \pi \text{ ' lang (Suc } n \text{) (ENC (Suc } n \text{) (FOV } \varphi)) \text{ (is - } \implies ?L = ?R)$   
 $\langle \text{proof} \rangle$

**lemma** *lang<sub>WS1S</sub>-rexp-of-rexp-of'*:

$\text{wf-formula } n \varphi \implies \text{lang } n \text{ (rexp-of } n \varphi) = \text{lang } n \text{ (rexp-of' } n \varphi)$   
 $\langle \text{proof} \rangle$

**lemma** *SAMEQUOTO-UN[simp]*:  $\text{SAMEQUOT } x \text{ (} \bigcup y \in A. B y) = \text{SAMEQUOT } x \text{ (} B y)$

$\langle \text{proof} \rangle$

**lemma** *finite-positions-in-row[simp]*:

$n > 0 \implies \text{finite (positions-in-row (x @- sconst (any, replicate } n \text{ False)) } 0)$   
 $\langle \text{proof} \rangle$

**lemma** *fin-cut-same-snoc*:  $\text{fin-cut-same } x \text{ (xs @ [y]) = (if } x = y \text{ then fin-cut-same } x \text{ xs else xs @ [y])}$

$\langle \text{proof} \rangle$

**lemma** *fin-cut-same-idem*:  $\text{fin-cut-same } x \text{ (fin-cut-same } x \text{ xs) = fin-cut-same } x \text{ xs}$

$\langle \text{proof} \rangle$

**lemma** *cut-same-sconst*:  $\text{cut-same } x \text{ (xs @- sconst } x) = \text{fin-cut-same } x \text{ xs}$

$\langle \text{proof} \rangle$

**lemma** *length-cut-same*:  $\text{length (cut-same } x \text{ s) = (LEAST } n. \text{sdrop } n \text{ s = sconst } x)$

$\langle \text{proof} \rangle$

**lemma** *enc-alt: wf-interp w I*  $\implies$   
 $x \in \text{enc } (w, I) \iff x \text{ @- sconst } ((\text{any}, \text{replicate } (\text{length } I) \text{ False})) = \text{stream-enc}$   
 $(w, I)$   
 ⟨proof⟩

**lemma** *stream-stream-eqI*:  $\llbracket \forall (-, x) \in \text{sset } xs. x \neq []; \forall (-, x) \in \text{sset } ys. x \neq [];$   
 $\text{smap } (\lambda(-, x). \text{hd } x) \text{ } xs = \text{smap } (\lambda(-, x). \text{hd } x) \text{ } ys; \text{smap } \pi \text{ } xs = \text{smap } \pi \text{ } ys \rrbracket \implies$   
 $xs = ys$   
 ⟨proof⟩

**lemma** *project-enc-extend*:  
 fixes  $x I$   
 defines  $n \equiv \text{length } I$   
 defines  $z \equiv \lambda n. (\text{any}, \text{replicate } n \text{ False})$   
 defines  $I' \equiv \text{Inr } (\text{positions-in-row } (x \text{ @- sconst } (z (\text{Suc } n))) 0) \# I$   
 assumes *wf*: *wf-interp w I*  
 assumes *enc*: *fin-cut-same*  $(z n) (\text{map } \pi \text{ } x) \text{ @ replicate } m (z n) \in \text{enc } (w, I)$   
 assumes *nonempty*:  $\forall (-, x) \in \text{set } x. x \neq []$   
 shows  $x \in \text{enc } (w, I')$   
 ⟨proof⟩

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

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

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

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

**lemma** *map-project-ENC*:  
 fixes  $n$   
 assumes  $X \subseteq \{0 \dots n + 1\} \text{ } Z \subseteq \text{lists } ((\text{set } o \sigma \Sigma) (n + 1))$   
 defines  $z \equiv (\text{any}, \text{replicate } n \text{ False})$   
 shows  $\text{SAMEQUOT } z (\text{map } \pi \text{ ' } (Z \cap \text{lang } (n + 1) (\text{ENC } (n + 1) X))) =$   
 (if  $0 \in X$   
 then  $\text{SAMEQUOT } z (\text{map } \pi \text{ ' } (Z \cap \text{lang } (n + 1) (\text{ENC } (n + 1) \{0\}))) \cap \text{lang}$   
 $n (\text{ENC } n ((\lambda x. x - 1) \text{ ' } (X - \{0\})))$

$else\ SAMEQUOT\ z\ (map\ \pi\ 'Z) \cap\ lang\ n\ (ENC\ n\ ((\lambda x. x - 1)\ ' (X - \{0\})))$   
 $(is\ ?L = (if - then\ ?R1\ else\ ?R2))$   
 <proof>

