

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

Dmitriy Traytel and Tobias Nipkow

June 24, 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 . . . . .	7
1.4	Right-Quotients of Languages . . . . .	8
1.5	Two-Sided-Quotients of Languages . . . . .	10
1.6	Arden's Lemma . . . . .	11
1.7	Lists of Fixed Length . . . . .	14
<b>2</b>	<b>II-Extended Regular Expressions</b>	<b>14</b>
2.1	Syntax of regular expressions . . . . .	14
2.2	ACI normalization . . . . .	15

2.3	Finality . . . . .	19
2.4	Wellformedness w.r.t. an alphabet . . . . .	19
2.5	Language . . . . .	21
<b>3</b>	<b>Derivatives of <math>\Pi</math>-Extended Regular Expressions</b>	<b>23</b>
3.1	Syntactic Derivatives . . . . .	23
3.2	Finiteness of ACI-Equivalent Derivatives . . . . .	23
3.3	Wellformedness and language of derivatives . . . . .	28
3.4	Deriving preserves ACI-equivalence . . . . .	29
<b>4</b>	<b>Some Useful Regular Operators</b>	<b>31</b>
4.1	Quotienting by the same letter . . . . .	34
4.2	Suffix and Prefix Languages . . . . .	40
<b>5</b>	<b><math>\Pi</math>-Extended Dual Regular Expressions</b>	<b>41</b>
5.1	Syntax of regular expressions . . . . .	41
<b>6</b>	<b>Deciding Equivalence of <math>\Pi</math>-Extended Regular Expressions</b>	<b>49</b>
<b>7</b>	<b>Initial Normalization of the Input</b>	<b>61</b>
<b>8</b>	<b>Partial Derivatives-like Normalization</b>	<b>67</b>
<b>9</b>	<b>Monadic Second-Order Logic Formulas</b>	<b>69</b>
9.1	Interpretations and Encodings . . . . .	69
9.2	Syntax and Semantics of MSO . . . . .	69
9.3	ENC . . . . .	71
<b>10</b>	<b>M2L</b>	<b>73</b>
10.1	Encodings . . . . .	73
10.2	Welldefinedness of enc wrt. Models . . . . .	79
10.3	From M2L to Regular expressions . . . . .	84
<b>11</b>	<b>Normalization of M2L Formulas</b>	<b>102</b>
<b>12</b>	<b>Deciding Equivalence of M2L Formulas</b>	<b>103</b>
<b>13</b>	<b>WS1S</b>	<b>107</b>
13.1	Encodings . . . . .	107
13.2	Welldefinedness of enc wrt. Models . . . . .	118
13.3	From WS1S to Regular expressions . . . . .	122
<b>14</b>	<b>Normalization of WS1S Formulas</b>	<b>143</b>
<b>15</b>	<b>Deciding Equivalence of WS1S Formulas</b>	<b>144</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 | xs ys. xs:A & ys:B}

**lemma** [code]:

A @@ B = (%(xs, ys). xs @ ys) ' (A  $\times$  B)  
**unfolding** conc-def **by** auto

**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)

**by** (induct n) auto

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

star A = ( $\bigcup$  n. A ^^ n)

## 1.1 Concatenation of Languages

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

**by** (auto simp add: conc-def)

**lemma** concE[elim]:

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

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

**using** assms **by** (auto simp: conc-def)

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

**by** (auto simp: conc-def)

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

**by** auto

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

**by** (simp-all add:conc-def)

**lemma conc-assoc:**  $(A \text{ @@ } B) \text{ @@ } C = A \text{ @@ } (B \text{ @@ } C)$   
**by** (*auto elim!*: concE) (*simp only*: append-assoc[symmetric] concI)

**lemma conc-Un-distrib:**  
**shows**  $A \text{ @@ } (B \cup C) = A \text{ @@ } B \cup A \text{ @@ } C$   
**and**  $(A \cup B) \text{ @@ } C = A \text{ @@ } C \cup B \text{ @@ } C$   
**by** *auto*

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

**lemma hom-image-conc:**  $\llbracket \bigwedge xs \ ys. f (xs \text{ @ } ys) = f xs \text{ @ } f ys \rrbracket \implies f \text{ ' } (A \text{ @@ } B) = f \text{ ' } A \text{ @@ } f \text{ ' } B$   
**unfolding** conc-def **by** (*auto simp*: image-iff) metis

**lemma map-image-conc[simp]:**  $\text{map } f \text{ ' } (A \text{ @@ } B) = \text{map } f \text{ ' } A \text{ @@ } \text{map } f \text{ ' } B$   
**by** (*simp add*: hom-image-conc)

**lemma conc-subset-lists:**  $A \subseteq \text{lists } S \implies B \subseteq \text{lists } S \implies A \text{ @@ } B \subseteq \text{lists } S$   
**by** (*fastforce simp*: conc-def in-lists-conv-set)

## 1.2 Iteration of Languages

**lemma lang-pow-add:**  $A \text{ ^^ } (n + m) = A \text{ ^^ } n \text{ @@ } A \text{ ^^ } m$   
**by** (*induct n*) (*auto simp*: conc-assoc)

**lemma lang-pow-simps:**  $(A \text{ ^^ } \text{Suc } n) = (A \text{ ^^ } n \text{ @@ } A)$   
**using** lang-pow-add[*of n Suc 0 A*] **by** *auto*

**lemma lang-pow-empty:**  $\{\} \text{ ^^ } n = (\text{if } n = 0 \text{ then } \{\} \text{ else } \{\})$   
**by** (*induct n*) *auto*

**lemma lang-pow-empty-Suc[simp]:**  $(\{\} \text{ ' } a \text{ lang}) \text{ ^^ } \text{Suc } n = \{\}$   
**by** (*simp add*: lang-pow-empty)

**lemma conc-pow-comm:**  
**shows**  $A \text{ @@ } (A \text{ ^^ } n) = (A \text{ ^^ } n) \text{ @@ } A$   
**by** (*induct n*) (*simp-all add*: conc-assoc[symmetric])

**lemma length-lang-pow-ub:**  
 $\text{ALL } w : A. \text{length } w \leq k \implies w : A \text{ ^^ } n \implies \text{length } w \leq k * n$   
**by** (*induct n arbitrary*: w) (*fastforce simp*: conc-def)+

**lemma length-lang-pow-lb:**  
 $\text{ALL } w : A. \text{length } w \geq k \implies w : A \text{ ^^ } n \implies \text{length } w \geq k * n$   
**by** (*induct n arbitrary*: w) (*fastforce simp*: conc-def)+

**lemma lang-pow-subset-lists:**  $A \subseteq \text{lists } S \implies A^{\wedge n} \subseteq \text{lists } S$   
**by** (*induct n*) (*auto simp: conc-subset-lists*)

**lemma star-subset-lists:**  $A \subseteq \text{lists } S \implies \text{star } A \subseteq \text{lists } S$   
**unfolding** *star-def* **by**(*blast dest: lang-pow-subset-lists*)

**lemma star-if-lang-pow[*simp*]:**  $w : A^{\wedge n} \implies w : \text{star } A$   
**by** (*auto simp: star-def*)

**lemma Nil-in-star[*iff*]:**  $[] : \text{star } A$   
**proof** (*rule star-if-lang-pow*)  
**show**  $[] : A^{\wedge 0}$  **by** *simp*  
**qed**

**lemma star-if-lang[*simp*]:** **assumes**  $w : A$  **shows**  $w : \text{star } A$   
**proof** (*rule star-if-lang-pow*)  
**show**  $w : A^{\wedge 1}$  **using**  $\langle w : A \rangle$  **by** *simp*  
**qed**

**lemma append-in-starI[*simp*]:**  
**assumes**  $u : \text{star } A$  **and**  $v : \text{star } A$  **shows**  $u@v : \text{star } A$   
**proof** –  
**from**  $\langle u : \text{star } A \rangle$  **obtain**  $m$  **where**  $u : A^{\wedge m}$  **by** (*auto simp: star-def*)  
**moreover**  
**from**  $\langle v : \text{star } A \rangle$  **obtain**  $n$  **where**  $v : A^{\wedge n}$  **by** (*auto simp: star-def*)  
**ultimately have**  $u@v : A^{\wedge (m+n)}$  **by** (*simp add: lang-pow-add*)  
**thus** *?thesis* **by** *simp*  
**qed**

**lemma conc-star-star:**  $\text{star } A @@ \text{star } A = \text{star } A$   
**by** (*auto simp: conc-def*)

**lemma conc-star-comm:**  
**shows**  $A @@ \text{star } A = \text{star } A @@ A$   
**unfolding** *star-def conc-pow-comm conc-UNION-distrib*  
**by** *simp*

**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** –  
**{** **fix**  $n$  **have**  $w : A^{\wedge n} \implies P w$   
**by** (*induct n arbitrary: w*) (*auto intro: \langle P [] \rangle step star-if-lang-pow*) **}**  
**with**  $\langle w : \text{star } A \rangle$  **show**  $P w$  **by** (*auto simp: star-def*)  
**qed**

```

lemma star-empty[simp]: star {} = {}
  by (auto elim: star-induct)

lemma star-epsilon[simp]: star {} = {}
  by (auto elim: star-induct)

lemma star-idemp[simp]: star (star A) = star A
  by (auto elim: star-induct)

lemma star-unfold-left: star A = A @@ star A ∪ {} (is ?L = ?R)
proof
  show ?L ⊆ ?R by (rule, erule star-induct) auto
qed auto

lemma concat-in-star: set ws ⊆ A ⇒ concat ws : star A
  by (induct ws) simp-all

lemma in-star-iff-concat:
  w : star A = (EX ws. set ws ⊆ A & w = concat ws & [] ∉ set ws)
  (is - = (EX ws. ?R w ws))
proof
  assume w : star A thus EX ws. ?R w ws
  proof induct
    case Nil have ?R [] [] by simp
    thus ?case ..
  next
    case (append u v)
    moreover
    then obtain ws where set ws ⊆ A ∧ v = concat ws ∧ [] ∉ set ws by blast
    ultimately have ?R (u@v) (if u = [] then ws else u#ws) by auto
    thus ?case ..
  qed
next
  assume EX ws. ?R w ws thus w : star A
  by (auto simp: concat-in-star)
qed

lemma star-conv-concat: star A = {concat ws | ws. set ws ⊆ A & [] ∉ set ws}
  by (fastforce simp: in-star-iff-concat)

lemma star-insert-eps[simp]: star (insert [] A) = star(A)
proof -
  { fix us
    have set us ⊆ insert [] A ⇒ EX vs. concat us = concat vs ∧ set vs ⊆ A
      (is ?P ⇒ EX vs. ?Q vs)
    proof
      let ?vs = filter (%u. u ≠ []) us
      show ?P ⇒ ?Q ?vs by (induct us) auto
    }
qed

```

} thus ?thesis by (auto simp: star-conv-concat)  
qed

**lemma star-decom:**

assumes  $a: x \in \text{star } A \ x \neq []$   
shows  $\exists a \ b. x = a @ b \wedge a \neq [] \wedge a \in A \wedge b \in \text{star } A$   
using a by (induct rule: star-induct) (blast)+

**lemma Ball-starI:**  $\forall a \in \text{set } as. [a] \in A \implies as \in \text{star } A$   
by (induct as rule: rev-induct) auto

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

by (auto elim: star-induct) (auto elim: star-induct simp del: map-append simp: map-append[symmetric] intro!: imageI)

### 1.3 Left-Quotients of Languages

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

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

**abbreviation**

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

where

$l\text{Quotss } s \ As \equiv \bigcup (l\text{Quots } s \text{ ' } As)$

**lemma lQuot-empty[simp]:**  $l\text{Quot } a \ \{\} = \{\}$

**and lQuot-epsilon[simp]:**  $l\text{Quot } a \ \{\} = \{\}$

**and lQuot-char[simp]:**  $l\text{Quot } a \ \{[b]\} = (\text{if } a = b \text{ then } \{\} \text{ else } \{\})$

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

**and lQuot-union[simp]:**  $l\text{Quot } a \ (A \cup B) = l\text{Quot } a \ A \cup l\text{Quot } a \ B$

**and lQuot-inter[simp]:**  $l\text{Quot } a \ (A \cap B) = l\text{Quot } a \ A \cap l\text{Quot } a \ B$

**and lQuot-compl[simp]:**  $l\text{Quot } a \ (-A) = - \ l\text{Quot } a \ A$

by (auto simp: lQuot-def)

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

**proof**

fix w assume  $w \in ?L$

then obtain u v where  $w = u @ v \ a \ \# \ u \in A \ v \in B$

by (auto simp: lQuot-def)

then have  $a \ \# \ w \in A \ @\@ \ B$

by (auto intro: concI[of a # u, simplified])

thus  $w \in ?R$  by (auto simp: lQuot-def)

qed

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

then  $lQuot\ c\ B$  else  $\{\}$ )  
**unfolding**  $lQuot\text{-}def\ conc\text{-}def$   
**by** (*auto simp add: Cons-eq-append-conv*)

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

**proof** –

**have**  $incl: [] \in A \implies lQuot\ c\ (star\ A) \subseteq (lQuot\ c\ A)\ @@\ star\ A$

**unfolding**  $lQuot\text{-}def\ conc\text{-}def$

**apply**(*auto simp add: Cons-eq-append-conv*)

**apply**(*drule star-decom*)

**apply**(*auto simp add: Cons-eq-append-conv*)

**done**

**have**  $lQuot\ c\ (star\ A) = lQuot\ c\ (A\ @@\ star\ A \cup \{\})$

**by** (*simp only: star-unfold-left[symmetric]*)

**also have**  $\dots = lQuot\ c\ (A\ @@\ star\ A)$

**by** (*simp only: lQuot-union*) (*simp*)

**also have**  $\dots = (lQuot\ c\ A)\ @@\ (star\ A) \cup (if\ [] \in A\ then\ lQuot\ c\ (star\ A)\ else\ \{\})$

**by** *simp*

**also have**  $\dots = (lQuot\ c\ A)\ @@\ star\ A$

**using** *incl* **by** *auto*

**finally show**  $lQuot\ c\ (star\ A) = (lQuot\ c\ A)\ @@\ star\ A .$

**qed**

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

**by**(*auto simp add: lQuot-def*)

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

**by**(*auto simp add: lQuot-def*)

**lemma**  $lQuots\text{-}simps$  [*simp*]:

**shows**  $lQuots\ []\ A = A$

**and**  $lQuots\ (c\ \# s)\ A = lQuots\ s\ (lQuot\ c\ A)$

**and**  $lQuots\ (s1\ @\ s2)\ A = lQuots\ s2\ (lQuots\ s1\ A)$

**unfolding**  $lQuots\text{-}def\ lQuot\text{-}def$  **by** *auto*

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

**by** (*induct w arbitrary: v A*) (*auto simp add: lQuot-def*)

## 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**  $rQuot\text{-}rev\text{-}lQuot$ :  $rQuot x A = rev ' lQuot x (rev ' A)$   
**unfolding**  $rQuot\text{-}def$   $lQuot\text{-}def$  **by** (*auto simp: rev-swap[symmetric]*)

**lemma**  $rQuots\text{-}rev\text{-}lQuots$ :  $rQuots x A = rev ' lQuots x (rev ' A)$   
**unfolding**  $rQuots\text{-}def$   $lQuots\text{-}def$  **by** (*auto simp: rev-swap[symmetric]*)

**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*]:  $rQuot a (-A) = - rQuot a A$   
**by** (*auto simp: rQuot-def*)

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

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

**lemma**  $rev\text{-}simp\text{-}invert$ :  $(xs @ [x] = rev zs) = (zs = x \# rev xs)$   
**by** (*induct zs auto*)

**lemma**  $rev\text{-}append\text{-}invert$ :  $(xs @ ys = rev zs) = (zs = rev ys @ rev xs)$   
**by** (*induct xs arbitrary: ys rule: rev-induct auto*)

**lemma**  $image\text{-}rev\text{-}lists$ [*simp*]:  $rev ' lists S = lists S$   
**proof** (*intro set-eqI*)  
**fix**  $xs$   
**show**  $xs \in rev ' lists S \longleftrightarrow xs \in lists S$   
**proof** (*induct xs rule: rev-induct*)  
**case** (*snoc x xs*)  
**thus** *?case* **by** (*auto intro!: image-eqI[of - rev] simp: rev-simp-invert*)  
**qed** *simp*  
**qed**

**lemma**  $image\text{-}rev\text{-}conc$ [*simp*]:  $rev ' (A @@@ B) = rev ' B @@@ rev ' A$   
**by** *auto* (*auto simp: rev-append[symmetric] simp del: rev-append*)

**lemma**  $image\text{-}rev\text{-}star$ [*simp*]:  $rev ' star A = star (rev ' A)$   
**by** (*auto elim: star-induct*) (*auto elim: star-induct simp: rev-append[symmetric] simp del: rev-append*)

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

**unfolding**  $rQuot\text{-}rev\text{-}lQuot$  **by** (*auto simp: image-image image-Un*)

**lemma**  $rQuot\text{-}star$  [*simp*]:  $rQuot\ c\ (star\ A) = star\ A\ @@\ (rQuot\ c\ A)$   
**unfolding**  $rQuot\text{-}rev\text{-}lQuot$  **by** (*auto simp: image-image*)

**lemma**  $rQuot\text{-}diff$  [*simp*]:  $rQuot\ c\ (A - B) = rQuot\ c\ A - rQuot\ c\ B$   
**by** (*auto simp add: rQuot-def*)

**lemma**  $rQuot\text{-}lists$  [*simp*]:  $c : S \implies rQuot\ c\ (lists\ S) = lists\ S$   
**by** (*auto simp add: rQuot-def*)

**lemma**  $rQuots\text{-}simps$  [*simp*]:  
**shows**  $rQuots\ []\ A = A$   
**and**  $rQuots\ (c\ \#s)\ A = rQuots\ s\ (rQuot\ c\ A)$   
**and**  $rQuots\ (s1\ @\ s2)\ A = rQuots\ s2\ (rQuots\ s1\ A)$   
**unfolding**  $rQuots\text{-}def\ rQuot\text{-}def$  **by** *auto*

**lemma**  $rQuots\text{-}append$  [*iff*]:  $v \in rQuots\ w\ A \longleftrightarrow v\ @\ rev\ w \in A$   
**by** (*induct w arbitrary: v A auto simp add: rQuot-def*)

## 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**  $biQuot\text{-}rQuot\text{-}lQuot$ :  $biQuot\ x\ y\ A = rQuot\ y\ (lQuot\ x\ A)$   
**unfolding**  $biQuot\text{-}def\ rQuot\text{-}def\ lQuot\text{-}def$  **by** *auto*

**lemma**  $biQuot\text{-}lQuot\text{-}rQuot$ :  $biQuot\ x\ y\ A = lQuot\ x\ (rQuot\ y\ A)$   
**unfolding**  $biQuot\text{-}def\ rQuot\text{-}def\ lQuot\text{-}def$  **by** *auto*

**lemma**  $biQuots\text{-}rQuots\text{-}lQuots$ :  $biQuots\ x\ y\ A = rQuots\ y\ (lQuots\ x\ A)$   
**unfolding**  $biQuots\text{-}def\ rQuots\text{-}def\ lQuots\text{-}def$  **by** *auto*

**lemma**  $biQuots\text{-}lQuots\text{-}rQuots$ :  $biQuots\ x\ y\ A = lQuots\ x\ (rQuots\ y\ A)$   
**unfolding**  $biQuots\text{-}def\ rQuots\text{-}def\ lQuots\text{-}def$  **by** *auto*

**lemma**  $biQuot\text{-}empty$  [*simp*]:  $biQuot\ a\ b\ \{\} = \{\}$   
**and**  $biQuot\text{-}epsilon$  [*simp*]:  $biQuot\ a\ b\ \{\}\} = \{\}$   
**and**  $biQuot\text{-}char$  [*simp*]:  $biQuot\ a\ b\ \{[c]\} = \{\}$   
**and**  $biQuot\text{-}union$  [*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]:  $biQuot\ a\ b\ (-A) = -\ biQuot\ a\ b\ A$   
**by** (*auto simp: biQuot-def*)

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

**lemma** *biQuot-star* [simp]:  $biQuot\ a\ b\ (star\ A) = biQuot\ a\ b\ A \cup lQuot\ a\ A\ @@\$   
 $star\ A\ @@\ rQuot\ b\ A$   
**unfolding** *biQuot-rQuot-lQuot* **by** *auto*

**lemma** *biQuot-diff*[simp]:  $biQuot\ a\ b\ (A - B) = biQuot\ a\ b\ A - biQuot\ a\ b\ B$   
**by**(*auto simp add: biQuot-def*)

**lemma** *biQuot-lists*[simp]:  $a : S \implies b : S \implies biQuot\ a\ b\ (lists\ S) = lists\ S$   
**by**(*auto simp add: biQuot-def*)

**lemma** *biQuots-simps* [simp]:  
**shows** *biQuots*  $\square\ \square\ A = A$   
**and** *biQuots*  $(a\ \#\ as)\ (b\ \#\ bs)\ A = biQuots\ as\ bs\ (biQuot\ a\ b\ A)$   
**and**  $\llbracket length\ s1 = length\ t1; length\ s2 = length\ t2 \rrbracket \implies$   
 $biQuots\ (s1\ @\ s2)\ (t1\ @\ t2)\ A = biQuots\ s2\ t2\ (biQuots\ s1\ t1\ A)$   
**unfolding** *biQuots-def biQuot-def* **by** *auto*

**lemma** *biQuots-append*[iff]:  $v \in biQuots\ u\ w\ A \longleftrightarrow u\ @\ v\ @\ rev\ w \in A$   
**unfolding** *biQuots-def* **by** *auto*