**lemma**  $lang_{M2L}\text{-rexp-of}'\text{-rexp-of}''$ :  
 $wf\text{-formula}\ n\ \varphi \implies lang\ n\ (rexp\text{-of}'\ n\ \varphi) = lang\ n\ (rexp\text{-of}''\ n\ \varphi)$   
 <proof>

**theorem**  $lang_{WS1S}\text{-rexp-of}'$ :  $wf\text{-formula}\ n\ \varphi \implies lang_{WS1S}\ n\ \varphi = lang\ n\ (rexp\text{-of}'\ n\ \varphi)$   
 <proof>

**theorem**  $lang_{WS1S}\text{-rexp-of}''$ :  $wf\text{-formula}\ n\ \varphi \implies lang_{WS1S}\ n\ \varphi = lang\ n\ (rexp\text{-of}''\ n\ \varphi)$   
 <proof>

end

## 14 Normalization of WS1S Formulas

**fun**  $nNot$  **where**  
 $nNot\ (FNot\ \varphi) = \varphi$   
 $| nNot\ (FAnd\ \varphi1\ \varphi2) = FOr\ (nNot\ \varphi1)\ (nNot\ \varphi2)$   
 $| nNot\ (FOr\ \varphi1\ \varphi2) = FAnd\ (nNot\ \varphi1)\ (nNot\ \varphi2)$   
 $| nNot\ \varphi = FNot\ \varphi$

**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**  $satisfies\text{-}nNot[simp]$ :  $(w, I) \models nNot\ \varphi \longleftrightarrow (w, I) \models FNot\ \varphi$   
 <proof>

**lemma**  $FOV\text{-}nNot[simp]$ :  $FOV\ (nNot\ \varphi) = FOV\ (FNot\ \varphi)$   
 <proof>

**lemma**  $SOV\text{-}nNot[simp]$ :  $SOV\ (nNot\ \varphi) = SOV\ (FNot\ \varphi)$   
 <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<sub>WS1S</sub>-norm[simp]*: *lang<sub>WS1S</sub>*  $n$  (*norm*  $\varphi$ ) = *lang<sub>WS1S</sub>*  $n$   $\varphi$   
 ⟨*proof*⟩

end

## 15 Deciding Equivalence of WS1S Formulas

**global-interpretation** *embed2 set o  $\sigma$   $\Sigma$  wf-atom  $\Sigma$   $\pi$  lookup  $\varepsilon$   $\Sigma$  case-prod Singleton*

**for**  $\Sigma :: 'a :: linorder$  list

**defines**

$\mathfrak{D} = embed.lderiv$  lookup ( $\varepsilon$   $\Sigma$ )

**and**  $Co\mathfrak{D} = embed.lderiv-dual$  lookup ( $\varepsilon$   $\Sigma$ )

**and**  $r\mathfrak{D} = embed.rderiv$  lookup ( $\varepsilon$   $\Sigma$ )

**and**  $r\mathfrak{D}-add = embed2.rderiv-and-add$  lookup ( $\varepsilon$   $\Sigma$ )

**and**  $\mathfrak{Q} = embed2.samequot-exec$  lookup ( $\varepsilon$   $\Sigma$ ) (*case-prod Singleton*)

⟨*proof*⟩

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

**global-interpretation**  $\Phi$ : *formula* *Enum.enum*  $:: 'a :: \{enum, linorder\}$  list

**rewrites** *embed2.samequot-exec* lookup ( $\varepsilon$  (*Enum.enum*  $:: 'a :: \{enum, linorder\}$  list)) (*case-prod Singleton*) =  $\mathfrak{Q}$  *Enum.enum*

**defines**

*pre-wf-formula* =  $\Phi.pre-wf-formula$

**and** *wf-formula* =  $\Phi.wf-formula$

**and** *rexp-of* =  $\Phi.rexp-of$

**and** *rexp-of-alt* =  $\Phi.rexp-of-alt$

**and**  $rexp\text{-of}\text{-alt}' = \Phi.rexp\text{-of}\text{-alt}'$   
**and**  $rexp\text{-of}' = \Phi.rexp\text{-of}'$   
**and**  $rexp\text{-of}'' = \Phi.rexp\text{-of}''$   
**and**  $valid\text{-ENC} = \Phi.valid\text{-ENC}$   
**and**  $ENC = \Phi.ENC$   
**and**  $dec\text{-interp} = \Phi.stream\text{-dec}$   
**and**  $any = \Phi.any$   
 $\langle proof \rangle$