## 1.6 Arden's Lemma

**lemma** *arden-helper*:  
**assumes** *eq*:  $X = A @@ X \cup B$   
**shows**  $X = (A \ \hat{\ } \ Suc\ n) @@ X \cup (\bigcup_{m \leq n}. (A \ \hat{\ } \ m) @@ B)$   
**proof** (*induct n*)  
**case**  $0$   
**show**  $X = (A \ \hat{\ } \ Suc\ 0) @@ X \cup (\bigcup_{m \leq 0}. (A \ \hat{\ } \ m) @@ B)$   
**using** *eq* **by** *simp*  
**next**  
**case**  $(Suc\ n)$   
**have** *ih*:  $X = (A \ \hat{\ } \ Suc\ n) @@ X \cup (\bigcup_{m \leq n}. (A \ \hat{\ } \ m) @@ B)$  **by** *fact*  
**also have**  $\dots = (A \ \hat{\ } \ Suc\ n) @@ (A @@ X \cup B) \cup (\bigcup_{m \leq n}. (A \ \hat{\ } \ m) @@ B)$   
**using** *eq* **by** *simp*  
**also have**  $\dots = (A \ \hat{\ } \ Suc\ (Suc\ n)) @@ X \cup ((A \ \hat{\ } \ Suc\ n) @@ B) \cup (\bigcup_{m \leq n}. (A \ \hat{\ } \ m) @@ B)$   
**by** (*simp add: conc-Un-distrib conc-assoc[symmetric] conc-pow-comm*)

also have  $\dots = (A \hat{\hat{}} \text{Suc} (\text{Suc } n)) \text{@@} X \cup (\bigcup m \leq \text{Suc } n. (A \hat{\hat{}} m) \text{@@} B)$   
 by *(auto simp add: atMost-Suc)*  
 finally show  $X = (A \hat{\hat{}} \text{Suc} (\text{Suc } n)) \text{@@} X \cup (\bigcup m \leq \text{Suc } n. (A \hat{\hat{}} m) \text{@@} B)$

qed

**lemma** *Arden*:

assumes  $\square \notin A$   
 shows  $X = A \text{@@} X \cup B \longleftrightarrow X = \text{star } A \text{@@} B$

**proof**

assume *eq*:  $X = A \text{@@} X \cup B$

{ **fix**  $w$  **assume**  $w : X$

**let**  $?n = \text{size } w$

**from**  $\langle \square \notin A \rangle$  **have**  $\text{ALL } u : A. \text{length } u \geq 1$

**by** *(metis Suc-eq-plus1 add-leD2 le-0-eq length-0-conv not-less-eq-eq)*

**hence**  $\text{ALL } u : A \hat{\hat{}} (?n+1). \text{length } u \geq ?n+1$

**by** *(metis length-lang-pow-lb nat-mult-1)*

**hence**  $\text{ALL } u : A \hat{\hat{}} (?n+1) \text{@@} X. \text{length } u \geq ?n+1$

**by** *(auto simp only: conc-def length-append)*

**hence**  $w \notin A \hat{\hat{}} (?n+1) \text{@@} X$  **by** *auto*

**hence**  $w : \text{star } A \text{@@} B$  **using**  $\langle w : X \rangle$  **using** *arden-helper[OF eq, where  $n=?n$ ]*

**by** *(auto simp add: star-def conc-UNION-distrib)*

} **moreover**

{ **fix**  $w$  **assume**  $w : \text{star } A \text{@@} B$

**hence**  $\text{EX } n. w : A \hat{\hat{}} n \text{@@} B$  **by** *(auto simp: conc-def star-def)*

**hence**  $w : X$  **using** *arden-helper[OF eq]* **by** *blast*

} **ultimately show**  $X = \text{star } A \text{@@} B$  **by** *blast*

**next**

assume *eq*:  $X = \text{star } A \text{@@} B$

**have**  $\text{star } A = A \text{@@} \text{star } A \cup \{\square\}$

**by** *(rule star-unfold-left)*

**then have**  $\text{star } A \text{@@} B = (A \text{@@} \text{star } A \cup \{\square\}) \text{@@} B$

**by** *metis*

**also have**  $\dots = (A \text{@@} \text{star } A) \text{@@} B \cup B$

**unfolding** *conc-Un-distrib* **by** *simp*

**also have**  $\dots = A \text{@@} (\text{star } A \text{@@} B) \cup B$

**by** *(simp only: conc-assoc)*

**finally show**  $X = A \text{@@} X \cup B$

**using** *eq* **by** *blast*

qed

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

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

shows  $X = X \text{@@} (A \hat{\hat{}} \text{Suc } n) \cup (\bigcup m \leq n. B \text{@@} (A \hat{\hat{}} m))$

**proof** *(induct n)*

**case**  $0$

**show**  $X = X \text{@@} (A \hat{\hat{}} \text{Suc } 0) \cup (\bigcup m \leq 0. B \text{@@} (A \hat{\hat{}} m))$

using *eq* by *simp*  
 next  
 case (*Suc n*)  
 have *ih*:  $X = X \text{@@} (A \text{^^} \text{Suc } n) \cup (\bigcup m \leq n. B \text{@@} (A \text{^^} m))$  by *fact*  
 also have  $\dots = (X \text{@@} A \cup B) \text{@@} (A \text{^^} \text{Suc } n) \cup (\bigcup m \leq n. B \text{@@} (A \text{^^} m))$   
 using *eq* by *simp*  
 also have  $\dots = X \text{@@} (A \text{^^} \text{Suc } (\text{Suc } n)) \cup (B \text{@@} (A \text{^^} \text{Suc } n)) \cup (\bigcup m \leq n. B \text{@@} (A \text{^^} m))$   
 by (*simp add: conc-Un-distrib conc-assoc*)  
 also have  $\dots = X \text{@@} (A \text{^^} \text{Suc } (\text{Suc } n)) \cup (\bigcup m \leq \text{Suc } n. B \text{@@} (A \text{^^} m))$   
 by (*auto simp add: atMost-Suc*)  
 finally show  $X = X \text{@@} (A \text{^^} \text{Suc } (\text{Suc } n)) \cup (\bigcup m \leq \text{Suc } n. B \text{@@} (A \text{^^} m))$   
 .  
 qed

**theorem** *reversed-Arden*:

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

shows  $X = X \text{@@} A \cup B \longleftrightarrow X = B \text{@@} \text{star } A$

**proof**

assume *eq*:  $X = X \text{@@} A \cup B$

{ **fix** *w* **assume**  $w : X$

  let  $?n = \text{size } w$

**from**  $\langle \square \notin A \rangle$  **have**  $\text{ALL } u : A. \text{length } u \geq 1$

    by (*metis Suc-eq-plus1 add-leD2 le-0-eq length-0-conv not-less-eq-eq*)

**hence**  $\text{ALL } u : A \text{^^} (?n+1). \text{length } u \geq ?n+1$

    by (*metis length-lang-pow-lb nat-mult-1*)

**hence**  $\text{ALL } u : X \text{@@} A \text{^^} (?n+1). \text{length } u \geq ?n+1$

    by (*auto simp only: conc-def length-append*)

**hence**  $w \notin X \text{@@} A \text{^^} (?n+1)$  by *auto*

**hence**  $w : B \text{@@} \text{star } A$  **using**  $\langle w : X \rangle$  **using** *reversed-arden-helper*[*OF eq*,

*where*  $n=?n$ ]

  by (*auto simp add: star-def conc-UNION-distrib*)

} **moreover**

{ **fix** *w* **assume**  $w : B \text{@@} \text{star } A$

**hence**  $\text{EX } n. w : B \text{@@} A \text{^^} n$  by (*auto simp: conc-def star-def*)

**hence**  $w : X$  **using** *reversed-arden-helper*[*OF eq*] **by** *blast*

} **ultimately show**  $X = B \text{@@} \text{star } A$  **by** *blast*

**next**

assume *eq*:  $X = B \text{@@} \text{star } A$

have  $\text{star } A = \{\square\} \cup \text{star } A \text{@@} A$

**unfolding** *conc-star-comm*[*symmetric*]

  by (*metis Un-commute star-unfold-left*)

**then have**  $B \text{@@} \text{star } A = B \text{@@} (\{\square\} \cup \text{star } A \text{@@} A)$

  by *metis*

**also have**  $\dots = B \cup B \text{@@} (\text{star } A \text{@@} A)$

**unfolding** *conc-Un-distrib* **by** *simp*

**also have**  $\dots = B \cup (B \text{@@} \text{star } A) \text{@@} A$

  by (*simp only: conc-assoc*)

**finally show**  $X = X \text{@@} A \cup B$

using *eq* by *blast*  
 qed

## 1.7 Lists of Fixed Length

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

**lemma** *tl-listsN*:  $A \subseteq listsN\ (n + 1)\ S \implies tl\ 'A \subseteq listsN\ n\ S$

**proof** (*intro image-subsetI*)  
 fix *xs* assume  $A \subseteq listsN\ (n + 1)\ S$   $xs \in A$   
 thus  $tl\ xs \in listsN\ n\ S$  by (*induct xs*) *auto*  
 qed

**lemma** *map-tl-listsN*:  $A \subseteq lists\ (listsN\ (n + 1)\ S) \implies map\ tl\ 'A \subseteq lists\ (listsN\ n\ S)$

**proof** (*intro image-subsetI*)  
 fix *xss* assume  $A \subseteq lists\ (listsN\ (n + 1)\ S)$   $xss \in A$   
 hence  $set\ xss \subseteq listsN\ (n + 1)\ S$  by *auto*  
 hence  $\forall xs \in set\ xss.\ tl\ xs \in listsN\ n\ S$  using *tl-listsN*[*of set xss S n*] by *auto*  
 thus  $map\ tl\ xss \in lists\ (listsN\ n\ S)$  by (*induct xss*) *auto*  
 qed

## 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* OPERATION *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**  $P xs$

**using** *assms*

**proof** (*induct xs*)

**case** (*Cons x xs*) **thus** ?*case* **by** (*cases xs*) *auto*

**qed** *simp*

## 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*  $r \neq \{\}$

**by** (*induct r*) *auto*

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

*finite* (*toplevel-summands*  $r$ )

**by** (*induct r*) *auto*

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

$\llcorner\!\!\!\!\!\! \gg$  *Zero* = *Zero*

|  $\llcorner\!\!\!\!\!\! \gg$  *Full* = *Full*

|  $\llcorner\!\!\!\!\!\! \gg$  *One* = *One*

|  $\llcorner\!\!\!\!\!\! \gg$  *Atom a* = *Atom a*

|  $\llcorner\!\!\!\!\!\! \gg$  *Plus r s* = *flatten PLUS (toplevel-summands (Plus  $\llcorner\!\!\!\!\!\! \gg$   $\llcorner\!\!\!\!\!\! \gg$ ))*

|  $\llcorner\!\!\!\!\!\! \gg$  *Times r s* = *Times  $\llcorner\!\!\!\!\!\! \gg$   $\llcorner\!\!\!\!\!\! \gg$*

|  $\llcorner\!\!\!\!\!\! \gg$  *Star r* = *Star  $\llcorner\!\!\!\!\!\! \gg$*

|  $\llcorner\!\!\!\!\!\! \gg$  *Not r* = *Not  $\llcorner\!\!\!\!\!\! \gg$*

|  $\llcorner\!\!\!\!\!\! \gg$  *Inter r s* = *Inter  $\llcorner\!\!\!\!\!\! \gg$   $\llcorner\!\!\!\!\!\! \gg$*

|  $\llcorner\!\!\!\!\!\! \gg$  *Pr r* = *Pr  $\llcorner\!\!\!\!\!\! \gg$*

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

$Plus r s \in$  *toplevel-summands*  $t \implies$  *False*

**by** (*induct t*) *auto*

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

$(\forall r s. x \neq Plus r s) \implies$  *toplevel-summands*  $x = \{x\}$

**by** (*induct x*) *auto*

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

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

**by** (*induct xs rule: list-singleton-induct*) *auto*

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

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

**using** *toplevel-summands-PLUS-strong*[*of sorted-list-of-set X*]

**unfolding** *list-all-iff* **by** *fastforce*

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

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

**by** (*induct r*) *auto*

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

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

**by** (*intro toplevel-summands-flatten*) (*auto dest!: ACI-norm-Plus intro: Plus-toplevel-summands*)

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

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

$\text{ACI-norm } \text{'toplevel-summands } r \cup \text{ACI-norm } \text{'toplevel-summands } s$

**by** (*intro toplevel-summands-flatten*) (*auto dest!: ACI-norm-Plus[OF sym] intro: Plus-toplevel-summands*)

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

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

**by** (*induct r*) (*auto simp: toplevel-summands-flatten-ACI-norm-image-Union*)

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

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

**by** (*induct r*) (*auto simp: image-Un toplevel-summands-flatten-ACI-norm-image*)

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

$\langle \langle r \rangle \rangle = \langle r \rangle$

**proof** (*induct r*)

**case** (*Plus r s*)

**have**  $\langle \langle \text{Plus } r s \rangle \rangle = \langle \text{flatten } \text{PLUS } (\text{toplevel-summands } \langle r \rangle \cup \text{toplevel-summands } \langle s \rangle) \rangle$

(*is - = <flatten PLUS ?U>*) **by** *simp*

**also have**  $\dots = \text{flatten } \text{PLUS } (\text{ACI-norm } \text{'toplevel-summands } (\text{flatten } \text{PLUS } \text{'?U}))$

**by** (*simp only: ACI-norm-flatten*)

**also have**  $\text{toplevel-summands } (\text{flatten } \text{PLUS } \text{'?U}) = \text{'?U}$

**by** (*intro toplevel-summands-flatten*) (*auto intro: Plus-toplevel-summands*)

**also have**  $\text{flatten } \text{PLUS } (\text{ACI-norm } \text{'?U}) = \text{flatten } \text{PLUS } (\text{toplevel-summands } \langle r \rangle \cup \text{toplevel-summands } \langle s \rangle)$

**by** (*simp only: image-Un toplevel-summands-ACI-norm[symmetric] Plus*)



**finally show** *?case by simp*  
**qed** *auto*

**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*)  
**by** (*induct* *r* *s* *rule*: *ACI-nPlus.induct*) *auto*

**lemma** *toplevel-summands-ACI-norm-alt*:  
*toplevel-summands* (*ACI-norm-alt* *r*) = *ACI-norm-alt* ' *toplevel-summands* *r*  
**by** (*induct* *r*) (*auto simp*: *toplevel-summands-ACI-nPlus*)

**lemma** *ACI-norm-alt-Plus*:  
*ACI-norm-alt* *r* = *Plus* *s* *t*  $\Longrightarrow$   $\exists$  *s* *t*. *r* = *Plus* *s* *t*  
**by** (*induct* *r*) *auto*

**lemma** *toplevel-summands-flatten-ACI-norm-alt-image*:  
*toplevel-summands* (*flatten* *PLUS* (*ACI-norm-alt* ' *toplevel-summands* *r*)) =  
*ACI-norm-alt* ' *toplevel-summands* *r*  
**by** (*intro* *toplevel-summands-flatten*) (*auto dest*!: *ACI-norm-alt-Plus* *intro*: *Plus-toplevel-summands*)

**lemma** *ACI-norm-ACI-norm-alt*:  $\langle\langle$ *ACI-norm-alt* *r* $\rangle\rangle$  =  $\langle\langle$ *r* $\rangle\rangle$

**proof** (*induction* *r*)  
  **case** (*Plus* *r* *s*) **show** *?case*  
  **using** *ACI-norm-flatten* [*of* *ACI-norm-alt* (*Plus* *r* *s*)] *ACI-norm-flatten* [*of* *Plus*  
  *r* *s*]

by (auto simp: image-Un toplevel-summands-ACI-nPlus)  
 (metis Plus.IH toplevel-summands-ACI-norm)

qed auto

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

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

**proof** (induct xs rule: list-singleton-induct)

case (single y)

thus ?case

by (cases x y rule: linorder-cases) (induct x y rule: ACI-nPlus.induct, auto)+

next

case (cons y1 y2 ys) thus ?case by (cases x) (auto)

qed simp

**lemma** ACI-nPlus-PLUS:

$\llbracket xs1 \neq []; xs2 \neq []; \forall x \in \text{set } (xs1 @ xs2). \neg(\exists r s. x = \text{Plus } r s); \text{sorted } xs2; \text{distinct } xs2 \rrbracket \implies$

$\text{ACI-nPlus } (\text{PLUS } xs1) (\text{PLUS } xs2) = \text{flatten } \text{PLUS } (\text{set } (xs1 @ xs2))$

**proof** (induct xs1 arbitrary: xs2 rule: list-singleton-induct)

case (single x1) thus ?case

apply (auto intro!: trans[OF ACI-nPlus-singleton-PLUS] simp: insert-absorb  
 simp del: sorted-list-of-set-insert)

apply (metis finite-set finite-sorted-distinct-unique sorted-list-of-set)

apply (metis remdups-id-iff-distinct sorted-list-of-set-sort-remdups sorted-sort-id)

done

next

case (cons x11 x12 xs1) thus ?case

apply (simp del: sorted-list-of-set-insert)

apply (rule trans[OF ACI-nPlus-singleton-PLUS])

apply (auto simp del: sorted-list-of-set-insert simp add: insert-commute[of  
 x11])

apply (auto simp only: Un-insert-left[of x11, symmetric] insert-absorb) []

apply (auto simp only: Un-insert-right[of - x11, symmetric] insert-absorb) []

apply (auto simp add: insert-commute[of x12])

done

qed simp

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

$\llbracket X1 \neq \{\}; X2 \neq \{\}; \text{finite } X1; \text{finite } X2; \forall x \in X1 \cup X2. \neg(\exists r s. x = \text{Plus } r s) \rrbracket \implies$

$\text{ACI-nPlus } (\text{flatten } \text{PLUS } X1) (\text{flatten } \text{PLUS } X2) = \text{flatten } \text{PLUS } (X1 \cup X2)$

by (rule trans[OF ACI-nPlus-PLUS]) auto

**lemma** ACI-nPlus-ACI-norm[simp]:  $\text{ACI-nPlus } \llbracket r \rrbracket \llbracket s \rrbracket = \llbracket \text{Plus } r s \rrbracket$

using ACI-norm-flatten [of r] ACI-norm-flatten [of s] ACI-norm-flatten [of Plus  
 r s]

apply (auto intro!: trans [OF ACI-nPlus-flatten-PLUS])

apply (auto simp: image-Un Un-assoc intro!: trans [OF ACI-nPlus-flatten-PLUS])

```

apply (metis ACI-norm-Plus Plus-toplevel-summands)+
done

```

```

lemma ACI-norm-alt:
  ACI-norm-alt r = <<r>>
by (induct r) auto

```

```

declare ACI-norm-alt[symmetric, code]

```

## 2.3 Finality

```

primrec final :: 'a rexpr  $\Rightarrow$  bool
where
  final Zero = False
| final Full = True
| final One = True
| final (Atom -) = False
| final (Plus r s) = (final r  $\vee$  final s)
| final (Times r s) = (final r  $\wedge$  final s)
| final (Star -) = True
| final (Not r) = ( $\sim$  final r)
| final (Inter r1 r2) = (final r1  $\wedge$  final r2)
| final (Pr r) = final r

```

```

lemma toplevel-summands-final:
  final s = ( $\exists r \in$  toplevel-summands s. final r)
by (induct s) auto

```

```

lemma final-PLUS:
  final (PLUS xs) = ( $\exists r \in$  set xs. final r)
by (induct xs rule: list-singleton-induct) auto

```

```

theorem ACI-norm-final[simp]:
  final <<r>> = final r
proof (induct r)
  case (Plus r1 r2) thus ?case using toplevel-summands-final by (auto simp:
final-PLUS)
qed auto

```

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

```

locale alphabet =
fixes  $\Sigma$  :: nat  $\Rightarrow$  'a set ( $\Sigma$  -)
and wf-atom :: nat  $\Rightarrow$  'b :: linorder  $\Rightarrow$  bool
begin

```

```

primrec wf :: nat  $\Rightarrow$  'b rexpr  $\Rightarrow$  bool
where
  wf n Zero = True |
  wf n Full = True |

```

$wf\ n\ One = True \mid$   
 $wf\ n\ (Atom\ a) = (wf\text{-}atom\ n\ a) \mid$   
 $wf\ n\ (Plus\ r\ s) = (wf\ n\ r \wedge wf\ n\ s) \mid$   
 $wf\ n\ (Times\ r\ s) = (wf\ n\ r \wedge wf\ n\ s) \mid$   
 $wf\ n\ (Star\ r) = wf\ n\ r \mid$   
 $wf\ n\ (Not\ r) = wf\ n\ r \mid$   
 $wf\ n\ (Inter\ r\ s) = (wf\ n\ r \wedge wf\ n\ s) \mid$   
 $wf\ n\ (Pr\ r) = wf\ (n + 1)\ r$

**primrec** *wf-word* **where**

$wf\text{-}word\ n\ [] = True$   
 $\mid wf\text{-}word\ n\ (w \# ws) = ((w \in \Sigma\ n) \wedge wf\text{-}word\ n\ ws)$

**lemma** *wf-word-snoc*[*simp*]:  $wf\text{-}word\ n\ (ws\ @\ [w]) = ((w \in \Sigma\ n) \wedge wf\text{-}word\ n\ ws)$   
**by** (*induct ws*) *auto*

**lemma** *wf-word-append*[*simp*]:  $wf\text{-}word\ n\ (ws\ @\ vs) = (wf\text{-}word\ n\ ws \wedge wf\text{-}word\ n\ vs)$   
**by** (*induct ws arbitrary: vs*) *auto*

**lemma** *wf-word*:  $wf\text{-}word\ n\ w = (w \in lists\ (\Sigma\ n))$   
**by** (*induct w*) *auto*

**lemma** *toplevel-summands-wf*:  
 $wf\ n\ s = (\forall r \in toplevel\text{-}summands\ s. wf\ n\ r)$   
**by** (*induct s*) *auto*

**lemma** *wf-PLUS*[*simp*]:  
 $wf\ n\ (PLUS\ xs) = (\forall r \in set\ xs. wf\ n\ r)$   
**by** (*induct xs rule: list-singleton-induct*) *auto*

**lemma** *wf-TIMES*[*simp*]:  
 $wf\ n\ (TIMES\ xs) = (\forall r \in set\ xs. wf\ n\ r)$   
**by** (*induct xs rule: list-singleton-induct*) *auto*

**lemma** *wf-flatten-PLUS*[*simp*]:  
 $finite\ X \implies wf\ n\ (flatten\ PLUS\ X) = (\forall r \in X. wf\ n\ r)$   
**using** *wf-PLUS*[*of n sorted-list-of-set X*] **by** *fastforce*

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

$wf\ n\ \langle\langle r \rangle\rangle = wf\ n\ r$

**proof** (*induct r arbitrary: n*)

**case** (*Plus r1 r2*) **thus** *?case*

**using** *toplevel-summands-wf*[*of n <<r1>>*] *toplevel-summands-wf*[*of n <<r2>>*] **by**  
*auto*

**qed** *auto*

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

$wf\ n\ (INTERSECT\ xs) = (\forall r \in set\ xs. wf\ n\ r)$

**by** (*induct xs rule: list-singleton-induct*) *auto*

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

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

**using** *wf-INTERSECT*[*of n sorted-list-of-set X*] **by** *fastforce*

**end**

## 2.5 Language

**locale** *project* =

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

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

**and** *lookup*  $:: 'b \Rightarrow 'a \Rightarrow \text{bool}$

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

**begin**

**primrec** *lang*  $:: \text{nat} \Rightarrow 'b \text{ rexp} \Rightarrow 'a \text{ lang}$  **where**

*lang n Zero* =  $\{\}$  |

*lang n Full* = *lists* ( $\Sigma \ n$ ) |

*lang n One* =  $\{\ \ \}$  |

*lang n (Atom b)* =  $\{[x] \mid x. \text{lookup } b \ x \wedge x \in \Sigma \ n\}$  |

*lang n (Plus r s)* =  $(\text{lang } n \ r) \cup (\text{lang } n \ s)$  |

*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)* =  $(\text{lang } n \ r \cap \text{lang } n \ s)$  |

*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*)

**by** (*induct ws arbitrary: n*) (*auto intro: project*)

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

**by** (*induct r arbitrary: n*) (*auto elim: rev-subsetD[OF - conc-mono] star-induct intro: iffD2[OF wf-word]*)

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

**proof** (*induct r arbitrary: n*)

**case** *Pr* **thus** *?case* **by** (*fastforce intro!: project*)

**qed** (*auto simp: conc-subset-lists star-subset-lists*)

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

$r \in \text{toplevel-summands } s \implies \text{lang } n \ r \subseteq \text{lang } n \ s$

**by** (*induct s*) *auto*

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

$\text{lang } n \ s = \bigcup_{r \in \text{toplevel-summands } s. \text{lang } n \ r$

**by** (*induct s*) *auto*

**lemma** *toplevel-summands-in-lang*:  
 $w \in \text{lang } n \ s = (\exists r \in \text{toplevel-summands } s. w \in \text{lang } n \ r)$   
**by** (*induct s*) *auto*

**lemma** *lang-PLUS[simp]*:  
 $\text{lang } n \ (\text{PLUS } xs) = (\bigcup r \in \text{set } xs. \text{lang } n \ r)$   
**by** (*induct xs rule: list-singleton-induct*) *auto*

**lemma** *lang-TIMES[simp]*:  
 $\text{lang } n \ (\text{TIMES } xs) = \text{foldr } (@@) \ (\text{map } (\text{lang } n) \ xs) \ \{\}\}$   
**by** (*induct xs rule: list-singleton-induct*) *auto*

**lemma** *lang-flatten-PLUS*:  
 $\text{finite } X \implies \text{lang } n \ (\text{flatten } \text{PLUS } X) = (\bigcup r \in X. \text{lang } n \ r)$   
**using** *lang-PLUS[of n sorted-list-of-set X]* **by** *fastforce*

**theorem** *ACI-norm-lang[simp]*:  
 $\text{lang } n \ \langle\langle r \rangle\rangle = \text{lang } n \ r$   
**proof** (*induct r arbitrary: n*)  
**case** (*Plus r1 r2*)  
**moreover**  
**from** *Plus[symmetric]* **have**  $\text{lang } n \ (\text{Plus } r1 \ r2) \subseteq \text{lang } n \ \langle\langle \text{Plus } r1 \ r2 \rangle\rangle$   
**using** *toplevel-summands-in-lang[of - n <<r1>>]* *toplevel-summands-in-lang[of - n <<r2>>]*  
**by** *auto*  
**ultimately show** *?case by (fastforce dest!: toplevel-summands-lang)*  
**qed** *auto*

**lemma** *lang-final*:  $\text{final } r = (\ [] \in \text{lang } n \ r)$   
**using** *concI[of [] - []]* **by** (*induct r arbitrary: n*) *auto*

**lemma** *in-lang-INTERSECT*:  
 $\text{wf-word } n \ w \implies w \in \text{lang } n \ (\text{INTERSECT } xs) = (\forall r \in \text{set } xs. w \in \text{lang } n \ r)$   
**by** (*induct xs rule: list-singleton-induct*) (*auto simp: wf-word*)

**lemma** *lang-INTERSECT*:  
 $\text{lang } n \ (\text{INTERSECT } xs) = (\text{if } xs = [] \text{ then lists } (\Sigma \ n) \text{ else } \bigcap r \in \text{set } xs. \text{lang } n \ r)$   
**by** (*induct xs rule: list-singleton-induct*) *auto*

**lemma** *lang-flatten-INTERSECT[simp]*:  
**assumes** *finite X X ≠ {}*  $\forall r \in X. \text{wf } n \ r$   
**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**  
**assume** *?L*  
**thus** *?R using in-lang-INTERSECT[OF bspec[OF wf-lang-wf-word[OF iffD2[OF*

```

wf-flatten-INTERSECT]]],
  OF assms(1,3) ( ?L, of sorted-list-of-set X] assms(1)
  by auto
next
  assume ?R
  with assms show ?L by (intro iffD2[OF in-lang-INTERSECT]) (auto dest:
wf-lang-wf-word)
qed

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:  $\bigwedge a. a \in \Sigma n \implies b \in \text{set } (embed \ a) = (b \in \Sigma (Suc \ n) \wedge project \ b = a)$ 
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 ( $\lambda a'. lderiv \ a' \ r$ ) (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* (*lderiv as r*) = ( $\bigcup_{s \in \text{toplevel-summands } r} \text{toplevel-summands } (lderiv \text{ as } s)$ )

**by** (*induct r*) (*auto simp: Let-def*)

**lemma** *lderiv-Zero*[*simp*]: *lderiv xs Zero = Zero*

**by** (*induct xs*) *auto*

**lemma** *lderiv-Full*[*simp*]: *lderiv xs Full = Full*

**by** (*induct xs*) *auto*

**lemma** *lderiv-One*: *lderiv xs One*  $\in$  {*Zero, One*}

**by** (*induct xs*) *auto*

**lemma** *lderiv-Atom*: *lderiv xs (Atom as)*  $\in$  {*Zero, One, Atom as*}

**proof** (*induct xs*)

**case** (*Cons x xs*) **thus** ?*case* **by** (*auto intro: insertE[OF lderiv-One]*)

**qed** *simp*

**lemma** *lderiv-Plus*: *lderiv xs (Plus r s) = Plus (lderiv xs r) (lderiv xs s)*

**by** (*induct xs arbitrary: r s*) *auto*

**lemma** *lderiv-PLUS*: *lderiv xs (PLUS ys) = PLUS (map (lderiv xs) ys)*

**by** (*induct ys rule: list-singleton-induct*) (*auto simp: lderiv-Plus*)

**lemma** *toplevel-summands-lderiv-Times*: *toplevel-summands (lderiv xs (Times r s))*  $\subseteq$

{*Times (lderiv xs r) s*}  $\cup$

{*r'.  $\exists ys zs. r' \in \text{toplevel-summands } (lderiv ys s) \wedge ys \neq [] \wedge zs @ ys = xs$* }

**proof** (*induct xs arbitrary: r s*)

**case** (*Cons x xs*)

**thus** ?*case* **by** (*auto simp: Let-def lderiv-Plus*) (*fastforce intro: exI[of - x#xs]*)+

**qed** *simp*

**lemma** *toplevel-summands-lderiv-Star-nonempty*:

*xs*  $\neq [] \implies \text{toplevel-summands } (lderiv xs (Star r)) \subseteq$

{*Times (lderiv ys r) (Star r) | ys.  $\exists zs. ys \neq [] \wedge zs @ ys = xs$* }

**proof** (*induct xs rule: length-induct*)

**case** (*1 xs*)

**then obtain** *y ys* **where** *xs = y # ys* **by** (*cases xs*) *auto*

**thus** ?*case* **using** *spec[OF 1(1)]*

**by** (*auto dest!: subsetD[OF toplevel-summands-lderiv-Times] intro: exI[of - y#ys]*)

(*auto elim!: impE dest!: meta-spec subsetD*)

**qed**

**lemma** *toplevel-summands-lderiv-Star*:

*toplevel-summands (lderiv xs (Star r))*  $\subseteq$

{*Star r*}  $\cup$  {*Times (lderiv ys r) (Star r) | ys.  $\exists zs. ys \neq [] \wedge zs @ ys = xs$* }

**by** (*cases xs = []*) (*auto dest!: toplevel-summands-lderiv-Star-nonempty*)



**lemma** *ex-lderivs-Pr*:  $\exists s. \text{lderivs ass } (Pr\ r) = Pr\ s$   
**by** (*induct ass arbitrary*: *r*) *auto*

**lemma** *toplevel-summands-PLUS*:  
 $xs \neq [] \implies \text{toplevel-summands } (PLUS\ (\text{map } f\ xs)) = (\bigcup r \in \text{set } xs. \text{toplevel-summands } (f\ r))$   
**by** (*induct xs rule*: *list-singleton-induct*) *simp-all*

**lemma** *lderiv-toplevel-summands-Zero*:  
 $\llbracket \text{lderivs } xs\ (Pr\ r) = Pr\ s; \text{toplevel-summands } r = \{\text{Zero}\} \rrbracket \implies \text{toplevel-summands } s = \{\text{Zero}\}$   
**proof** (*induct xs arbitrary*: *r s*)  
**case** (*Cons y ys*)  
**from** *Cons.prem1* **have**  $\text{toplevel-summands } (PLUS\ (\text{map } (\lambda a. \text{lderiv } a\ r)\ (\text{embed } y))) = \{\text{Zero}\}$   
**proof** (*cases embed y = []*)  
**case** *False*  
**show** *?thesis* **using** *Cons.prem2* **unfolding** *toplevel-summands-PLUS[OF False]*  
**by** (*subst toplevel-summands-lderiv*) (*simp add: False*)  
**qed** *simp*  
**with** *Cons* **show** *?case* **by** *simp*  
**qed** *simp*

**lemma** *toplevel-summands-lderivs-Pr*:  
 $\llbracket xs \neq []; \text{lderivs } xs\ (Pr\ r) = Pr\ s \rrbracket \implies \text{toplevel-summands } s \subseteq \{\text{Zero}\} \vee \text{toplevel-summands } s \subseteq (\bigcup xs. \text{toplevel-summands } (\text{lderivs } xs\ r))$   
**proof** (*induct xs arbitrary*: *r s*)  
**case** (*Cons y ys*) **note**  $*$  = *this*  
**show** *?case*  
**proof** (*cases embed y = []*)  
**case** *True* **with** *Cons* **show** *?thesis* **by** (*cases ys = []*) (*auto dest: lderiv-toplevel-summands-Zero*)  
**next**  
**case** *False*  
**show** *?thesis*  
**proof** (*cases ys*)  
**case** *Nil* **with**  $*$  **show** *?thesis*  
**by** (*auto simp: toplevel-summands-PLUS[OF False]*) (*metis lderiv.simps*)  
**next**  
**case** (*Cons z zs*)  
**have**  $\text{toplevel-summands } s \subseteq \{\text{Zero}\} \vee \text{toplevel-summands } s \subseteq (\bigcup xs. \text{toplevel-summands } (\text{lderivs } xs\ (PLUS\ (\text{map } (\lambda a. \text{lderiv } a\ r)\ (\text{embed } y))))))$  (*is -  $\vee$  ?B*)  
**by** (*rule \*(1)*) (*auto simp: Cons \*(3)[symmetric]*)  
**thus** *?thesis*  
**proof**  
**assume** *?B*

**also have**  $\dots \subseteq (\bigcup xs. \text{toplevel-summands } (\text{lderivs } xs \ r))$   
**by** (*auto simp: lderivs-PLUS topLevel-summands-PLUS[OF False]*) (*metis lderivs.simps(2)*)  
**finally show** *?thesis ..*  
**qed blast**  
**qed**  
**qed**  
**qed simp**

**lemma lderivs-Pr:**

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

**proof** (*rule subsetI*)

**fix**  $s$  **assume**  $s \in ?L$

**then obtain**  $xs$  **where**  $s = \text{lderivs } xs \ (Pr \ r)$  **by** *blast*

**moreover obtain**  $t$  **where**  $\text{lderivs } xs \ (Pr \ r) = Pr \ t$  **using** *ex-lderivs-Pr* **by** *blast*

**ultimately show**  $s \in ?R$

**by** (*cases*  $xs = []$ ) (*auto dest!: topLevel-summands-lderivs-Pr elim!: allE[of - []]*)

**qed**

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

**by** (*subst ACI-norm-flatten*) (*auto dest: subset-singletonD*)

**lemma ACI-norm-lderivs-Pr:**

$ACI\text{-norm } \{ \text{lderivs } xs \ (Pr \ r) \mid xs. \text{True} \} \subseteq$   
 $\{Pr \ \text{Zero}\} \cup \{Pr \ \ll s \gg \mid s. \text{toplevel-summands } s \subseteq (\bigcup xs. \text{toplevel-summands } \ll \text{lderivs } xs \ r \gg)\}$

**proof** (*intro subset-trans[OF image-mono[OF lderivs-Pr]] subsetI,*

*elim imageE CollectE exE conjE disjE*)

**fix**  $x \ x' \ s :: 'b \ \text{rexp}$

**assume**  $*$ :  $x = \ll x' \gg \ x' = Pr \ s$  **and**  $\text{toplevel-summands } s \subseteq \{\text{Zero}\}$

**hence**  $\ll Pr \ s \gg = Pr \ \text{Zero}$  **using** *ACI-norm-toplevel-summands-Zero* **by** *simp*

**thus**  $x \in \{Pr \ \text{Zero}\} \cup$

$\{Pr \ \ll s \gg \mid s. \text{toplevel-summands } s \subseteq (\bigcup xs. \text{toplevel-summands } \ll \text{lderivs } xs \ r \gg)\}$

**unfolding**  $*$  **by** *blast*

**next**

**fix**  $x \ x' \ s :: 'b \ \text{rexp}$

**assume**  $*$ :  $x = \ll x' \gg \ x' = Pr \ s$  **and**  $\text{toplevel-summands } s \subseteq (\bigcup xs. \text{toplevel-summands } (\text{lderivs } xs \ r))$

**hence**  $\text{toplevel-summands } \ll s \gg \subseteq (\bigcup xs. \text{toplevel-summands } \ll \text{lderivs } xs \ r \gg)$

**by** (*fastforce simp: topLevel-summands-ACI-norm*)

**moreover have**  $x = Pr \ \ll \ll s \gg \gg$  **unfolding**  $*$  *ACI-norm-idem ACI-norm.simps(10)*

**..**

**ultimately show**  $x \in \{Pr \ \text{Zero}\} \cup$

$\{Pr \ \ll s \gg \mid s. \text{toplevel-summands } s \subseteq (\bigcup xs. \text{toplevel-summands } \ll \text{lderivs } xs \ r \gg)\}$

**by** *blast*

qed

**lemma** *finite-ACI-norm-toplevel-summands*:  $\text{finite } B \implies \text{finite } \{f \ll s \gg \mid s. \text{toplevel-summands } s \subseteq B\}$   
**apply** (*elim finite-surj* [*OF iffD2* [*OF finite-Pow-iff*], *of - - f o flatten PLUS o image ACI-norm*])  
**apply** (*subst ACI-norm-flatten*)  
**apply** *auto*  
**done**

**lemma** *lderivs-Not*:  $\text{lderivs } xs \ (\text{Not } r) = \text{Not } (\text{lderivs } xs \ r)$   
**by** (*induct xs arbitrary: r*) *auto*

**lemma** *lderivs-Inter*:  $\text{lderivs } xs \ (\text{Inter } r \ s) = \text{Inter } (\text{lderivs } xs \ r) \ (\text{lderivs } xs \ s)$   
**by** (*induct xs arbitrary: r s*) *auto*

**theorem** *finite-lderivs*:  $\text{finite } \{\ll \text{lderivs } xs \ r \gg \mid xs . \text{True}\}$   
**proof** (*induct r*)  
**case** *Zero* **show** *?case by simp*  
**next**  
**case** *Full* **show** *?case by simp*  
**next**  
**case** *One* **show** *?case*  
**by** (*rule finite-surj*[*of* {*Zero, One*}] (*blast intro: insertE*[*OF lderivs-One*]))+  
**next**  
**case** (*Atom as*) **show** *?case*  
**by** (*rule finite-surj*[*of* {*Zero, One, Atom as*}] (*blast intro: insertE*[*OF lderivs-Atom*]))+  
**next**  
**case** (*Plus r s*)  
**show** *?case by* (*auto simp: lderivs-Plus intro!: finite-surj*[*OF finite-cartesian-product*[*OF Plus*]])  
**next**  
**case** (*Times r s*)  
**hence** *finite* ( $\bigcup (\text{toplevel-summands } ' \{\ll \text{lderivs } xs \ s \gg \mid xs . \text{True}\})$ ) **by** *auto*  
**moreover** **have**  $\{\ll r' \gg \mid r'. \exists ys. r' \in \text{toplevel-summands } (\text{lderivs } ys \ s)\} = \{r'. \exists ys. r' \in \text{toplevel-summands } \ll \text{lderivs } ys \ s \gg\}$   
**unfolding** *toplevel-summands-ACI-norm* **by** *auto*  
**ultimately** **have** *fin*:  $\text{finite } \{\ll r' \gg \mid r'. \exists ys. r' \in \text{toplevel-summands } (\text{lderivs } ys \ s)\}$   
**by** (*fastforce intro: finite-subset*[*of* -  $\bigcup (\text{toplevel-summands } ' \{\ll \text{lderivs } xs \ s \gg \mid xs . \text{True}\})$ ])  
**let** *?X* =  $\lambda xs. \{\text{Times } (\text{lderivs } ys \ r) \ s \mid ys. \text{True}\} \cup \{r'. r' \in (\bigcup ys. \text{toplevel-summands } (\text{lderivs } ys \ s))\}$   
**show** *?case*  
**proof** (*simp only: ACI-norm-flatten*,  
*rule finite-surj*[*of* {*X. \exists xs. X \subseteq ACI-norm ' ?X xs* } - *flatten PLUS*])  
**show** *finite* {*X. \exists xs. X \subseteq ACI-norm ' ?X xs*}  
**using** *fin* **by** (*fastforce simp: image-Un elim: finite-subset*[*rotated*] *intro: finite-surj*[*OF Times*(1), *of* -  $\lambda r. \text{Times } r \ll s \gg$ ])