**lemmas**  $lang_{WS1S}\text{-rexp}\text{-of}\text{-norm} = trans[OF\ sym[OF\ \Phi.lang_{WS1S}\text{-norm}] \Phi.lang_{WS1S}\text{-rexp}\text{-of}]$   
**lemmas**  $lang_{WS1S}\text{-rexp}\text{-of}'\text{-norm} = trans[OF\ sym[OF\ \Phi.lang_{WS1S}\text{-norm}] \Phi.lang_{WS1S}\text{-rexp}\text{-of}' ]$   
**lemmas**  $lang_{WS1S}\text{-rexp}\text{-of}''\text{-norm} = trans[OF\ sym[OF\ \Phi.lang_{WS1S}\text{-norm}] \Phi.lang_{WS1S}\text{-rexp}\text{-of}'' ]$

$\langle ML \rangle$

**global-interpretation**  $D: rexp\text{-DFA } \sigma \Sigma wf\text{-atom } \Sigma \pi lookup \lambda x. \ll pnorm (inorm x) \gg$

$\lambda a r. \ll \mathfrak{D} \Sigma a r \gg final\ alphabet.wf (wf\text{-atom } \Sigma) n pnorm lang \Sigma n n$

**for**  $\Sigma :: 'a :: linorder\ list$  **and**  $n :: nat$

**defines**

$test = rexp\text{-DA}.test (final :: 'a\ atom\ rexp \Rightarrow bool)$

**and**  $step = rexp\text{-DA}.step (\sigma \Sigma) (\lambda a r. \ll \mathfrak{D} \Sigma a r \gg) pnorm n$

**and**  $closure = rexp\text{-DA}.closure (\sigma \Sigma) (\lambda a r. \ll \mathfrak{D} \Sigma a r \gg) final\ pnorm n$

**and**  $check\text{-eqv}RE = rexp\text{-DA}.check\text{-eqv} (\sigma \Sigma) (\lambda x. \ll pnorm (inorm x) \gg) (\lambda a r. \ll \mathfrak{D} \Sigma a r \gg) final\ pnorm n$

**and**  $test\text{-invariant} = rexp\text{-DA}.test\text{-invariant} (final :: 'a\ atom\ rexp \Rightarrow bool) ::$

$(('a \times bool\ list)\ list \times -)\ list \times - \Rightarrow bool$

**and**  $step\text{-invariant} = rexp\text{-DA}.step\text{-invariant} (\sigma \Sigma) (\lambda a r. \ll \mathfrak{D} \Sigma a r \gg) pnorm n$

**and**  $closure\text{-invariant} = rexp\text{-DA}.closure\text{-invariant} (\sigma \Sigma) (\lambda a r. \ll \mathfrak{D} \Sigma a r \gg) final\ pnorm n$

**and**  $counterexampleRE = rexp\text{-DA}.counterexample (\sigma \Sigma) (\lambda x. \ll pnorm (inorm x) \gg) (\lambda a r. \ll \mathfrak{D} \Sigma a r \gg) final\ pnorm n$

**and**  $reachable = rexp\text{-DA}.reachable (\sigma \Sigma) (\lambda x. \ll pnorm (inorm x) \gg) (\lambda a r. \ll \mathfrak{D} \Sigma a r \gg) pnorm n$

**and**  $automaton = rexp\text{-DA}.automaton (\sigma \Sigma) (\lambda x. \ll pnorm (inorm x) \gg) (\lambda a r. \ll \mathfrak{D} \Sigma a r \gg) pnorm n$

$\langle proof \rangle$

**definition**  $check\text{-eqv}$  **where**

$check\text{-eqv } n \varphi \psi \longleftrightarrow wf\text{-formula } n (FOr \varphi \psi) \wedge$

$slow.check\text{-eqv}RE\ Enum.enum\ n (rexp\text{-of}''\ n (norm \varphi)) (rexp\text{-of}''\ n (norm \psi))$

**definition**  $counterexample$  **where**

$counterexample\ n \varphi \psi =$

$map\text{-option} (\lambda w. dec\text{-interp } n (FOV (FOr \varphi \psi)) (w @- sconst (any, replicate\ n\ False)))$

$(slow.counterexampleRE\ Enum.enum\ n (rexp\text{-of}''\ n (norm \varphi)) (rexp\text{-of}''\ n (norm \psi)))$

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

**lemma completeness:**

**assumes**  $\Phi.lang_{WS1S}\ n\ \varphi = \Phi.lang_{WS1S}\ n\ \psi\ wf\text{-}formula\ n\ (FOr\ \varphi\ \psi)$

**shows**  $slow.check\text{-}eqv\ n\ \varphi\ \psi$

⟨proof⟩

⟨ML⟩

**global-interpretation**  $D$ :  $rexp\text{-}DA\text{-}no\text{-}post\ \sigma\ \Sigma\ wf\text{-}atom\ \Sigma\ \pi\ lookup\ \lambda x. pnorm$   
 ( $inorm\ x$ )