```

qed (fastforce dest!: subsetD[OF toplevel-summands-lderivs-Times] intro!: im-
ageI)
next
  case (Star r)
  let ?f =  $\lambda f r'. \text{Times } r' (Star (f r))$ 
  let ?X =  $\{Star\ r\} \cup ?f\ id \ \{r'. r' \in \{lderivs\ ys\ r | ys. True\}\}$ 
  show ?case
  proof (simp only: ACI-norm-flatten,
    rule finite-surj[of  $\{X. X \subseteq ACI\text{-norm } ' ?X\}$  - flatten PLUS])
  have *:  $\bigwedge X. ACI\text{-norm } ' ?f (\lambda x. x) \ ' X = ?f\ ACI\text{-norm } ' ACI\text{-norm } ' X$  by
(auto simp: image-def)
  show finite  $\{X. X \subseteq ACI\text{-norm } ' ?X\}$ 
  by (rule finite-Collect-subsets)
  (auto simp: * intro!: finite-imageI[of - ?f ACI-norm] intro: finite-subset[OF
- Star])
  qed (fastforce dest!: subsetD[OF toplevel-summands-lderivs-Star] intro!: imageI)
next
  case (Not r) thus ?case by (auto simp: lderivs-Not) (blast intro: finite-surj)
next
  case (Inter r s)
  show ?case by (auto simp: lderivs-Inter intro!: finite-surj[OF finite-cartesian-product[OF
Inter]])
next
  case (Pr r)
  hence *: finite  $(\bigcup (\text{toplevel-summands } ' \{\ll lderivs\ xs\ r\} | xs. True\}))$  by auto
  have finite  $(\bigcup xs. \text{toplevel-summands } \ll lderivs\ xs\ r\gg)$  by (rule finite-subset[OF
- *]) auto
  hence fin: finite  $\{Pr \ \ll s\gg | s. \text{toplevel-summands } s \subseteq (\bigcup xs. \text{toplevel-summands } \ll lderivs\ xs\ r\gg)\}$ 
  by (intro finite-ACI-norm-toplevel-summands)
  have  $\{s. \exists xs. s = \ll lderivs\ xs\ (Pr\ r)\gg\} = \{\ll s\gg | s. \exists xs. s = lderivs\ xs\ (Pr\ r)\}$ 
by auto
  thus ?case using finite-subset[OF ACI-norm-lderivs-Pr, of r] fin unfolding
image-Collect by auto
qed

```

### 3.3 Wellformedness and language of derivatives

**lemma** wf-lderiv[simp]:  $wf\ n\ r \implies wf\ n\ (lderiv\ w\ r)$   
**by** (induct r arbitrary: n w) (auto simp add: Let-def)

**lemma** wf-lderivs[simp]:  $wf\ n\ r \implies wf\ n\ (lderivs\ ws\ r)$   
**by** (induct ws arbitrary: r) (auto intro: wf-lderiv)

**lemma** lQuot-map-project:  
**assumes**  $as \in \Sigma\ n\ A \subseteq \text{lists } (\Sigma\ (Suc\ n))$   
**shows**  $lQuot\ as\ (\text{map}\ project\ 'A) = \text{map}\ project\ ' (\bigcup a \in set\ (\text{embed}\ as). lQuot\ a\ A)$  (**is** ?L = ?R)  
**proof** (intro equalityI image-subsetI subsetI)

```

fix  $xss$  assume  $xss \in ?L$ 
with  $assms$  obtain  $zss$ 
  where  $zss: zss \in A$  as  $\#$   $xss = map\ project\ zss$ 
  unfolding  $lQuot-def$  by  $fastforce$ 
  hence  $xss = map\ project\ (tl\ zss)$  by  $auto$ 
  with  $zss\ assms(2)$  show  $xss \in ?R$  using  $embed[OF\ project, of - n]$  unfolding
 $lQuot-def$  by  $fastforce$ 
next
  fix  $xss$  assume  $xss \in (\bigcup a \in set\ (embed\ as).\ lQuot\ a\ A)$ 
  with  $assms(1)$  show  $map\ project\ xss \in lQuot\ as\ (map\ project\ 'A)$  unfolding
 $lQuot-def$ 
  by  $(fastforce\ intro!: rev-image-eqI\ simp: embed)$ 
qed

```

```

lemma  $lang-lderiv: \llbracket wf\ n\ r; w \in \Sigma\ n \rrbracket \implies lang\ n\ (lderiv\ w\ r) = lQuot\ w\ (lang\ n\ r)$ 
proof  $(induct\ r\ arbitrary: n\ w)$ 
  case  $(Pr\ r)$ 
  hence  $*$ :  $wf\ (Suc\ n)\ r \wedge w'. w' \in set\ (embed\ w) \implies w' \in \Sigma\ (Suc\ n)$  by  $(auto\ simp: embed)$ 
  from  $Pr(1)[OF\ *] lQuot-map-project[OF\ Pr(3)\ lang-subset-lists[OF\ *(1)]]$  show
 $?case$ 
  by  $(auto\ simp: wf-lderiv[OF\ *(1)])$ 
qed  $(auto\ simp: Let-def\ lang-final[symmetric])$ 

```

```

lemma  $lang-lderivs: \llbracket wf\ n\ r; wf-word\ n\ ws \rrbracket \implies lang\ n\ (lderivs\ ws\ r) = lQuots\ ws\ (lang\ n\ r)$ 
by  $(induct\ ws\ arbitrary: r)\ (auto\ simp: lang-lderiv)$ 

```

```

corollary  $lderivs-final:$ 
assumes  $wf\ n\ r\ wf-word\ n\ ws$ 
shows  $final\ (lderivs\ ws\ r) \longleftrightarrow ws \in lang\ n\ r$ 
using  $lang-lderivs[OF\ assms]\ lang-final[of\ lderivs\ ws\ r\ n]$  by  $auto$ 

```

```

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

```

### 3.4 Deriving preserves ACI-equivalence

```

lemma  $ACI-norm-PLUS:$ 
 $list-all2\ (\lambda r\ s. \ll r \gg = \ll s \gg)\ xs\ ys \implies \ll PLUS\ xs \gg = \ll PLUS\ ys \gg$ 
proof  $(induct\ rule: list-all2-induct)$ 
  case  $(Cons\ x\ xs\ y\ ys)$ 
  hence  $length\ xs = length\ ys$  by  $(elim\ list-all2-lengthD)$ 
  thus  $?case$  using  $Cons$  by  $(induct\ xs\ ys\ rule: list-induct2)\ auto$ 
qed  $simp$ 

```

```

lemma  $toplevel-summands-ACI-norm-lderiv:$ 
 $(\bigcup a \in toplevel-summands\ r. toplevel-summands\ \ll lderiv\ as\ \ll a \gg \gg) = toplevel-summands$ 

```

```

<lderv as <r>>
proof (induct r)
  case (Plus r1 r2) thus ?case
    unfolding toplevel-summands.simps toplevel-summands-ACI-norm
      toplevel-summands-lderv[of as <Plus r1 r2>] image-Un Union-Un-distrib
    by (simp add: image-UN)
qed (auto simp: Let-def)

theorem ACI-norm-lderv:
  <lderv as <r>> = <lderv as r>
proof (induct r arbitrary: as)
  case (Plus r1 r2) thus ?case
    unfolding lderiv.simps ACI-norm-flatten[of lderiv as <Plus r1 r2>]
      toplevel-summands-lderv[of as <Plus r1 r2>] image-Un image-UN
    by (auto simp: toplevel-summands-ACI-norm toplevel-summands-flatten-ACI-norm-image-Union)
      (auto simp: toplevel-summands-ACI-norm[symmetric] toplevel-summands-ACI-norm-lderv)
next
  case (Pr r)
    hence list-all2 (λr s. <r> = <s>)
      (map (λa. lderiv a <r>) (embed as)) (map (λa. lderiv a r) (embed as))
    unfolding list-all2-map1 list-all2-map2 by (intro list-all2-refl)
    thus ?case unfolding lderiv.simps ACI-norm.simps by (blast intro: ACI-norm-PLUS)
qed (simp-all add: Let-def)

corollary lderiv-preserves: <r> = <s>  $\implies$  <lderv as r> = <lderv as s>
  by (rule box-equals[OF - ACI-norm-lderv ACI-norm-lderv]) (erule arg-cong)

lemma lderivs-snoc[simp]: lderivs (ws @ [w]) r = (lderv w (ldervs ws r))
  by (induct ws arbitrary: r) auto

theorem ACI-norm-ldervs:
  <ldervs ws <r>> = <ldervs ws r>
proof (induct ws arbitrary: r rule: rev-induct)
  case (snoc w ws) thus ?case
    using ACI-norm-lderv[of w lderivs ws r] ACI-norm-lderv[of w lderivs ws <r>]
by auto
qed simp

lemma lderivs-alt: <ldervs w r> = fold (λa r. <lderv a r>) w <r>
  by (induct w arbitrary: r) (auto simp: ACI-norm-lderv)

lemma finite-fold-lderv: finite {fold (λa r. <lderv a r>) w <s> | w. True}
  using finite-ldervs unfolding lderivs-alt .

end

```

## 4 Some Useful Regular Operators

**primrec** *REV* :: 'a rexp  $\Rightarrow$  'a rexp **where**

$REV\ Zero = Zero$   
 $| REV\ Full = Full$   
 $| REV\ One = One$   
 $| REV\ (Atom\ a) = Atom\ a$   
 $| REV\ (Plus\ r\ s) = Plus\ (REV\ r)\ (REV\ s)$   
 $| REV\ (Times\ r\ s) = Times\ (REV\ s)\ (REV\ r)$   
 $| REV\ (Star\ r) = Star\ (REV\ r)$   
 $| REV\ (Not\ r) = Not\ (REV\ r)$   
 $| REV\ (Inter\ r\ s) = Inter\ (REV\ r)\ (REV\ s)$   
 $| REV\ (Pr\ r) = Pr\ (REV\ r)$

**lemma** *REV-REV[simp]*:  $REV\ (REV\ r) = r$   
**by** (*induct r auto*)

**lemma** *final-REV[simp]*:  $final\ (REV\ r) = final\ r$   
**by** (*induct r auto*)

**lemma** *REV-PLUS*:  $REV\ (PLUS\ xs) = PLUS\ (map\ REV\ xs)$   
**by** (*induct xs rule: list-singleton-induct auto*)

**lemma** (**in** *alphabet*) *wf-REV[simp]*:  $wf\ n\ r \Longrightarrow wf\ n\ (REV\ r)$   
**by** (*induct r arbitrary: n auto*)

**lemma** (**in** *project*) *lang-REV[simp]*:  $lang\ n\ (REV\ r) = rev\ 'lang\ n\ r$   
**by** (*induct r arbitrary: n (auto simp: image-image rev-map image-set-diff)*)

**context** *embed*

**begin**

**primrec** *rderiv* :: 'a  $\Rightarrow$  'b rexp  $\Rightarrow$  'b rexp **where**

$rderiv\ -\ Zero = Zero$   
 $| rderiv\ -\ Full = Full$   
 $| rderiv\ -\ One = Zero$   
 $| rderiv\ a\ (Atom\ b) = (if\ lookup\ b\ a\ then\ One\ else\ Zero)$   
 $| rderiv\ a\ (Plus\ r\ s) = Plus\ (rderiv\ a\ r)\ (rderiv\ a\ s)$   
 $| rderiv\ a\ (Times\ r\ s) =$   
 $\quad (let\ rs' = Times\ r\ (rderiv\ a\ s)$   
 $\quad\quad in\ if\ final\ s\ then\ Plus\ rs'\ (rderiv\ a\ r)\ else\ rs')$   
 $| rderiv\ a\ (Star\ r) = Times\ (Star\ r)\ (rderiv\ a\ r)$   
 $| rderiv\ a\ (Not\ r) = Not\ (rderiv\ a\ r)$   
 $| rderiv\ a\ (Inter\ r\ s) = Inter\ (rderiv\ a\ r)\ (rderiv\ a\ s)$   
 $| rderiv\ a\ (Pr\ r) = Pr\ (PLUS\ (map\ (\lambda a'. rderiv\ a'\ r)\ (embed\ a)))$

**primrec** *rderivs* **where**

$rderivs\ []\ r = r$

|  $rderiv (w\#ws) r = rderiv ws (rderiv w r)$

**lemma** *rderiv-snoc*:  $rderiv (ws @ [w]) r = rderiv w (rderiv ws r)$   
**by** (*induct ws arbitrary: r*) *auto*

**lemma** *rderiv-append*:  $rderiv (ws @ ws') r = rderiv ws' (rderiv ws r)$   
**by** (*induct ws arbitrary: r*) *auto*

**lemma** *rderiv-lderiv*:  $rderiv as r = REV (lderiv as (REV r))$   
**by** (*induct r arbitrary: as*) (*auto simp: Let-def o-def REV-PLUS*)

**lemma** *rderiv-lderivs*:  $rderiv w r = REV (lderivs w (REV r))$   
**by** (*induct w arbitrary: r*) (*auto simp: rderiv-lderiv*)

**lemma** *wf-rderiv[simp]*:  $wf n r \implies wf n (rderiv w r)$   
**unfolding** *rderiv-lderiv* **by** (*rule wf-REV[OF wf-lderiv[OF wf-REV]]*)

**lemma** *wf-rderivs[simp]*:  $wf n r \implies wf n (rderivs ws r)$   
**unfolding** *rderivs-lderivs* **by** (*rule wf-REV[OF wf-lderivs[OF wf-REV]]*)

**lemma** *lang-rderiv*:  $\llbracket wf n r; as \in \Sigma n \rrbracket \implies lang n (rderiv as r) = rQuot as (lang n r)$   
**unfolding** *rderiv-lderiv rQuot-rev-lQuot* **by** (*simp add: lang-lderiv*)

**lemma** *lang-rderivs*:  $\llbracket wf n r; wf-word n w \rrbracket \implies lang n (rderivs w r) = rQuots w (lang n r)$   
**unfolding** *rderivs-lderivs rQuots-rev-lQuots* **by** (*simp add: lang-lderivs*)

**corollary** *rderivs-final*:

**assumes** *wf n r wf-word n w*

**shows** *final (rderivs w r)  $\longleftrightarrow$  rev w  $\in$  lang n r*

**using** *lang-rderivs[OF assms] lang-final[of rderivs w r n]* **by** *auto*

**lemma** *toplevel-summands-REV[simp]*:  $toplevel-summands (REV r) = REV ' toplevel-summands r$   
**by** (*induct r*) *auto*

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

**proof** (*induct r*)

**case** (*Plus r s*)

**show** *?case*

**using**  $\llbracket unfold-abs-def = false \rrbracket$

**unfolding** *REV.simps ACI-norm.simps Plus[symmetric] image-Un[symmetric]*

*toplevel-summands.simps(1) toplevel-summands-ACI-norm toplevel-summands-REV*

**unfolding** *toplevel-summands.simps(1)[symmetric] ACI-norm-flatten toplevel-summands-REV*

**unfolding** *ACI-norm-flatten[symmetric] toplevel-summands-ACI-norm*

..

**qed** *auto*



**lemma** *ACI-norm-rderiv*:  $\langle\langle rderiv\ as\ \langle r \rangle\rangle\rangle = \langle\langle rderiv\ as\ r \rangle\rangle$   
**unfolding** *rderiv-lderiv* **by** (*metis ACI-norm-REV ACI-norm-lderiv*)

**lemma** *ACI-norm-rderivs*:  $\langle\langle rderivs\ w\ \langle r \rangle\rangle\rangle = \langle\langle rderivs\ w\ r \rangle\rangle$   
**unfolding** *rderivs-lderivs* **by** (*metis ACI-norm-REV ACI-norm-lderivs*)

**theorem** *finite-rderivs*: *finite*  $\{\langle\langle rderivs\ xs\ r \rangle\rangle \mid xs . True\}$   
**unfolding** *rderivs-lderivs*  
**by** (*subst ACI-norm-REV[symmetric]*) (*auto intro: finite-surj[OF finite-lderivs, of -  $\lambda r. \langle\langle REV\ r \rangle\rangle$ ]*)

**lemma** *lderiv-PLUS[simp]*:  $lderiv\ a\ (PLUS\ xs) = PLUS\ (map\ (lderiv\ a)\ xs)$   
**by** (*induct xs rule: list-singleton-induct*) *auto*

**lemma** *rderiv-PLUS[simp]*:  $rderiv\ a\ (PLUS\ xs) = PLUS\ (map\ (rderiv\ a)\ xs)$   
**by** (*induct xs rule: list-singleton-induct*) *auto*

**lemma** *lang-rderiv-lderiv*:  $lang\ n\ (rderiv\ a\ (lderiv\ b\ r)) = lang\ n\ (lderiv\ b\ (rderiv\ a\ r))$   
**by** (*induct r arbitrary: n a b*) (*auto simp: Let-def conc-assoc*)

**lemma** *lang-lderiv-rderiv*:  $lang\ n\ (lderiv\ a\ (rderiv\ b\ r)) = lang\ n\ (rderiv\ b\ (lderiv\ a\ r))$   
**by** (*induct r arbitrary: n a b*) (*auto simp: Let-def conc-assoc*)

**lemma** *lang-rderiv-lderivs[simp]*:  $\llbracket wf\ n\ r; wf\ word\ n\ w; a \in \Sigma\ n \rrbracket \implies lang\ n\ (rderiv\ a\ (lderivs\ w\ r)) = lang\ n\ (lderivs\ w\ (rderiv\ a\ r))$   
**by** (*induct w arbitrary: n r*)  
*(auto, auto simp: lang-lderivs lang-lderiv lang-rderiv lQuot-rQuot)*

**lemma** *lang-lderiv-rderivs[simp]*:  $\llbracket wf\ n\ r; wf\ word\ n\ w; a \in \Sigma\ n \rrbracket \implies lang\ n\ (lderiv\ a\ (rderivs\ w\ r)) = lang\ n\ (rderivs\ w\ (lderiv\ a\ r))$   
**by** (*induct w arbitrary: n r*)  
*(auto, auto simp: lang-rderivs lang-lderiv lang-rderiv lQuot-rQuot)*

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

**lemma** *lang-biderivs*:  $\llbracket wf\ n\ r; wf\ word\ n\ w1; wf\ word\ n\ w2 \rrbracket \implies lang\ n\ (biderivs\ w1\ w2\ r) = biQuots\ w1\ w2\ (lang\ n\ r)$   
**unfolding** *biderivs-def* **by** (*auto simp: lang-rderivs lang-lderivs in-lists-conv-set*)

**lemma** *wf-biderivs[simp]*:  $wf\ n\ r \implies wf\ n\ (biderivs\ w1\ w2\ r)$   
**unfolding** *biderivs-def* **by** *simp*

**corollary** *biderivs-final*:  
**assumes** *wf n r wf-word n w1 wf-word n w2*  
**shows** *final (biderivs w1 w2 r)  $\longleftrightarrow$  w1 @ rev w2  $\in$  lang n r*  
**using** *lang-biderivs[OF assms] lang-final[of biderivs w1 w2 r n]* **by** *auto*

**lemma** *ACI-norm-biderivs*:  $\llbracket \text{biderivs } w1 \ w2 \ \llbracket r \rrbracket \rrbracket = \llbracket \text{biderivs } w1 \ w2 \ r \rrbracket$   
**unfolding** *biderivs-def* **by** (*metis ACI-norm-lderiv ACI-norm-rderiv o-apply*)

**lemma** *finite*  $\{\llbracket \text{biderivs } w1 \ w2 \ r \rrbracket \mid w1 \ w2 \ . \ \text{True}\}$

**proof** –

**have**  $\{\llbracket \text{biderivs } w1 \ w2 \ r \rrbracket \mid w1 \ w2 \ . \ \text{True}\} =$

$(\bigcup s \in \{\llbracket \text{lderiv } as \ r \rrbracket \mid as \ . \ \text{True}\}. \ \{\llbracket \text{rderiv } bs \ s \rrbracket \mid bs \ . \ \text{True}\})$

**unfolding** *biderivs-def* **by** (*fastforce simp: ACI-norm-rderiv*)

**also have** *finite* ... **by** (*rule iffD2[OF finite-UN[OF finite-lderiv] ballI[OF finite-rderiv]]*)

**finally show** *?thesis* .

**qed**

**end**

#### 4.1 Quotienting by the same letter

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

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

**unfolding** *fin-cut-same-def* **by** *simp*

**lemma** *Least-fin-cut-same*:  $(\text{LEAST } n. \ \text{drop } n \ xs = \text{replicate } (\text{length } xs - n) \ y) =$

$\text{length } xs - \text{length } (\text{takeWhile } (\lambda x. \ x = y) \ (\text{rev } xs))$

(*is Least ?P = ?min*)

**proof** (*rule Least-equality*)

**show** *?P ?min* **by** (*induct xs rule: rev-induct*) (*auto simp: Suc-diff-le replicate-append-same*)

**next**

**fix**  $m$  **assume** *?P m*

**have**  $\text{length } xs - m \leq \text{length } (\text{takeWhile } (\lambda x. \ x = y) \ (\text{rev } xs))$

**proof** (*intro length-takeWhile-less-P-nth*)

**fix**  $i$  **assume**  $i < \text{length } xs - m$

**hence**  $\text{rev } xs ! i \in \text{set } (\text{drop } m \ xs)$

**by** (*induct xs arbitrary: i rule: rev-induct*) (*auto simp: nth-Cons'*)

**with**  $\langle ?P \ m \rangle$  **show**  $\text{rev } xs ! i = y$  **by** *simp*

**qed** *simp*

**thus**  $?min \leq m$  **by** *linarith*

**qed**

**lemma** *takeWhile-takes-all*:  $\text{length } xs = m \implies m \leq \text{length } (\text{takeWhile } P \ xs) \longleftrightarrow \text{Ball } (\text{set } xs) \ P$

**by** *hypsubst-thin* (*induct xs, auto*)

**lemma** *fin-cut-same-Cons[simp]*:  $\text{fin-cut-same } x \ (y \ \# \ xs) =$

(*if fin-cut-same } x \ xs = [] then if } x = y then [] else [y] else } y \ \# fin-cut-same } x \ xs*)

**unfolding** *fin-cut-same-def Least-fin-cut-same*

```

apply auto
apply (simp add: takeWhile-takes-all)
apply (simp add: takeWhile-takes-all)
apply auto
apply (metis (full-types) Suc-diff-le length-rev length-take While-le take-Suc-Cons)
apply (simp add: takeWhile-takes-all)
apply (subst takeWhile-append2)
apply auto
apply (simp add: takeWhile-takes-all)
apply auto
apply (metis (full-types) Suc-diff-le length-rev length-take While-le take-Suc-Cons)
done

lemma fin-cut-same-singleton[simp]: fin-cut-same x (xs @ [x]) = fin-cut-same x xs
  by (induct xs) auto

lemma fin-cut-same-replicate[simp]: fin-cut-same x (xs @ replicate n x) = fin-cut-same x xs
  by (induct n arbitrary: xs)
  (auto simp: replicate-append-same[symmetric] append-assoc[symmetric] simp del: append-assoc)

lemma fin-cut-sameE: fin-cut-same x xs = ys  $\implies$   $\exists m. xs = ys @ replicate m x$ 
  by (induct xs arbitrary: ys) (auto, metis replicate-Suc)

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

lemma SAMEQUOT-mono: A  $\subseteq$  B  $\implies$  SAMEQUOT a A  $\subseteq$  SAMEQUOT a B
  unfolding SAMEQUOT-def by auto

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

lemma finite-rderivs-same: finite { $\llcorner$ rderivs (replicate m a) r $\gg$  | m . True}
  by (auto intro: finite-subset[OF - finite-rderivs])

lemma wf-word-replicate[simp]: a  $\in$   $\Sigma$  n  $\implies$  wf-word n (replicate m a)
  by (induct m) auto

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

```

**proof** (*intro equalityI subsetI*)  
**fix** *xs* **assume**  $xs \in \text{star } \{[x]\}$   
**thus**  $xs \in \{\text{replicate } m \ x \mid m . \text{True}\}$  **by** (*induct xs*) (*auto, metis replicate-Suc*)  
**qed** (*auto intro: Ball-starI*)

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

**lemma** *wf-samequot*:  $\ll \text{wf } n \ r; a \in \Sigma \ n \gg \implies \text{wf } n \ (\text{samequot } a \ r)$   
**unfolding** *samequot-def wf.simps wf-flatten-PLUS[OF finite-rderivs-same]* **by** *auto*

**lemma** *lang-samequot*:  $\ll \text{wf } n \ r; a \in \Sigma \ n \gg \implies$   
 $\text{lang } n \ (\text{samequot } a \ r) = \text{SAMEQUOT } a \ (\text{lang } n \ r)$   
**unfolding** *SAMEQUOT-def samequot-def lang.simps lang-flatten-PLUS[OF finite-rderivs-same]*  
**apply** (*rule sym*)  
**apply** (*auto simp: lang-rderivs*)  
**apply** (*intro concI*)  
**apply** *auto*  
**apply** (*insert fin-cut-sameE[OF refl, of - a]*)  
**apply** (*drule meta-spec*)  
**apply** (*erule exE*)  
**apply** (*intro exI conjI*)  
**apply** (*rule refl*)  
**apply** (*auto simp: lang-rderivs*)  
**apply** (*erule subst*)  
**apply** *assumption*  
**apply** (*erule concE*)  
**apply** (*auto simp: lang-rderivs*)  
**apply** (*drule meta-spec*)  
**apply** (*erule exE*)  
**apply** (*intro exI conjI*)  
**defer**  
**apply** *assumption*  
**unfolding** *fin-cut-same-replicate*  
**apply** (*erule trans*)  
**unfolding** *fin-cut-same-replicate*  
**apply** (*rule refl*)  
**done**

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

**definition** *invar-rderiv-and-add* as  $r \ \text{brs} \equiv$   
 $(\text{if } \text{fst } \text{brs} \text{ then } \text{True} \text{ else } \ll r \text{deriv as } (\text{hd } (\text{snd } \text{brs})) \gg \in \text{set } (\text{snd } \text{brs})) \wedge$

$snd\ brs \neq [] \wedge distinct\ (snd\ brs) \wedge$   
 $(\forall i < length\ (snd\ brs).\ snd\ brs\ !\ i = \llcorner rderivs\ (replicate\ (length\ (snd\ brs) - 1$   
 $- i)\ as)\ r \gg)$

**lemma** *invar-rderiv-and-add-init*: *invar-rderiv-and-add as r (True, [ $\llcorner r \gg$ ])*  
**unfolding** *invar-rderiv-and-add-def* **by** *auto*

**lemma** *invar-rderiv-and-add-step*: *invar-rderiv-and-add as r brs  $\implies$  fst brs  $\implies$*   
*invar-rderiv-and-add as r (rderiv-and-add as brs)*  
**unfolding** *invar-rderiv-and-add-def* **by** *(cases brs) (auto simp:*  
*Let-def nth-Cons' ACI-norm-rderiv rderivs-snoc[symmetric] neq-Nil-conv replicate-append-same)*

**lemma** *rderivs-replicate-mult*:  $\llbracket \llcorner rderivs\ (replicate\ i\ as)\ r \gg = \llcorner r \gg; i > 0 \rrbracket \implies$   
 $\llcorner rderivs\ (replicate\ (m * i)\ as)\ r \gg = \llcorner r \gg$

**proof** *(induct m arbitrary: r)*

**case** *(Suc m)*

**hence**  $\llcorner rderivs\ (replicate\ (m * i)\ as)\ \llcorner rderivs\ (replicate\ i\ as)\ r \gg \gg = \llcorner r \gg$   
**by** *(auto simp: ACI-norm-rderivs)*

**thus** *?case* **by** *(auto simp: ACI-norm-rderivs replicate-add rderivs-append)*

**qed** *simp*

**lemma** *rderivs-replicate-mult-rest*:

**assumes**  $\llcorner rderivs\ (replicate\ i\ as)\ r \gg = \llcorner r \gg\ k < i$

**shows**  $\llcorner rderivs\ (replicate\ (m * i + k)\ as)\ r \gg = \llcorner rderivs\ (replicate\ k\ as)\ r \gg$  **(is**  
 $?L = ?R)$

**proof**  $-$

**have**  $?L = \llcorner rderivs\ (replicate\ k\ as)\ \llcorner rderivs\ (replicate\ (m * i)\ as)\ r \gg \gg$

**by** *(simp add: ACI-norm-rderivs replicate-add rderivs-append)*

**also have**  $\llcorner rderivs\ (replicate\ (m * i)\ as)\ r \gg = \llcorner r \gg$  **using** *assms*

**by** *(simp add: rderivs-replicate-mult)*

**finally show** *?thesis* **by** *(simp add: ACI-norm-rderivs)*

**qed**

**lemma** *rderivs-replicate-mod*:

**assumes**  $\llcorner rderivs\ (replicate\ i\ as)\ r \gg = \llcorner r \gg\ i > 0$

**shows**  $\llcorner rderivs\ (replicate\ m\ as)\ r \gg = \llcorner rderivs\ (replicate\ (m\ mod\ i)\ as)\ r \gg$  **(is**  
 $?L = ?R)$

**by** *(subst div-mult-mod-eq[symmetric, of m i])*

*(intro rderivs-replicate-mult-rest[OF assms(1)] mod-less-divisor[OF assms(2)])*

**lemma** *rderivs-replicate-diff*:  $\llbracket \llcorner rderivs\ (replicate\ i\ as)\ r \gg = \llcorner rderivs\ (replicate\ j$   
 $as)\ r \gg; i > j \rrbracket \implies$

$\llcorner rderivs\ (replicate\ (i - j)\ as)\ (rderivs\ (replicate\ j\ as)\ r) \gg = \llcorner rderivs\ (replicate$   
 $j\ as)\ r \gg$

**unfolding** *rderivs-append[symmetric] replicate-add[symmetric]* **by** *auto*

**lemma** *samequot-wf*:

**assumes** *wf n r while-option fst (rderiv-and-add as) (True, [ $\llcorner r \gg$ ]) = Some (b,*  
 $rs)$

shows  $wf\ n\ (PLUS\ rs)$   
**proof** –  
**have**  $\neg\ b$  **using** *while-option-stop*[*OF assms*(2)] **by** *simp*  
**from** *while-option-rule*[**where**  $P = invar\ rderiv\ and\ add\ as\ r,$   
*OF invar-rderiv-and-add-step assms*(2) *invar-rderiv-and-add-init*]  
**have**  $*$ : *invar-rderiv-and-add as r (b, rs)* **by** *simp*  
**thus**  $wf\ n\ (PLUS\ rs)$  **unfolding** *invar-rderiv-and-add-def wf-PLUS*  
**by** (*auto simp: in-set-conv-nth wf-rderivs*[*OF assms*(1)])  
**qed**

**lemma** *samequot-soundness*:

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

**proof** –  
**have**  $\neg\ b$  **using** *while-option-stop*[*OF assms*] **by** *simp*  
**moreover**  
**from** *while-option-rule*[**where**  $P = invar\ rderiv\ and\ add\ as\ r,$   
*OF invar-rderiv-and-add-step assms invar-rderiv-and-add-init*]  
**have**  $*$ : *invar-rderiv-and-add as r (b, rs)* **by** *simp*  
**ultimately obtain**  $i$  **where**  $i < length\ rs$  **and**  $\ll rderivs\ (replicate\ (length\ rs - Suc\ i)\ as)\ r \gg =$   
 $\ll rderivs\ (replicate\ (Suc\ (length\ rs - Suc\ 0))\ as)\ r \gg$  (**is**  $\ll rderivs\ ?x\ r \gg = -$ )  
**unfolding** *invar-rderiv-and-add-def* **by** (*auto simp: in-set-conv-nth hd-conv-nth ACI-norm-rderiv*  
*rderivs-snoc*[*symmetric*] *replicate-append-same*)  
**with**  $*$  **have**  $\ll rderivs\ ?x\ r \gg = \ll rderivs\ (replicate\ (length\ rs)\ as)\ r \gg$   
**by** (*auto simp: invar-rderiv-and-add-def*)  
**with**  $i$  **have** *cyc*:  $\ll rderivs\ (replicate\ (Suc\ i)\ as)\ (rderivs\ ?x\ r) \gg = \ll rderivs\ ?x\ r \gg$   
**by** (*fastforce dest: rderivs-replicate-diff*[*OF sym*])  
**{ fix**  $m$   
**have**  $\exists\ i < length\ rs.\ rs\ !\ i = \ll rderivs\ (replicate\ m\ as)\ r \gg$   
**proof** (*cases*  $m > length\ rs - Suc\ i$ )  
**case** *True*  
**with**  $i$  **obtain**  $m'$  **where**  $m: m = m' + length\ rs - Suc\ i$   
**by** *atomize-elim (auto intro: exI*[*of - m - (length rs - Suc i)*])  
**with**  $i$  **have**  $\ll rderivs\ (replicate\ m\ as)\ r \gg = \ll rderivs\ (replicate\ m'\ as)\ (rderivs\ ?x\ r) \gg$   
**unfolding** *replicate-add*[*symmetric*] *rderivs-append*[*symmetric*] **by** (*simp add: add.commute*)  
**also from** *cyc* **have**  $\dots = \ll rderivs\ (replicate\ (m'\ mod\ (Suc\ i))\ as)\ (rderivs\ ?x\ r) \gg$   
**by** (*elim rderivs-replicate-mod*) *simp*  
**also from**  $i$  **have**  $\dots = \ll rderivs\ (replicate\ (m'\ mod\ (Suc\ i) + length\ rs - Suc\ i)\ as)\ r \gg$   
**unfolding** *rderivs-append*[*symmetric*] *replicate-add*[*symmetric*] **by** (*simp add: add.commute*)  
**also from**  $m\ i$  **have**  $\dots = \ll rderivs\ (replicate\ ((m - (length\ rs - Suc\ i)))$

```

mod (Suc i) + length rs - Suc i) as) r>>
  by simp
  also have ... = <<rderivs (replicate (length rs - Suc (i - (m - (length rs -
Suc i)) mod (Suc i))) as) r>>
    by (subst Suc-diff-le[symmetric])
      (metis less-Suc-eq-le mod-less-divisor zero-less-Suc, simp add: add.commute)
  finally have  $\exists j < \text{length } rs. \llbracket rderivs (\text{replicate } m \text{ as}) r \rrbracket = \llbracket rderivs (\text{replicate}
(\text{length } rs - \text{Suc } j) \text{ as}) r \rrbracket$ 
    using i by (metis less-imp-diff-less)
  with * show ?thesis unfolding invar-rderiv-and-add-def by auto
next
case False
with i have  $\exists j < \text{length } rs. m = \text{length } rs - \text{Suc } j$ 
  by (induct m)
    (metis diff-self-eq-0 gr-implies-not0 lessI nat.exhaust,
    metis (no-types) One-nat-def Suc-diff-Suc diff-Suc-1 gr0-conv-Suc less-imp-diff-less
    not-less-eq not-less-iff-gr-or-eq)
  with * show ?thesis unfolding invar-rderiv-and-add-def by auto
qed
}
hence  $\bigcup (\text{lang } n \text{ ' } \{\llbracket rderivs (\text{replicate } m \text{ as}) r \rrbracket \mid m. \text{True}\}) \subseteq \text{lang } n \text{ (PLUS}
rs)$ 
  by (fastforce simp: in-set-conv-nth intro!: beXI[rotated])
  moreover from * have  $\text{lang } n \text{ (PLUS } rs) \subseteq \bigcup (\text{lang } n \text{ ' } \{\llbracket rderivs (\text{replicate}
m \text{ as}) r \rrbracket \mid m. \text{True}\})$ 
    unfolding invar-rderiv-and-add-def by (fastforce simp: in-set-conv-nth)
  ultimately show  $\text{lang } n \text{ (PLUS } rs) = \bigcup (\text{lang } n \text{ ' } \{\llbracket rderivs (\text{replicate } m \text{ as})
r \rrbracket \mid m. \text{True}\})$  by blast
qed

```

**lemma** *length-subset-card*:  $\llbracket \text{finite } X; \text{distinct } (x \# xs); \text{set } (x \# xs) \subseteq X \rrbracket \implies \text{length } xs < \text{card } X$   
 by (metis card-mono distinct-card impossible-Cons not-le-imp-less order-trans)

**lemma** *samequot-termination*:

```

assumes while-option fst (rderiv-and-add as) (True, [ $\llbracket r \rrbracket$ ]) = None (is ?cl =
None)
shows False
proof -
let ?D =  $\{\llbracket rderivs (\text{replicate } m \text{ as}) r \rrbracket \mid m. \text{True}\}$ 
let ?f =  $\lambda(b, rs). \text{card } ?D + 1 - \text{length } rs + (\text{if } b \text{ then } 1 \text{ else } 0)$ 
have  $\exists st. ?cl = \text{Some } st$ 
  apply (rule measure-while-option-Some[of invar-rderiv-and-add as r - - ?f])
  apply (auto simp: invar-rderiv-and-add-init invar-rderiv-and-add-step)
  apply (auto simp: invar-rderiv-and-add-def Let-def neq-Nil-conv in-set-conv-nth
    intro!: diff-less-mono2 length-subset-card[OF finite-rderivs-same, simplified])
  apply auto []
  apply fastforce
  apply (metis Suc-less-eq nth-Cons-Suc)

```

done  
with *assms* show *False* by *auto*  
qed

**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*:  $\llbracket wf\ n\ r; as \in \Sigma\ n \rrbracket \implies wf\ n\ (samequot-exec\ as\ r)$   
**unfolding** *samequot-exec-def*  
**by** (*cases while-option fst (rderiv-and-add as) (True, [<r>])*)  
(*auto dest: samequot-termination samequot-wf*)

**lemma** *samequot-exec-samequot*:  $lang\ n\ (samequot-exec\ as\ r) = lang\ n\ (samequot\ as\ r)$   
**unfolding** *samequot-exec-def samequot-def lang.simps lang-flatten-PLUS[OF finite-rderivs-same]*  
**by** (*cases while-option fst (rderiv-and-add as) (True, [<r>])*)  
(*auto dest: samequot-termination dest!: samequot-soundness[of - - - n] simp del: ACI-norm-lang*)

**lemma** *lang-samequot-exec*:  
 $\llbracket wf\ n\ r; as \in \Sigma\ n \rrbracket \implies lang\ n\ (samequot-exec\ as\ r) = SAMEQUOT\ as\ (lang\ n\ r)$   
**unfolding** *samequot-exec-samequot* **by** (*rule lang-samequot*)

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*:  $Prefix\ L = rev\ 'a\ Suffix\ (rev\ 'a\ L)$   
**unfolding** *Prefix-def Suffix-def*  
**by** (*auto simp: rev-append-invert*)  
*intro: image-eqI[of - rev, OF rev-rev-ident[symmetric]]*  
*image-eqI[of - rev, OF rev-append[symmetric]]*)

**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  $\{\ll\text{lderivs } w \ r\gg \mid w. \text{wf-word } n \ w\}$ 
  by (auto intro: finite-subset[OF - finite-lderivs])

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

lemma wf-SUFFIX[simp]:  $\text{wf } n \ r \Longrightarrow \text{wf } n \ (\text{SUFFIX } r)$ 
  unfolding SUFFIX-def by (intro iffD2[OF wf-flatten-PLUS[OF finite-lderivs-wf]])
  auto

lemma lang-SUFFIX[simp]:  $\text{lang } n \ r \Longrightarrow \text{lang } n \ (\text{SUFFIX } r) = \text{Suffix } (\text{lang } n \ r)$ 
  unfolding SUFFIX-def Suffix-def
  using lang-flatten-PLUS[OF finite-lderivs-wf] lang-lderivs wf-lang-wf-word
  by fastforce

lemma wf-PREFIX[simp]:  $\text{wf } n \ r \Longrightarrow \text{wf } n \ (\text{PREFIX } r)$ 
  unfolding PREFIX-def by auto

lemma lang-PREFIX[simp]:  $\text{lang } n \ r \Longrightarrow \text{lang } n \ (\text{PREFIX } r) = \text{Prefix } (\text{lang } n \ r)$ 
  unfolding PREFIX-def by (auto simp: Prefix-Suffix)

end

lemma take-drop-CycleI[intro!]:  $x \in L \Longrightarrow \text{drop } i \ x \ @ \ \text{take } i \ x \in \text{Cycle } L$ 
  unfolding Cycle-def by fastforce

lemma take-drop-CycleI'[intro!]:  $\text{drop } i \ x \ @ \ \text{take } i \ x \in L \Longrightarrow x \in \text{Cycle } L$ 
  by (drule take-drop-CycleI[of - - length x - i]) auto

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)  
**by** (induct xs rule: list-singleton-induct) auto

**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-dual* n (CoAtom b a) = set-unop-dual n b {[x] | x. lookup a x  $\wedge$  x  $\in$   $\Sigma$  n} |  
*lang-dual* n (CoPlus b r s) = set-binop-dual b (*lang-dual* n r) (*lang-dual* n s) |

$lang\text{-}dual\ n\ (CoTimes\ b\ r\ s) = set\text{-}unop\text{-}dual\ n\ b$   
 $(set\text{-}unop\text{-}dual\ n\ b\ (lang\text{-}dual\ n\ r)\ @\@ set\text{-}unop\text{-}dual\ n\ b\ (lang\text{-}dual\ n\ s))\ |\$   
 $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)))\ |\$   
 $lang\text{-}dual\ n\ (CoPr\ b\ r) = set\text{-}unop\text{-}dual\ n\ b\ (map\ project\ ' (set\text{-}unop\text{-}dual\ (n + 1)\ b\ (lang\text{-}dual\ (n + 1)\ r)))$

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

**by** (*induct r arbitrary: n*) (*auto elim: rev-subsetD[OF - conc-mono]*) *star-induct*  
*intro: iffD2[OF wf-word] wf-word-map-project*)

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

**proof** (*induct r arbitrary: n*)

**case** (*CoPr b r*) **thus** ?*case by* (*cases b*) (*fastforce intro!: project*)  
**qed** (*auto simp: conc-subset-lists star-subset-lists*)

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

**by** (*induct r arbitrary: n*) (*auto intro: concI[of [] - [], simplified]*)

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

$lang\text{-}dual\ n\ (CoPLUS\text{-}dual\ True\ xs) = (\bigcup r \in set\ xs. lang\text{-}dual\ n\ r)$

**by** (*induct xs rule: list-singleton-induct*) *auto*

**lemma** *lang-dual-CoPLUS-dual[simp]*:

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

**by** (*induct xs rule: list-singleton-induct*) *auto*

**end**

**context** *embed*

**begin**

**primrec** *lderiv-dual* ::  $'a \Rightarrow 'b\ rexp\text{-}dual \Rightarrow 'b\ rexp\text{-}dual$  **where**

$lderiv\text{-}dual\ -\ (CoZero\ b) = (CoZero\ b)$

$| lderiv\text{-}dual\ -\ (CoOne\ b) = (CoZero\ b)$

$| lderiv\text{-}dual\ a\ (CoAtom\ b\ c) = (if\ lookup\ c\ a\ then\ CoOne\ b\ else\ CoZero\ b)$

$| lderiv\text{-}dual\ a\ (CoPlus\ b\ r\ s) = CoPlus\ b\ (lderiv\text{-}dual\ a\ r)\ (lderiv\text{-}dual\ a\ s)$

$| lderiv\text{-}dual\ a\ (CoTimes\ b\ r\ s) =$

$(let\ r's = CoTimes\ b\ (lderiv\text{-}dual\ a\ r)\ s$

$in\ if\ bool\text{-}unop\text{-}dual\ b\ (final\text{-}dual\ r)\ then\ CoPlus\ b\ r's\ (lderiv\text{-}dual\ a\ s)\ else\ r's)$

$| lderiv\text{-}dual\ a\ (CoStar\ b\ r) = CoTimes\ b\ (lderiv\text{-}dual\ a\ r)\ (CoStar\ b\ r)$

$| lderiv\text{-}dual\ a\ (CoPr\ b\ r) = CoPr\ b\ (CoPLUS\text{-}dual\ b\ (map\ (\lambda a'. lderiv\text{-}dual\ a'\ r)\ (embed\ a)))$

**primrec** *lderivs-dual* **where**

$lderivs\text{-}dual\ []\ r = r$

$| lderivs\text{-}dual\ (w\#ws)\ r = lderivs\text{-}dual\ ws\ (lderiv\text{-}dual\ w\ r)$

**lemma** *wf-dual-lderiv-dual*[simp]:  $wf\text{-dual } n \ r \implies wf\text{-dual } n \ (lderiv\text{-dual } w \ r)$   
**by** (*induct r arbitrary: n w*) (*auto simp add: Let-def*)

**lemma** *wf-dual-lderivs-dual*[simp]:  $wf\text{-dual } n \ r \implies wf\text{-dual } n \ (lderivs\text{-dual } ws \ r)$   
**by** (*induct ws arbitrary: r*) (*auto intro: wf-dual-lderiv-dual*)

**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** (*induct r arbitrary: n w*)

**case** (*CoPr b r*)

**hence** \*:  $wf\text{-dual } (Suc \ n) \ r \wedge w'. w' \in set \ (embed \ w) \implies w' \in \Sigma \ (Suc \ n)$  **by**  
(*auto simp: embed*)

**then show** ?*case using* *lQuot-map-project*[*OF CoPr(3) lang-dual-subset-lists*[*OF*  
\*(1)]]

*lQuot-map-project*[*OF CoPr(3) Diff-subset, of lang-dual (n + 1) r*]

**by** (*simp-all add: CoPr(1,3)*)

**qed** (*auto 0 3 simp: Let-def lang-dual-final-dual*[*symmetric*])

**lemma** *lang-dual-lderivs-dual*:  $\llbracket wf\text{-dual } n \ r; wf\text{-word } n \ ws \rrbracket \implies$   
 $lang\text{-dual } n \ (lderivs\text{-dual } ws \ r) = lQuots \ ws \ (lang\text{-dual } n \ r)$   
**by** (*induct ws arbitrary: r*) (*auto simp: lang-dual-lderiv-dual*)

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

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

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

**using** *lang-dual-lderivs-dual*[*OF assms*] *lang-dual-final-dual*[*of lderivs-dual ws r*  
*n*] **by** *auto*

**end**

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

$pnCoPlus \ b1 \ (CoZero \ b2) \ r = (if \ b1 = b2 \ then \ r \ else \ CoZero \ b2)$

|  $pnCoPlus \ b1 \ r \ (CoZero \ b2) = (if \ b1 = b2 \ then \ r \ else \ CoZero \ b2)$

|  $pnCoPlus \ b1 \ (CoPlus \ b2 \ r \ s) \ t =$

$(if \ b1 = b2 \ then \ pnCoPlus \ b2 \ r \ (pnCoPlus \ b2 \ s \ t) \ else \ CoPlus \ b1 \ (CoPlus \ b2 \ r$   
*s*) *t*)

|  $pnCoPlus \ b1 \ r \ (CoPlus \ b2 \ s \ t) =$

$(if \ b1 = b2 \ then$

$(if \ r = s \ then \ (CoPlus \ b2 \ s \ t)$

$else \ if \ r \leq s \ then \ CoPlus \ b2 \ r \ (CoPlus \ b2 \ s \ t)$

$else \ CoPlus \ b2 \ s \ (pnCoPlus \ b2 \ r \ t))$

$else \ CoPlus \ b1 \ r \ (CoPlus \ b2 \ s \ t))$

|  $pnCoPlus \ b \ r \ s =$

$(if \ r = s \ then \ r$

$else \ if \ r \leq s \ then \ CoPlus \ b \ r \ s$

$else \ CoPlus \ b \ s \ r)$