$\lambda a\ r. pnorm\ (\mathfrak{D}\ \Sigma\ a\ r)\ final\ alphabet.wf\ (wf\text{-}atom\ \Sigma)\ n\ lang\ \Sigma\ n\ n$

**for**  $\Sigma :: 'a :: linorder\ list$  **and**  $n :: nat$

**defines**

$test = rexp\text{-}DA.test\ (final :: 'a\ atom\ rexp \Rightarrow bool)$

**and**  $step = rexp\text{-}DA.step\ (\sigma\ \Sigma)\ (\lambda a\ r. pnorm\ (\mathfrak{D}\ \Sigma\ a\ r))\ id\ n$

**and**  $closure = rexp\text{-}DA.closure\ (\sigma\ \Sigma)\ (\lambda a\ r. pnorm\ (\mathfrak{D}\ \Sigma\ a\ r))\ final\ id\ n$

**and**  $check\text{-}eqvRE = rexp\text{-}DA.check\text{-}eqv\ (\sigma\ \Sigma)\ (\lambda x. pnorm\ (inorm\ x))\ (\lambda a\ r. pnorm\ (\mathfrak{D}\ \Sigma\ a\ r))\ final\ id\ n$

**and**  $test\text{-}invariant = rexp\text{-}DA.test\text{-}invariant\ (final :: 'a\ atom\ rexp \Rightarrow bool) ::$

$(('a \times bool\ list)\ list \times -)\ list \times - \Rightarrow bool$

**and**  $step\text{-}invariant = rexp\text{-}DA.step\text{-}invariant\ (\sigma\ \Sigma)\ (\lambda a\ r. pnorm\ (\mathfrak{D}\ \Sigma\ a\ r))\ id\ n$

**and**  $closure\text{-}invariant = rexp\text{-}DA.closure\text{-}invariant\ (\sigma\ \Sigma)\ (\lambda a\ r. pnorm\ (\mathfrak{D}\ \Sigma\ a\ r))\ final\ id\ n$

**and**  $counterexampleRE = rexp\text{-}DA.counterexample\ (\sigma\ \Sigma)\ (\lambda x. pnorm\ (inorm\ x))\ (\lambda a\ r. pnorm\ (\mathfrak{D}\ \Sigma\ a\ r))\ final\ id\ n$

**and**  $reachable = rexp\text{-}DA.reachable\ (\sigma\ \Sigma)\ (\lambda x. pnorm\ (inorm\ x))\ (\lambda a\ r. pnorm\ (\mathfrak{D}\ \Sigma\ a\ r))\ id\ n$

**and**  $automaton = rexp\text{-}DA.automaton\ (\sigma\ \Sigma)\ (\lambda x. pnorm\ (inorm\ x))\ (\lambda a\ r. pnorm\ (\mathfrak{D}\ \Sigma\ a\ r))\ id\ n$

⟨proof⟩

**definition check-eqv where**

$check\text{-}eqv\ n\ \varphi\ \psi \longleftrightarrow wf\text{-}formula\ n\ (FOr\ \varphi\ \psi) \wedge$

$fast.check\text{-}eqvRE\ Enum.enum\ n\ (rexp\text{-}of''\ n\ (norm\ \varphi))\ (rexp\text{-}of''\ n\ (norm\ \psi))$

**definition counterexample where**

$counterexample\ n\ \varphi\ \psi =$

$map\text{-}option\ (\lambda w. dec\text{-}interp\ n\ (FOV\ (FOr\ \varphi\ \psi))\ (w\ @-\ sconst\ (any,\ replicate\ n\ False)))$

$(fast.counterexampleRE\ Enum.enum\ n\ (rexp\text{-}of''\ n\ (norm\ \varphi))\ (rexp\text{-}of''\ n\ (norm\ \psi)))$

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

$\langle ML \rangle$

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

**definition** *check-equiv where*

*check-equiv*  $n \varphi \psi \longleftrightarrow \text{wf-formula } n (\text{FOr } \varphi \psi) \wedge$   
 $\text{dual.check-equivRE Enum.enum } n (\text{rexp-of'' } n (\text{norm } \varphi)) (\text{rexp-of'' } n (\text{norm } \psi))$

**definition** *counterexample where*

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

**lemma** *soundness*:  $\text{dual.check-equiv } n \varphi \psi \Longrightarrow \Phi.\text{lang}_{\text{WS1S}} n \varphi = \Phi.\text{lang}_{\text{WS1S}} n$   
 $\psi$   
   $\langle \text{proof} \rangle$

$\langle ML \rangle$



## 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.