```

lemma (in alphabet) wf-dual-pnCoPlus[simp]:  $\llbracket wf\text{-dual } n \ r; \ wf\text{-dual } n \ s \rrbracket \implies wf\text{-dual } n \ (pnCoPlus \ b \ r \ s)$ 
  by (induct b r s rule: pnCoPlus.induct) auto

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 (induct b r s rule: pnCoPlus.induct)
  case 1 thus ?case by (auto dest: lang-dual-subset-lists)
next
  case 2-1 thus ?case by auto
next
  case 2-2 thus ?case by auto
next
  case 2-3 thus ?case by (auto dest: lang-dual-subset-lists)
next
  case 2-4 thus ?case by (auto dest!: lang-dual-subset-lists dest: subsetD[OF conc-subset-lists, unfolded in-lists-conv-set, rotated -1])
next
  case 2-5 thus ?case by (auto dest!: lang-dual-subset-lists dest: subsetD[OF star-subset-lists, unfolded in-lists-conv-set, rotated -1])
next
  case 2-6 thus ?case by (auto 4 4 dest!: lang-dual-subset-lists intro: project)
next
  case 3-1 thus ?case by auto
next
  case 3-2 thus ?case by auto
next
  case 3-3 thus ?case by auto
next
  case 3-4 thus ?case by auto
next
  case 3-5 thus ?case by auto
next
  case 3-6 thus ?case by auto
next
  case 4-1 thus ?case by auto
next
  case 4-2 thus ?case by auto
next
  case 4-3 thus ?case by auto
next
  case 4-4 thus ?case by auto
next
  case 4-5 thus ?case by auto
next
  case 5-1 thus ?case by auto
next
  case 5-2 thus ?case by auto
next

```

```

    case 5-3 thus ?case by auto
next
    case 5-4 thus ?case by auto
next
    case 5-5 thus ?case by auto
next
    case 5-6 thus ?case by auto
next
    case 5-7 thus ?case by auto
next
    case 5-8 thus ?case by auto
next
    case 5-9 thus ?case by auto
next
    case 5-10 thus ?case
    by auto (metis (no-types, hide-lams) Cons-in-lists-iff Diff-iff imageI list.simps(8)
list.simps(9) lists.Nil)+
next
    case 5-11 thus ?case by auto
next
    case 5-12 thus ?case by auto
next
    case 5-13 thus ?case by auto
next
    case 5-14 thus ?case by auto
next
    case 5-15 thus ?case by auto
next
    case 5-16 thus ?case by auto
next
    case 5-17 thus ?case by auto
next
    case 5-18 thus ?case by auto
next
    case 5-19 thus ?case by (auto dest!: lang-dual-subset-lists dest:
subsetD[OF star-subset-lists, unfolded in-lists-conv-set, rotated -1])
next
    case 5-20 thus ?case by auto
next
    case 5-21 thus ?case by auto
next
    case 5-22 thus ?case
    by auto (metis (no-types, hide-lams) Cons-in-lists-iff Diff-iff imageI list.simps(8)
list.simps(9) lists.Nil)+
next
    case 5-23 thus ?case by auto
next
    case 5-24 thus ?case by auto
next

```

**case 5-25 thus ?case by auto**  
**qed**

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

**lemma** (in *alphabet*) *wf-dual-pnCoTimes[simp]*:  $\llbracket wf-dual\ n\ r; wf-dual\ n\ s \rrbracket \Longrightarrow wf-dual\ n\ (pnCoTimes\ b\ r\ s)$   
**by** (induct *b* *r* *s* rule: *pnCoTimes.induct*) *auto*

**lemma** (in *project*) *lang-dual-pnCoTimes[simp]*:  $\llbracket wf-dual\ n\ r; wf-dual\ n\ s \rrbracket \Longrightarrow lang-dual\ n\ (pnCoTimes\ b\ r\ s) = lang-dual\ n\ (CoTimes\ b\ r\ s)$   
**apply** (induct *b* *r* *s* rule: *pnCoTimes.induct*)  
**apply** (*auto*, *auto* dest!: *lang-dual-subset-lists* dest: *project*  
*subsetD[OF star-subset-lists, unfolded in-lists-conv-set, rotated -1]*  
*subsetD[OF conc-subset-lists, unfolded in-lists-conv-set, rotated -1]*)  
**by** (*metis* (*full-types*) *Diff-iff conc-epsilon(1) double-diff empty-subsetI in-listsI insert-subset lists.Nil subset-refl*)

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

**lemma** (in *alphabet*) *wf-dual-pnCoPr[simp]*:  $wf-dual\ (Suc\ n)\ r \Longrightarrow wf-dual\ n\ (pnCoPr\ b\ r)$   
**by** (induct *b* *r* rule: *pnCoPr.induct*) *auto*

**lemma** (in *project*) *lang-dual-pnCoPr[simp]*:  $wf-dual\ (Suc\ n)\ r \Longrightarrow lang-dual\ n\ (pnCoPr\ b\ r) = lang-dual\ n\ (CoPr\ b\ r)$   
**by** (induct *b* *r* rule: *pnCoPr.induct*) *auto*

**primrec** *pnorm-dual* :: *'a::linorder rexp-dual*  $\Rightarrow$  *'a rexp-dual* **where**  
*pnorm-dual* (*CoZero* *b*) = (*CoZero* *b*)  
| *pnorm-dual* (*CoOne* *b*) = (*CoOne* *b*)  
| *pnorm-dual* (*CoAtom* *b* *a*) = (*CoAtom* *b* *a*)  
| *pnorm-dual* (*CoPlus* *b* *r* *s*) = *pnCoPlus* *b* (*pnorm-dual* *r*) (*pnorm-dual* *s*)

|  $pnorm\text{-dual } (CoTimes\ b\ r\ s) = pnCoTimes\ b\ (pnorm\text{-dual } r)\ s$   
|  $pnorm\text{-dual } (CoStar\ b\ r) = CoStar\ b\ r$   
|  $pnorm\text{-dual } (CoPr\ b\ r) = pnCoPr\ b\ (pnorm\text{-dual } r)$

**lemma** (in *alphabet*)  $wf\text{-dual}\text{-}pnorm\text{-dual}[simp]: wf\text{-dual } n\ r \implies wf\text{-dual } n\ (pnorm\text{-dual } r)$   
**by** (induct *r arbitrary: n*) *auto*

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

**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**

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

**lemma** (in *alphabet*)  $wf\text{-dual}\text{-}CoNot[simp]: wf\text{-dual } n\ r \implies wf\text{-dual } n\ (CoNot\ r)$   
**by** (induct *r arbitrary: n*) *auto*

**lemma** (in *project*)  $lang\text{-dual}\text{-}CoNot[simp]: wf\text{-dual } n\ r \implies lang\text{-dual } n\ (CoNot\ r) = lists\ (\Sigma\ n) - lang\text{-dual } n\ r$   
**apply** (induct *r arbitrary: n*)  
**apply** (*auto dest!: lang-dual-subset-lists simp: double-diff intro!: project*)  
**apply** *force*  
**apply** (*metis (full-types) Diff-subset contra-subsetD in-listsD star-subset-lists*)  
**done**

**lemma** (in *alphabet*)  $wf\text{-dual}\text{-}rexp\text{-dual-of}[simp]: wf\ n\ r \implies wf\text{-dual } n\ (rexp\text{-dual-of } r)$   
**by** (induct *r arbitrary: n*) *auto*

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



```

r) = lang n r
  by (induct r arbitrary: n) auto

```

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)
  unfolding image2p-def fun-eq-iff by auto

```

```

lemma image2p-apply: BNF-Greatest-Fixpoint.image2p f g R x y = ( $\exists x' y'. R x'
y' \wedge f x' = x \wedge g y' = y$ )
  unfolding image2p-def fun-eq-iff by auto

```

```

lemma rtrancl-fold-product:

```

```

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

```

```

proof -

```

```

{ fix x :: ('a  $\times$  'a)  $\times$  'a  $\times$  'a
  obtain r s r' s' where x: x = ((r, s), (r', s')) by (cases x) auto
  have ((r, s), (r', s'))  $\in$  ?L  $\implies$  ((r, s), (r', s'))  $\in$  ?R
  proof(induction rule: converse-rtrancl-induct2)
    case refl show ?case by(force intro!: fold-simps(1)[symmetric])
  next
    case step thus ?case by(force intro!: fold-simps(2)[symmetric])
  qed
  with x have x  $\in$  ?L  $\implies$  x  $\in$  ?R by simp
} moreover

```

```

{ fix x :: ('a  $\times$  'a)  $\times$  'a  $\times$  'a
  obtain r s r' s' where x: x = ((r, s), (r', s')) by (cases x) auto
  { fix w have  $\forall x \in$  set w. x  $\in$  A  $\implies$  ((r, s), fold f w r, fold f w s)  $\in$  ?L
    proof(induction w rule: rev-induct)
      case Nil show ?case by simp
    next
      case snoc thus ?case by (auto elim!: rtrancl-into-rtrancl)
    qed
  }
  hence ((r, s), (r', s'))  $\in$  ?R  $\implies$  ((r, s), (r', s'))  $\in$  ?L by auto
  with x have x  $\in$  ?R  $\implies$  x  $\in$  ?L by simp
} ultimately show ?thesis by blast

```

```

qed

```

```

lemma in-fold-lQuot: v  $\in$  fold lQuot w L  $\longleftrightarrow$  w @ v  $\in$  L
  by (induct w arbitrary: L) (simp-all add: lQuot-def)

```

**lemma** (in project) lang-eq-ext:  $\llbracket wf\ n\ r; wf\ n\ s \rrbracket \implies (lang\ n\ r = lang\ n\ s) =$   
 $(\forall w \in lists(\Sigma\ n). w \in lang\ n\ r \longleftrightarrow w \in lang\ n\ s)$   
**by** (auto dest!: lang-subset-lists)

**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**  $lang\ n\ r = lang\ n\ s \longleftrightarrow (\forall (K, L) \in \mathfrak{B}. [] \in K \longleftrightarrow [] \in L)$   
**unfolding** lang-eq-ext[OF WF]  $\mathfrak{B}$ -def  
**by** (subst (1 2) in-fold-lQuot[of [], simplified, symmetric]) auto

**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**  $L\ init[simp]: wf\ n\ r \implies L\ (init\ r) = lang\ n\ r$   
**assumes**  $L\ delta[simp]: \llbracket a \in set\ (\sigma\ n); wf\ state\ s \rrbracket \implies L\ (delta\ a\ s) = lQuot\ a$   
 $(L\ s)$   
**assumes**  $final\ iff\ Nil[simp]: final\ s \longleftrightarrow [] \in L\ s$   
**assumes**  $L\ wf\ state[dest]: wf\ state\ s \implies L\ s \subseteq lists\ (set\ (\sigma\ n))$   
**assumes**  $init\ wf\ state[simp]: wf\ n\ r \implies wf\ state\ (init\ r)$   
**assumes**  $delta\ wf\ state[simp]: \llbracket a \in set\ (\sigma\ n); wf\ state\ s \rrbracket \implies wf\ state\ (delta\ a\ s)$   
**assumes**  $L\ post[simp]: wf\ state\ s \implies L\ (post\ s) = L\ s$   
**assumes**  $wf\ state\ post[simp]: wf\ state\ s \implies wf\ state\ (post\ s)$   
**begin**

**lemma**  $L\ deltas[simp]: \llbracket wf\ word\ n\ w; wf\ state\ s \rrbracket \implies L\ (fold\ delta\ w\ s) = fold\ lQuot\ w\ (L\ s)$   
**by** (induction w arbitrary: s) auto

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

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

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

**unfolding** progression-def image2p-def **by** fastforce

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

**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**

{ **fix**  $n$  **from** *assms* **have**  $(f^{\wedge n}) R \rightarrow (f^{\wedge \text{Suc } n}) R$

**by** (*induct*  $n$ ) (*auto simp: bisimulation-upto-def compatible-def*) }

**then show** *bisimulation*  $(\text{SUP } n. (f^{\wedge n}) R)$

**unfolding** *bisimulation-def* **by** (*auto simp del: funpow.simps*)

**show**  $R \leq (\text{SUP } n. (f^{\wedge n}) R)$  **by** (*auto intro!: exI[of - 0]*)

**qed**

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

**unfolding** *bisim-def* **by** (*auto simp: lQuot-def*)

**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** (*rule set-eqI*)

**fix**  $w$

**from**  $R$   $WF$  **show**  $w \in L s1 \longleftrightarrow w \in L s2$

**proof** (*induction w arbitrary: s1 s2*)

**case** *Nil* **then show** *?case* **using** *bisim* **by** *simp*

**next**

**case**  $(\text{Cons } a w s1 s2)$

**show** *?case*

**proof** *cases*

**assume**  $a: a \in \text{set } (\sigma n)$

**with**  $\langle R s1 s2 \rangle$  **obtain**  $s1' s2'$  **where**  $R s1' s2'$  *wf-state*  $s1'$  *wf-state*  $s2'$  **and**

$*[\text{symmetric}]$ :  $\text{post } s1' = \text{post } (\text{delta } a s1)$   $\text{post } s2' = \text{post } (\text{delta } a s2)$

**using** *bisim* **unfolding** *image2p-apply* **by** *blast*

**then have**  $w \in L (\text{post } (\text{delta } a s1)) \longleftrightarrow w \in L (\text{post } (\text{delta } a s2))$

```

    unfolding * using Cons.IH[of s1' s2'] by simp
  with a Cons.prem1(2,3) show ?case by (simp add: lQuot-def)
next
  assume a ∉ set (σ n)
  thus ?case using Cons.prem1 bisim by force
qed
qed
qed

```

**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 (rule bisimulation-upto-bisimulation[OF assms(5,1)])
  fix S assume R ≤ S
  assume bisimulation S
  then show L s1 = L s2
  proof (rule coinduction[OF - WF])
    from ⟨R ≤ S⟩ ⟨R s1 s2⟩ show S s1 s2 by blast
  qed
qed

```

**fun** *test-invariant* **where**

```

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

```

**fun** *step-invariant* **where** *step-invariant* (ws, ps, N) =

```

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

```

**fun** *step* **where** *step* (ws, N) =

```

  (let
    (r, s) = hd ws;
    succs = map (λa.
      let r' = delta a r; s' = delta a s
      in ((r', s'), (post r', post s'))) (σ n);
    new = remdups' snd (filter (λ(-, rs). rs ∉ N) succs)
  in (tl ws @ map fst new, set (map snd new) ∪ 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*  $r\ s = (\lambda(ws, ps, N).$   
 $(r, s) \in \text{snd } \langle \text{set } ws \cup \text{set } ps \wedge$   
 $\text{distinct } (\text{map } \text{snd } ws \ @ \ ps) \wedge$   
 $\text{bij-betw } (\text{map-prod } \text{post } \text{post}) (\text{set } (\text{map } \text{snd } ws \ @ \ ps)) \ N \wedge$   
 $(\forall (w, r', s') \in \text{set } ws. \text{fold } \text{delta } (\text{rev } w) \ r = r' \wedge \text{fold } \text{delta } (\text{rev } w) \ s = s' \wedge$   
 $\text{wf-word } n \ (\text{rev } w) \wedge \text{wf-state } r' \wedge \text{wf-state } s') \wedge$   
 $(\forall (r', s') \in \text{set } ps. (\exists w. \text{fold } \text{delta } w \ r = r' \wedge \text{fold } \text{delta } w \ s = s') \wedge$   
 $\text{wf-state } r' \wedge \text{wf-state } s' \wedge (\text{final } r' \longleftrightarrow \text{final } s') \wedge$   
 $(\forall a \in \text{set } (\sigma \ n). (\text{post } (\text{delta } a \ r'), \text{post } (\text{delta } a \ s')) \in N)))$

**lemma** *invariant-start:*

$\llbracket \text{wf-state } r; \text{wf-state } s \rrbracket \implies \text{invariant } r\ s \ (\llbracket [], r, s \rrbracket, [], \{(\text{post } r, \text{post } s)\})$   
**by** (*auto simp add: invariant-def bij-betw-def*)

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

**assumes** *step-invariant*  $(ws, ps, N) = (ws', ps', N')$   
**shows**  $\text{snd } \langle \text{set } ws \cup \text{set } ps \subseteq \text{snd } \langle \text{set } ws' \cup \text{set } ps' \rangle$   
**using** *assms proof* (*intro subsetI, elim UnE*)  
**fix**  $x$  **assume**  $x \in \text{snd } \langle \text{set } ws \rangle$   
**with** *assms show*  $x \in \text{snd } \langle \text{set } ws' \cup \text{set } ps' \rangle$   
**proof** (*cases*  $x = \text{snd } (\text{hd } ws)$ )  
**case** *False with*  $\langle x \in \text{image } \text{snd } (\text{set } ws) \rangle$  **have**  $x \in \text{snd } \langle \text{set } (\text{tl } ws) \rangle$  **by** (*cases*  
 $ws$ ) *auto*  
**with** *assms show* *?thesis by* (*auto split: prod.splits simp: Let-def*)  
**qed** (*auto split: prod.splits simp: Let-def*)  
**qed** (*auto split: prod.splits simp: Let-def*)

**lemma** *step-invariant-unfold:* *step-invariant*  $(w \# ws, ps, N) = (ws', ps', N') \implies$   
 $(\exists xs\ r\ s.$

$w = (xs, r, s) \wedge ps' = (r, s) \# ps \wedge$   
 $ws' = ws \ @ \ \text{remdups}' (\text{map-prod } \text{post } \text{post} \ o \ \text{snd}) (\text{filter } (\lambda(-, p). \text{map-prod } \text{post}$   
 $\text{post } p \notin N)$   
 $(\text{map } (\lambda a. (a \# xs, \text{delta } a \ r, \text{delta } a \ s)) (\sigma \ n))) \ (\sigma \ n))) \wedge$   
 $N' = \text{set } (\text{map } (\lambda a. (\text{post } (\text{delta } a \ r), \text{post } (\text{delta } a \ s))) (\sigma \ n)) \cup N$   
**by** (*auto split: prod.splits dest!: mp-remdups'*)  
*simp: Let-def filter-map set-n-lists image-Collect image-image comp-def*)

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

**proof** (*unfold invariant-def, (split prod.splits)+, elim case-prodE conjE, clarify,*  
*intro allI impI conjI*)

**fix**  $ws\ ps\ N\ ws'\ ps'\ N'$

**assume** *test-invariant:* *test-invariant*  $(ws, ps, N)$

**and** *step-invariant:* *step-invariant*  $(ws, ps, N) = (ws', ps', N')$



**lemma** *step-commute*:  $ws \neq [] \implies$   
*(case step-invariant (ws, ps, N) of (ws', ps', N')  $\Rightarrow$  (map snd ws', N')) = step*  
*(map snd ws, N)*  
**apply** (*auto split: prod.splits*)  
**apply** (*auto simp only: step-invariant.simps step.simps Let-def map-afst-remdups'*  
*filter-map list.map-comp afst-def map-prod-def snd-conv id-def*)  
**apply** (*auto simp: filter-map comp-def map-tl hd-map*)  
**apply** (*intro image-eqI, auto*)  
**done**

**lemma** *closure-invariant-closure*:  
*map-option ( $\lambda(ws, ps, N). (map\ snd\ ws, N)$ ) (closure-invariant (ws, ps, N)) =*  
*closure (map snd ws, N)*  
**unfolding** *closure-invariant-def closure-def*  
**by** (*rule trans[OF while-option-commute[of - test - - step]]*)  
*(auto split: list.splits simp del: step-invariant.simps step.simps list.map simp:*  
*step-commute)*

**lemma**  
**assumes** *result: closure-invariant ([([], init r, init s)], [], {(post (init r), post*  
*(init s))}) =*  
*Some(ws, ps, N) (is closure-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*  
**proof** –  
**from** *WF have wf-state: wf-state ?r wf-state ?s by simp-all*  
**from** *invariant invariant-start[OF wf-state] have invariant-ps: invariant ?r ?s*  
*(ws, ps, N)*  
**by** (*rule while-option-rule[OF - result[unfolded closure-invariant-def]]*)  
**{ assume**  $ws = []$   
**with** *invariant-ps have bisimulation (in-rel (set ps)) (?r, ?s)  $\in$  set ps*  
**by** (*auto simp: bij-betw-def invariant-def bisimulation-def progression-def*  
*image2p-in-rel)*  
**with** *wf-state have L ?r = L ?s by (auto dest: coinduction)*  
**with** *WF show lang n r = lang n s by simp*  
**}**  
**{ assume**  $ws \neq []$   
**then obtain**  $w\ r'\ s'\ ws'$  **where**  $ws = (w, r', s') \# ws'$  **by** (*cases ws*) *auto*  
**with** *invariant-ps have*  $r' = \text{fold delta (rev w) (init r)}$   $s' = \text{fold delta (rev w)}$   
*(init s)*  
*wf-word n (rev w) unfolding invariant-def by auto*  
**moreover have**  $\neg \text{test-invariant} ((w, r', s') \# ws', ps, N)$   
**by** (*rule while-option-stop[OF result[unfolded ws closure-invariant-def]]*)  
**ultimately have**  $\text{rev (fst (hd ws))} \in L\ ?r \longleftrightarrow \text{rev (fst (hd ws))} \notin L\ ?s$   
**unfolding**  $ws$  **using** *wf-state by (simp add: in-fold-lQuot)*  
**with** *WF show*  $\text{rev (fst (hd ws))} \in \text{lang } n\ r \longleftrightarrow \text{rev (fst (hd ws))} \notin \text{lang } n\ s$   
**by** *simp*

}  
qed

**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*  
**using** *trans*[*OF* *closure-invariant-closure*[*of*  $([], init\ r, init\ s)$ , *simplified*] *result*]  
**by** (*auto* *dest*: *closure-invariant-sound*[*OF* - *WF*])

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

*check-eqv* *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-eqv-sound*:

**assumes** *check-eqv* *r* *s* **and** *WF*: *wf* *n* *r* *wf* *n* *s*  
**shows** *lang* *n* *r* = *lang* *n* *s*  
**using** *closure-sound* *assms*  
**by** (*auto* *simp*: *check-eqv-def* *Let-def* *split*: *option.splits* *list.splits*)

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

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

**lemma** *counterexample-sound*:

**assumes** *result*: *counterexample* *r* *s* = *Some* *w* **and** *WF*: *wf* *n* *r* *wf* *n* *s*  
**shows**  $w \in lang\ n\ r \longleftrightarrow w \notin lang\ n\ s$   
**using** *assms* **unfolding** *counterexample-def* *Let-def*  
**by** (*auto* *dest*!: *counterexample*[*of* *r* *s*] *split*: *option.splits* *list.splits*)

Auxiliary executable functions:

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

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

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

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



*in ((new @ tl ws, post ' set new  $\cup$  Z), set 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 = final (fold delta w (init s))

**lemma** *match-correct*:  $\llbracket \text{wf-word } n \ w; \text{ wf } n \ s \rrbracket \Longrightarrow \text{match } s \ w \longleftrightarrow w \in \text{lang } n \ s$   
**unfolding** *match-def*  
**by** (induct w arbitrary: s) (auto simp: in-fold-lQuot lQuot-def)

**end**

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

**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  
**and** *init* :: 'b rexp  $\Rightarrow$  's  
**and** *delta* :: 'a  $\Rightarrow$  's  $\Rightarrow$  's  
**and** *final* :: 's  $\Rightarrow$  bool  
**and** *wf-state* :: 's  $\Rightarrow$  bool  
**and** *post* :: 's  $\Rightarrow$  's  
**and** *L* :: 's  $\Rightarrow$  'a lang  
**and** *n* :: nat +

**assumes** *fin*: finite {fold delta w (init s) | w. True}  
**begin**

**abbreviation** *Reachable* s  $\equiv$  {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** –

**let** ?D = post ' Reachable r  $\times$  post ' Reachable s  
**let** ?X =  $\lambda ps.$  ?D – map-prod post post ' set ps  
**let** ?f =  $\lambda(ws, ps, N).$  card (?X ps)

**have**  $\exists st.$  ?cl = Some st **unfolding** *closure-invariant-def*

**proof** (rule measure-while-option-Some[of invariant ?r ?s - - ?f], intro conjI)

**fix** st **assume** base: invariant ?r ?s st **and** test-invariant st

**hence** step: invariant ?r ?s (step-invariant st) **by** (rule invariant)

**obtain** ws ps N **where** st: st = (ws, ps, N) **by** (cases st) blast

**hence** finite (?X ps) **by** (blast intro: finite-cartesian-product fin)

**moreover** obtain ws' ps' N' **where** step-invariant: step-invariant (ws, ps, N)

= (ws', ps', N')

**by** (cases step-invariant (ws, ps, N)) blast

```

moreover
  { have map-prod post post ' set ps  $\subseteq$  ?D using base[unfolded st invariant-def]
by fast
  moreover
    have map-prod post post ' set ps'  $\subseteq$  ?D using step[unfolded st step-invariant invariant-def]
    by fast
  moreover
    { have distinct (map snd ws @ ps) inj-on (map-prod post post) (set (map snd ws @ ps))
      using base[unfolded st invariant-def] by (auto simp: bij-betw-def)
      hence distinct (map (map-prod post post) (map snd ws @ ps)) unfolding distinct-map ..
      hence map-prod post post ' set ps  $\subset$  map-prod post post ' set (snd (hd ws) # ps)
        using (test-invariant st) st by (cases ws) (simp-all, blast)
      moreover have map-prod post post ' set ps' = map-prod post post ' set (snd (hd ws) # ps)
        using step-invariant by (auto split: prod.splits)
      ultimately have map-prod post post ' set ps  $\subset$  map-prod post post ' set ps'
by simp
    }
    ultimately have ?X ps'  $\subset$  ?X ps by (auto simp add: image-set simp del: set-map)
  }
  ultimately show ?f (step-invariant st) < ?f st unfolding st step-invariant
    using psubset-card-mono[of ?X ps ?X ps'] by simp
  qed (auto simp add: invariant-start WF invariant)
  then show False using result by auto
qed

```

**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
using trans[OF closure-invariant-closure[of ([[init r, init s]], simplified] result)]
by (auto intro: closure-invariant-termination[OF WF])

```

**lemma** *closure-invariant-complete:*

```

assumes eq: lang n r = lang n s
and WF: wf n r wf n s
shows  $\exists ps N. \text{closure-invariant} ([[init r, init s]], [], \{(post (init r), post (init s))\}) =$ 
   $\text{Some}([], ps, N) \text{ (is } \exists - . \text{closure-invariant} ([[init r, ?r], ?s], -) = - \text{ is } \exists - . ?cl = -)$ 
proof (cases ?cl)
  case (Some st)
  moreover obtain ws ps N where ws-ps-N: st = (ws, ps, N) by (cases st) blast
  ultimately show ?thesis
proof (cases ws)

```

```

    case (Cons wrs ws)
      with Some obtain w where counterexample r s = Some w unfolding
counterexample-def
      by (cases wrs) (auto simp: ws-ps-N)
      with eq counterexample-sound[OF - WF] show ?thesis by blast
    qed blast
  qed (auto intro: closure-invariant-termination[OF WF])

```

```

lemma closure-complete:
  assumes lang n r = lang n s wf n r wf n s
  shows  $\exists N. \text{closure } ([(\text{init } r, \text{init } s)], \{(\text{post } (\text{init } r), \text{post } (\text{init } s)))\} = \text{Some } ([, N)$ 
  using closure-invariant-complete[OF assms]
  by (subst closure-invariant-closure[of [([], init r, init s), simplified, symmetric])
  auto

```

```

lemma check-eqv-complete:
  assumes lang n r = lang n s wf n r wf n s
  shows check-eqv r s
  using closure-complete[OF assms] by (auto simp: check-eqv-def)

```

```

lemma counterexample-complete:
  assumes lang n r  $\neq$  lang n s and WF: wf n r wf n s
  shows  $\exists w. \text{counterexample } r s = \text{Some } w$ 
  using closure-invariant-sound[OF - WF] closure-invariant-termination[OF WF]
  assms
  by (fastforce simp: counterexample-def Let-def split: option.splits list.splits)

```

end

```

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

```

```

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

```

```

in (tl ws @ new, set new ∪ N))
by (force simp: Let-def map-apfst-remdups' filter-map o-def split: prod.splits)

end

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

sublocale rexp-DA-no-post by unfold-locales

end

locale rexp-DA-sim = project set o σ wf-atom project lookup
  for σ :: nat ⇒ 'a list
  and wf-atom :: nat ⇒ 'b :: linorder ⇒ bool
  and project :: 'a ⇒ 'a
  and lookup :: 'b ⇒ 'a ⇒ bool +
  fixes init :: 'b rexp ⇒ 's
  fixes sim-delta :: 's ⇒ 's list
  fixes final :: 's ⇒ bool
  fixes wf-state :: 's ⇒ 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)

```

```

    by (metis L-sim-delta length-map)

lemma L-delta[simp]:  $\llbracket a \in \text{set } (\sigma \ n); \text{wf-state } s \rrbracket \implies L (\text{delta } a \ s) = \text{lQuot } a \ (L \ s)$ 
  using L-sim-delta[of s] unfolding delta-def in-set-conv-nth
  by (subst (asm) list-eq-iff-nth-eq) auto

lemma delta-wf-state[simp]:  $\llbracket a \in \text{set } (\sigma \ n); \text{wf-state } s \rrbracket \implies \text{wf-state } (\text{delta } a \ s)$ 
  unfolding delta-def by (auto intro: bspec[OF sim-delta-wf-state nth-mem])

sublocale rexp-DA  $\sigma$  wf-atom project lookup init delta final wf-state post L
  by unfold-locales auto

sublocale rexp-DA-sim-no-post: rexp-DA-no-post  $\sigma$  wf-atom project lookup init
  delta final wf-state L
  by unfold-locales auto

end

```

## 7 Initial Normalization of the Input

```

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

lemma toplevel-inters-nonempty[simp]:
  toplevel-inters r  $\neq$  {}
  by (induct r) auto

lemma toplevel-inters-finite[simp]:
  finite (toplevel-inters r)
  by (induct r) auto

context alphabet
begin

lemma toplevel-inters-wf:
  wf n s = ( $\forall r \in \text{toplevel-inters } s. \text{wf } n \ r$ )
  by (induct s) auto

end

context project
begin

lemma toplevel-inters-lang:

```

$r \in \text{toplevel-inters } s \implies \text{lang } n \ s \subseteq \text{lang } n \ r$   
**by** (induct  $s$ ) auto

**lemma** *toplevel-inters-lang-INT*:  
 $\text{lang } n \ s = (\bigcap r \in \text{toplevel-inters } s. \text{lang } n \ r)$   
**by** (induct  $s$ ) auto

**lemma** *toplevel-inters-in-lang*:  
 $w \in \text{lang } n \ s = (\forall r \in \text{toplevel-inters } s. w \in \text{lang } n \ r)$   
**by** (induct  $s$ ) auto

**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*  $(\forall r \in X. w \in \text{lang } n \ r)$ *)*  
**unfolding** *lang-INTERSECT* **by** auto

**end**

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

**lemma** *set-merge-distinct[simp]*:  $\text{set } (\text{merge-distinct } xs \ ys) = \text{set } xs \cup \text{set } ys$   
**by** (induct  $xs \ ys$  rule: *merge-distinct.induct*) auto

**lemma** *sorted-merge-distinct[simp]*:  $\llbracket \text{sorted } xs; \text{sorted } ys \rrbracket \implies \text{sorted } (\text{merge-distinct } xs \ ys)$   
**by** (induct  $xs \ ys$  rule: *merge-distinct.induct*) (auto)

**lemma** *distinct-merge-distinct[simp]*:  $\llbracket \text{sorted } xs; \text{distinct } xs; \text{sorted } ys; \text{distinct } ys \rrbracket$   
 $\implies$   
 $\text{distinct } (\text{merge-distinct } xs \ ys)$   
**by** (induct  $xs \ ys$  rule: *merge-distinct.induct*) (auto)

**lemma** *sorted-list-of-set-merge-distinct[simp]*:  $\llbracket \text{sorted } xs; \text{distinct } xs; \text{sorted } ys; \text{distinct } ys \rrbracket \implies$   
 $\text{merge-distinct } xs \ ys = \text{sorted-list-of-set } (\text{set } xs \cup \text{set } ys)$   
**by** (auto intro: *sorted-distinct-set-unique*)

**fun** *zip-with-option* **where**  
 $\text{zip-with-option } f \ (\text{Some } a) \ (\text{Some } b) = \text{Some } (f \ a \ b)$   
 $| \text{zip-with-option } - \ - \ - = \text{None}$

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

b)

**by** (*induct f x y rule: zip-with-option.induct*) *auto*

**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*  
**by** (*induct r*) *auto*

**lemma** *Pluss-Some*: *Pluss r = Some xs*  $\longleftrightarrow$

(*Full*  $\notin$  *set xs*  $\wedge$  *xs* = *sorted-list-of-set* (*toplevel-summands r* - {*Zero*}))

**proof** (*induct r arbitrary: xs*)

**case** (*Plus r s*)

**show** ?*case*

**proof** *safe*

**assume** *Pluss* (*Plus r s*) = *Some xs*

**then obtain** *a b* **where** \*: *Pluss r = Some a Pluss s = Some b xs = merge-distinct a b* **by** *auto*

**with** *Plus(1)[of a] Plus(2)[of b]*

**show** *xs* = *sorted-list-of-set* (*toplevel-summands* (*Plus r s*) - {*Zero*}) **by** (*simp add: Un-Diff*)

**assume** *Full*  $\in$  *set xs* **with** *Plus(1)[of a] Plus(2)[of b]* \* **show** *False* **by** (*simp add: Pluss-None*)

**next**

**assume** *Full*  $\notin$  *set* (*sorted-list-of-set* (*toplevel-summands* (*Plus r s*) - {*Zero*}))

**with** *Plus(1)[of sorted-list-of-set* (*toplevel-summands r* - {*Zero*})]

*Plus(2)[of sorted-list-of-set* (*toplevel-summands s* - {*Zero*})]

**show** *Pluss* (*Plus r s*) = *Some* (*sorted-list-of-set* (*toplevel-summands* (*Plus r s*) - {*Zero*}))

**by** (*simp add: Un-Diff*)

**qed**

**qed** *force+*

**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*  
**by** (*induct r*) *auto*

**lemma** *Inters-Some*: *Inters r = Some xs*  $\longleftrightarrow$

(*Zero*  $\notin$  *set xs*  $\wedge$  *xs* = *sorted-list-of-set* (*toplevel-inters r* - {*Full*}))

**proof** (*induct r arbitrary: xs*)

**case** (*Inter r s*)

```

show ?case
proof safe
  assume Inters (Inter r s) = Some xs
  then obtain a b where *: Inters r = Some a Inters s = Some b xs =
merge-distinct a b by auto
  with Inter(1)[of a] Inter(2)[of b]
  show xs = sorted-list-of-set (toplevel-inters (Inter r s) - {Full}) by (simp
add: Un-Diff)
  assume Zero ∈ set xs with Inter(1)[of a] Inter(2)[of b] * show False by
(simp add: Inters-None)
  next
  assume Zero ∉ set (sorted-list-of-set (toplevel-inters (Inter r s) - {Full}))
  with Inter(1)[of sorted-list-of-set (toplevel-inters r - {Full})]
  Inter(2)[of sorted-list-of-set (toplevel-inters s - {Full})]
  show Inters (Inter r s) = Some (sorted-list-of-set (toplevel-inters (Inter r s)
- {Full}))
  by (simp add: Un-Diff)
qed
qed force+

```

**definition** *inPlus* **where**

*inPlus* r s = (case Pluss (Plus r s) of None ⇒ Full | Some rs ⇒ PLUS rs)

**lemma** *inPlus-alt*: *inPlus* r s = (let X = topLevel-summands (Plus r s) - {Zero} in

flatten PLUS (if Full ∈ X then {Full} else X))

**proof** (cases Pluss r Pluss s rule: option.exhaust[case-product option.exhaust])

**case** Some-Some **then show** ?thesis **by** (simp add: *inPlus-def* Pluss-None) (simp add: Pluss-Some Un-Diff)

**qed** (simp-all add: *inPlus-def* Pluss-None)

**fun** *inTimes* **where**

*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**

*inStar* Zero = One

| *inStar* Full = Full

| *inStar* One = One

| *inStar* (Star r) = Star r

| *inStar* r = Star r

**definition** *inInter* **where**

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



**lemma** *inInter-alt*:  $\text{inInter } r \ s = (\text{let } X = \text{toplevel-inters } (\text{Inter } r \ s) - \{\text{Full}\} \text{ in } \text{flatten INTERSECT } (\text{if } \text{Zero} \in X \text{ then } \{\text{Zero}\} \text{ else } X))$   
**proof** (*cases Inters r Inters s rule: option.exhaust[case-product option.exhaust]*)  
**case** *Some-Some* **then show** *?thesis* **by** (*simp add: inInter-def Inters-None*)  
(*simp add: Inters-Some Un-Diff*)  
**qed** (*simp-all add: inInter-def Inters-None*)

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

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

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

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

**lemma** *wf-inPlus[simp]*:  $\llbracket \text{wf } n \ r; \text{wf } n \ s \rrbracket \implies \text{wf } n \ (\text{inPlus } r \ s)$   
**by** (*subst (asm) (1 2) toplevel-summands-wf*) (*auto simp: inPlus-alt*)

**lemma** *wf-inTimes[simp]*:  $\llbracket \text{wf } n \ r; \text{wf } n \ s \rrbracket \implies \text{wf } n \ (\text{inTimes } r \ s)$   
**by** (*induct r s rule: inTimes.induct*) *auto*

**lemma** *wf-inStar[simp]*:  $\text{wf } n \ r \implies \text{wf } n \ (\text{inStar } r)$   
**by** (*induct r rule: inStar.induct*) *auto*

**lemma** *wf-inInter[simp]*:  $\llbracket \text{wf } n \ r; \text{wf } n \ s \rrbracket \implies \text{wf } n \ (\text{inInter } r \ s)$   
**by** (*subst (asm) (1 2) toplevel-inters-wf*) (*auto simp: inInter-alt*)

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

**by** (*induct r rule: inNot.induct*) *auto*

**lemma** *wf-inPr[simp]*:  $wf (Suc\ n)\ r \implies wf\ n\ (inPr\ r)$   
**by** (*induct r rule: inPr.induct*) *auto*

**lemma** *wf-inorm[simp]*:  $wf\ n\ r \implies wf\ n\ (inorm\ r)$   
**by** (*induct r arbitrary: n*) *auto*

**end**

**context** *project begin*

**lemma** *lang-inPlus[simp]*:  $\llbracket wf\ n\ r; wf\ n\ s \rrbracket \implies lang\ n\ (inPlus\ r\ s) = lang\ n\ (Plus\ r\ s)$   
**by** (*auto 0 3 simp: inPlus-alt toplevel-summands-in-lang[of - n r] toplevel-summands-in-lang[of - n s]*)  
*dest: lang-subset-lists intro: beXI[of - Full]*

**lemma** *lang-inTimes[simp]*:  $\llbracket wf\ n\ r; wf\ n\ s \rrbracket \implies lang\ n\ (inTimes\ r\ s) = lang\ n\ (Times\ r\ s)$   
**by** (*induct r s rule: inTimes.induct*) (*auto simp: conc-assoc*)

**lemma** *lang-inStar[simp]*:  $wf\ n\ r \implies lang\ n\ (inStar\ r) = lang\ n\ (Star\ r)$   
**by** (*induct r rule: inStar.induct*)  
*(auto intro: star-if-lang dest: subsetD[OF star-subset-lists, rotated])*

**lemma** *Zero-toplevel-inters[dest]*:  $Zero \in toplevel-inters\ r \implies lang\ n\ r = \{\}$   
**by** (*metis lang.simps(1) subset-empty toplevel-inters-lang*)

**lemma** *toplevel-inters-Full*:  $\llbracket toplevel-inters\ r = \{Full\}; wf\ n\ r \rrbracket \implies lang\ n\ r = lists\ (\Sigma\ n)$   
**by** (*metis antisym lang.simps(2) subsetI toplevel-inters.simps(3) toplevel-inters-in-lang*)

**lemma** *toplevel-inters-subset-singleton[simp]*:  $toplevel-inters\ r \subseteq \{s\} \iff toplevel-inters\ r = \{s\}$   
**by** (*metis subset-refl subset-singletonD toplevel-inters-nonempty*)

**lemma** *lang-inInter[simp]*:  $\llbracket wf\ n\ r; wf\ n\ s \rrbracket \implies lang\ n\ (inInter\ r\ s) = lang\ n\ (Inter\ r\ s)$   
**using** *lang-subset-lists[of n, unfolded lang.simps(2)[symmetric]]*  
*toplevel-inters-nonempty[of r] toplevel-inters-nonempty[of s]*  
**apply** (*auto 0 2 simp: inInter-alt toplevel-inters-in-lang[of - n r] toplevel-inters-in-lang[of - n s]*)  
*toplevel-inters-wf[of n r] toplevel-inters-wf[of n s] Ball-def simp del: toplevel-inters-nonempty*  
*dest!: toplevel-inters-Full[of - n] split: if-splits*  
**by** *fastforce+*

**lemma** *lang-inNot[simp]*:  $wf\ n\ r \implies lang\ n\ (inNot\ r) = lang\ n\ (Not\ r)$   
**by** (*induct r rule: inNot.induct*) (*auto dest: lang-subset-lists*)

**lemma** *lang-inPr[simp]*:  $wf (Suc\ n)\ r \implies lang\ n\ (inPr\ r) = lang\ n\ (Pr\ r)$   
**by** (*induct r rule: inPr.induct*) *auto*

**lemma** *lang-inorm[simp]*:  $wf\ n\ r \implies lang\ n\ (inorm\ r) = lang\ n\ r$   
**by** (*induct r arbitrary: n*) *auto*

**end**

## 8 Partial Derivatives-like Normalization

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

**lemma** (**in** *alphabet*) *wf-pnPlus[simp]*:  $\llbracket wf\ n\ r; wf\ n\ s \rrbracket \implies wf\ n\ (pnPlus\ r\ s)$   
**by** (*induct r s rule: pnPlus.induct*) *auto*

**lemma** (**in** *project*) *lang-pnPlus[simp]*:  $\llbracket wf\ n\ r; wf\ n\ s \rrbracket \implies lang\ n\ (pnPlus\ r\ s)$   
=  $lang\ n\ (Plus\ r\ s)$   
**by** (*induct r s rule: pnPlus.induct*) (*auto dest!: lang-subset-lists dest: project*  
*subsetD[OF star-subset-lists, unfolded in-lists-conv-set, rotated -1]*  
*subsetD[OF conc-subset-lists, unfolded in-lists-conv-set, rotated -1]*)

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

**lemma** (**in** *alphabet*) *wf-pnTimes[simp]*:  $\llbracket wf\ n\ r; wf\ n\ s \rrbracket \implies wf\ n\ (pnTimes\ r\ s)$   
**by** (*induct r s rule: pnTimes.induct*) *auto*

**lemma** (**in** *project*) *lang-pnTimes[simp]*:  $\llbracket wf\ n\ r; wf\ n\ s \rrbracket \implies lang\ n\ (pnTimes\ r\ s)$   
=  $lang\ n\ (Times\ r\ s)$   
**by** (*induct r s rule: pnTimes.induct*) *auto*

**fun** *pnInter* :: '*a*::*linorder rexp*  $\Rightarrow$  '*a rexp*  $\Rightarrow$  '*a rexp* **where**  
*pnInter Zero* *r* = *Zero*

```

| pnInter r Zero = Zero
| pnInter Full r = r
| pnInter r Full = r
| pnInter (Plus r s) t = pnPlus (pnInter r t) (pnInter s t)
| pnInter r (Plus s t) = pnPlus (pnInter r s) (pnInter r t)
| pnInter (Inter r s) t = pnInter r (pnInter s t)
| pnInter r (Inter s t) =
  (if r = s then Inter s t
   else if r ≤ s then Inter r (Inter s t)
   else Inter s (pnInter r t))
| pnInter r s =
  (if r = s then s
   else if r ≤ s then Inter r s
   else Inter s r)

```

**lemma** (in *alphabet*) *wf-pnInter[simp]*:  $\llbracket wf\ n\ r; wf\ n\ s \rrbracket \implies wf\ n\ (pnInter\ r\ s)$   
**by** (*induct r s rule: pnInter.induct*) *auto*

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

**by** (*induct r s rule: pnInter.induct*) (*auto, auto dest!: lang-subset-lists dest:*  
*project*

*subsetD[OF star-subset-lists, unfolded in-lists-conv-set, rotated -1]*  
*subsetD[OF conc-subset-lists, unfolded in-lists-conv-set, rotated -1]*)

**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]*:  $wf\ n\ r \implies wf\ n\ (pnNot\ r)$   
**by** (*induct r rule: pnNot.induct*) *auto*

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

**by** (*induct r rule: pnNot.induct*) (*auto dest: lang-subset-lists*)

**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-pnPr[simp]*:  $wf\ (Suc\ n)\ r \implies wf\ n\ (pnPr\ r)$   
**by** (*induct r rule: pnPr.induct*) *auto*

**lemma** (in project) lang-pnPr[simp]: wf (Suc n) r  $\implies$  lang n (pnPr r) = lang n (Pr r)

by (induct r rule: pnPr.induct) auto

**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-pnorm[simp]: wf n r  $\implies$  wf n (pnorm r)

by (induct r arbitrary: n) auto

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

by (induct r arbitrary: n) auto

## 9 Monadic Second-Order Logic Formulas

### 9.1 Interpretations and Encodings

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

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

**abbreviation** enc-atom I n a  $\equiv$  (a, enc-atom-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 φ) = FOV φ
| FOV (FOr φ1 φ2) = FOV φ1 ∪ FOV φ2
| FOV (FAnd φ1 φ2) = FOV φ1 ∩ FOV φ2
| FOV (FExists φ) = (λx. x - 1) ‘ (FOV φ - {0})
| FOV (FEXISTS φ) = (λx. x - 1) ‘ FOV φ

```

**primrec** SOV :: 'a formula ⇒ nat set **where**

```

SOV (FQ a m) = {}
| SOV (FLess m1 m2) = {}
| SOV (FIn m M) = {M}
| SOV (FNot φ) = SOV φ
| SOV (FOr φ1 φ2) = SOV φ1 ∪ SOV φ2
| SOV (FAnd φ1 φ2) = SOV φ1 ∩ SOV φ2
| SOV (FExists φ) = (λx. x - 1) ‘ SOV φ
| SOV (FEXISTS φ) = (λx. x - 1) ‘ (SOV φ - {0})

```

**definition** σ = (λΣ n. concat (map (λbs. map (λa. (a, bs)) Σ) (List.n-lists n [True, False])))

**definition** π = (λ(a, bs). (a, tl bs))

**definition** ε = (λΣ (a::'a, bs). if a ∈ set Σ then [(a, True # bs), (a, False # bs)] else [])

**datatype** 'a atom =

```

Singleton 'a bool list
| AQ nat 'a
| Arbitrary-Except nat bool
| Arbitrary-Except2 nat nat

```

**derive** linorder atom

**fun** wf-atom **where**

```

wf-atom Σ n (Singleton a bs) = (a ∈ set Σ ∧ length bs = n)
| wf-atom Σ n (AQ m a) = (a ∈ set Σ ∧ m < n)
| wf-atom Σ n (Arbitrary-Except m -) = (m < n)
| wf-atom Σ n (Arbitrary-Except2 m1 m2) = (m1 < n ∧ m2 < n)

```

**fun** lookup **where**

```

lookup (Singleton a' bs') (a, bs) = (a = a' ∧ bs = bs')
| lookup (AQ m a') (a, bs) = (a = a' ∧ bs ! m)
| lookup (Arbitrary-Except m b) (-, bs) = (bs ! m = b)
| lookup (Arbitrary-Except2 m1 m2) (-, bs) = (bs ! m1 ∧ bs ! m2)

```

**lemma** π-σ: π ‘ (set o σ Σ) (n + 1) = (set o σ Σ) n

**unfolding** π-def σ-def set-map[symmetric] o-apply map-concat **by** auto

**locale** formula = embed2 set o (σ Σ) wf-atom Σ π lookup ε Σ case-prod Singleton

**for** Σ :: 'a :: linorder list +

**assumes** nonempty: Σ ≠ []

**begin**

**abbreviation**  $\Sigma$ -product-lists  $n \equiv$   
*List.maps* ( $\lambda$ bools. *map* ( $\lambda a.$  ( $a$ , 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*  $n$   $\varphi \implies \forall p \in \text{FOV } \varphi \cup \text{SOV } \varphi. p < n$   
**by** (*induct*  $\varphi$  *arbitrary*:  $n$ )  
(*auto split*: *if-split-asm*, (*metis Un-iff diff-Suc-less less-SucE less-imp-diff-less*)+)

**lemma** *finite-FOV*: *finite* (*FOV*  $\varphi$ )  
**by** (*induct*  $\varphi$ ) (*auto split*: *if-split-asm*)

### 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*  $n$   $p$ )  
**unfolding** *valid-ENC-def* **by** *auto*

**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*  $n$   $V$ )  
**unfolding** *ENC-def*  
**by** (*intro iffD2[OF wf-flatten-INTERSECT]*) (*auto simp*: *wf-rexp-valid-ENC*)

**lemma** *enc-atom- $\sigma$ -eq*:  $i < \text{length } w \implies$

$(\text{length } I = n \wedge p \in \text{set } \Sigma) \longleftrightarrow \text{enc-atom } I \text{ } i \text{ } p \in \text{set } (\sigma \Sigma n)$   
**by** (*auto simp:  $\sigma$ -def set-n-lists intro!: exI[of - enc-atom-bool I i] imageI*)

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

**lemma** *enc-atom-bool-take-drop-True:*

$\llbracket r < \text{length } I; \text{ case } I ! r \text{ of } \text{Inl } p' \Rightarrow p = p' \mid \text{Inr } P \Rightarrow p \in P \rrbracket \Longrightarrow$   
 $\text{enc-atom-bool } I \text{ } p = \text{take } r \text{ (enc-atom-bool } I \text{ } p) \text{ @ True \# drop (Suc } r)$   
*(enc-atom-bool I p)*  
**by** (*auto intro: trans[OF id-take-nth-drop]*)

**lemma** *enc-atom-bool-take-drop-True2:*

$\llbracket r < \text{length } I; \text{ case } I ! r \text{ of } \text{Inl } p' \Rightarrow p = p' \mid \text{Inr } P \Rightarrow p \in P;$   
 $s < \text{length } I; \text{ case } I ! s \text{ of } \text{Inl } p' \Rightarrow p = p' \mid \text{Inr } P \Rightarrow p \in P; r < s \rrbracket \Longrightarrow$   
 $\text{enc-atom-bool } I \text{ } p = \text{take } r \text{ (enc-atom-bool } I \text{ } p) \text{ @ True \#}$   
 $\text{take (s - Suc } r) \text{ (drop (Suc } r) \text{ (enc-atom-bool } I \text{ } p)) @ True \#}$   
 $\text{drop (Suc } s) \text{ (enc-atom-bool } I \text{ } p)$   
**using** *id-take-nth-drop[of r enc-atom-bool I p]*  
*id-take-nth-drop[of s - r - 1 drop (Suc r) (enc-atom-bool I p)]*  
**by** *auto*

**lemma** *enc-atom-bool-take-drop-False:*

$\llbracket r < \text{length } I; \text{ case } I ! r \text{ of } \text{Inl } p' \Rightarrow p \neq p' \mid \text{Inr } P \Rightarrow p \notin P \rrbracket \Longrightarrow$   
 $\text{enc-atom-bool } I \text{ } p = \text{take } r \text{ (enc-atom-bool } I \text{ } p) \text{ @ False \# drop (Suc } r)$   
*(enc-atom-bool I p)*  
**by** (*auto intro: trans[OF id-take-nth-drop] split: sum.splits*)

**lemma** *enc-atom-lang-AQ:  $\llbracket r < \text{length } I;$*

*case I ! r of Inl p'  $\Rightarrow$  p = p' | Inr P  $\Rightarrow$  p  $\in$  P; length I = n; a  $\in$  set  $\Sigma$   $\rrbracket \Longrightarrow$*   
 $[\text{enc-atom } I \text{ } p \text{ } a] \in \text{lang } n \text{ (Atom (AQ } r \text{ } a))$   
**unfolding** *lang.simps*  
**by** (*force intro!: enc-atom-bool-take-drop-True*  
*image-eqI[of - - ( $\lambda J. \text{take } r \text{ } J \text{ @ drop (r + 1) } J)$  (enc-atom-bool I p)]*  
*simp:  $\sigma$ -def set-n-lists*)

**lemma** *enc-atom-lang-Arbitrary-Except-True:  $\llbracket r < \text{length } I;$*

*case I ! r of Inl p'  $\Rightarrow$  p = p' | Inr P  $\Rightarrow$  p  $\in$  P; length I = n; a  $\in$  set  $\Sigma$   $\rrbracket \Longrightarrow$*   
 $[\text{enc-atom } I \text{ } p \text{ } a] \in \text{lang } n \text{ (Atom (Arbitrary-Except } r \text{ True))}$   
**unfolding** *lang.simps*  
**by** (*force intro!: enc-atom-bool-take-drop-True*  
*image-eqI[of - - ( $\lambda J. \text{take } r \text{ } J \text{ @ drop (r + 1) } J)$  (enc-atom-bool I p)]*  
*simp:  $\sigma$ -def set-n-lists*)

**lemma** *enc-atom-lang-Arbitrary-Except2:  $\llbracket r < \text{length } I; s < \text{length } I;$*

*case I ! r of Inl p'  $\Rightarrow$  p = p' | Inr P  $\Rightarrow$  p  $\in$  P;*  
*case I ! s of Inl p'  $\Rightarrow$  p = p' | Inr P  $\Rightarrow$  p  $\in$  P; length I = n; a  $\in$  set  $\Sigma$   $\rrbracket \Longrightarrow$*   
 $[\text{enc-atom } I \text{ } p \text{ } a] \in \text{lang } n \text{ (Atom (Arbitrary-Except2 } r \text{ } s))$   
**unfolding** *lang.simps*  
**by** (*force intro!: enc-atom-bool-take-drop-True2*)



*simp: set-n-lists  $\sigma$ -def take-Cons' drop-Cons' numeral-eq-Suc)*

**lemma** *enc-atom-lang-Arbitrary-Except-False*:  $\llbracket r < \text{length } I;$   
*case*  $I ! r$  of *Inl*  $p' \Rightarrow p \neq p' \mid$  *Inr*  $P \Rightarrow p \notin P; \text{length } I = n; a \in \text{set } \Sigma \rrbracket \Longrightarrow$   
 $\llbracket \text{enc-atom } I \ p \ a \rrbracket \in \text{lang } n \ (\text{Atom } (\text{Arbitrary-Except } r \ \text{False}))$   
**unfolding** *lang.simps*  
**by** (*force intro!*: *enc-atom-bool-take-drop-False*  
*image-eqI*[*of - -* ( $\lambda J. \text{take } r \ J \ @ \ \text{drop } (r + 1) \ J$ ) (*enc-atom-bool I p*)]  
*simp: set-n-lists  $\sigma$ -def split: sum.splits*)

**lemma** *AQ-D*:  
**assumes**  $v \in \text{lang } n \ (\text{Atom } (\text{AQ } m \ a)) \ m < n \ a \in \text{set } \Sigma$   
**shows**  $\exists x. v = [x] \wedge \text{fst } x = a \wedge \text{snd } x ! m$   
**using** *assms by auto*

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

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

**lemma** *star-Arbitrary-ExceptD*:  
 $\llbracket v \in \text{star } (\text{lang } n \ (\text{Atom } (\text{Arbitrary-Except } r \ b))); \ r < n; \ i < \text{length } v \rrbracket \Longrightarrow$   
 $\text{snd } (v ! i) ! r = b$   
**proof** (*induct arbitrary: i rule: star-induct*)  
**case** (*append u v*) **thus** *?case by* (*cases i*) (*auto dest: Arbitrary-ExceptD*)  
**qed** *simp*

**end**

**end**

## 10 M2L

### 10.1 Encodings

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

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

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

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

*wf-interp-for-formula* (w, I)  $\varphi$  =  
 (wf-interp w I  $\wedge$   
 ( $\forall n \in FOV \varphi$ . case I ! n of Inl -  $\Rightarrow$  True | -  $\Rightarrow$  False)  $\wedge$   
 ( $\forall n \in SOV \varphi$ . case I ! n of Inl -  $\Rightarrow$  False | -  $\Rightarrow$  True))

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

(w, I)  $\models$  FQ a m = (w ! (case I ! m of Inl p  $\Rightarrow$  p) = a)  
 | (w, I)  $\models$  FLess m1 m2 = ((case I ! m1 of Inl p  $\Rightarrow$  p) < (case I ! m2 of Inl p  $\Rightarrow$  p))  
 | (w, I)  $\models$  FIn m M = ((case I ! m of Inl p  $\Rightarrow$  p)  $\in$  (case I ! M 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<sub>M2L</sub>* :: nat  $\Rightarrow$  'a formula  $\Rightarrow$  ('a  $\times$  bool list) list set **where**

*lang<sub>M2L</sub>* n  $\varphi$  = {enc (w, I) | w I.  
 length I = n  $\wedge$  wf-interp-for-formula (w, I)  $\varphi \wedge$  satisfies (w, I)  $\varphi$ }

**definition** *dec-word*  $\equiv$  map fst

**definition** *positions-in-row* w i =

Option.these (set (map-index ( $\lambda p$  a-bs. if nth (snd a-bs) i then Some p else None)  
 w))

**definition** *dec-interp* n FO (w :: ('a  $\times$  bool list) list)  $\equiv$  map ( $\lambda i$ .

if i  $\in$  FO  
 then Inl (the-elem (positions-in-row w i))  
 else Inr (positions-in-row w i)) [0.. $n$ ]

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

**unfolding** *positions-in-row-def* Option.these-def **by** (auto intro!: image-eqI[of - the])

**lemma** *positions-in-row-unique*:  $\exists! p$ . p < length w  $\wedge$  snd (w ! p) ! i  $\implies$   
 the-elem (positions-in-row w i) = (THE p. p < length w  $\wedge$  snd (w ! p) ! i)  
**by** (rule the1I2) (auto simp: the-elem-def positions-in-row)

**lemma** *positions-in-row-length*:  $\exists! p$ . p < length w  $\wedge$  snd (w ! p) ! i  $\implies$   
 the-elem (positions-in-row w i) < length w  
**unfolding** *positions-in-row-unique* **by** (rule the1I2) auto

**lemma** *dec-interp-Inl*:  $\llbracket i \in FO; i < n \rrbracket \implies \exists p$ . dec-interp n FO x ! i = Inl p  
**unfolding** *dec-interp-def* **using** nth-map[of n [0.. $n$ ]] **by** auto

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

**unfolding** *dec-interp-def* **using** *nth-map*[of  $n$   $[0..<n]$ ] **by** *auto*

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

**unfolding** *dec-interp-def* **using** *nth-map*[of  $n$   $[0..<n]$ ] **by** *auto*

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

**unfolding** *dec-interp-def* **using** *nth-map*[of  $n$   $[0..<n]$ ] **by** *auto*

**lemma** *Inl-dec-interp-length*:

**assumes**  $\forall i \in \text{FO}. \exists ! p. p < \text{length } w \wedge \text{snd } (w ! p) ! i$

**shows**  $\text{Inl } p \in \text{set } (\text{dec-interp } n \text{ FO } w) \implies p < \text{length } w$

**unfolding** *dec-interp-def* **by** (*auto intro: positions-in-row-length*[*OF bspec*[*OF assms*]])

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

**unfolding** *dec-interp-def* **by** (*auto simp: positions-in-row*)

**lemma** *the-elem-Collect*[*simp*]:

**assumes**  $\exists ! x. P x$

**shows** *the-elem* (*Collect*  $P$ ) = (*The*  $P$ )

**proof** (*unfold the-elem-def, rule the-equality*)

**from** *assms* **have**  $P$  (*The*  $P$ ) **by** (*rule theI'*)

**with** *assms* **show** *Collect*  $P$  = {*The*  $P$ } **by** *auto*

**fix**  $x$  **assume** *Collect*  $P$  = { $x$ }

**then show**  $x$  = *The*  $P$  **by** (*auto intro: the-equality*[*symmetric*])

**qed**

**lemma** *enc-atom-dec*:

$\llbracket \text{wf-word } n \text{ } w; \forall i \in \text{FO}. i < n \longrightarrow (\exists ! p. p < \text{length } w \wedge \text{snd } (w ! p) ! i); p < \text{length } w \rrbracket \implies$

*enc-atom* (*dec-interp*  $n$  *FO*  $w$ )  $p$  (*fst* ( $w ! p$ )) =  $w ! p$

**unfolding** *wf-word dec-interp-def map-filter-def Let-def*

**by** (*auto 0 4 simp: comp-def  $\sigma$ -def set-n-lists positions-in-row dest: nth-mem*[of  $p$   $w$ ])

*intro!*: *trans*[*OF iffD2*[*OF prod.inject*] *prod.collapse*] *nth-equalityI the-equality*[*symmetric*]

*intro: the1I2*[of  $\lambda p. p < \text{length } w \wedge \text{snd } (w ! p) ! i \lambda p. \text{snd } (w ! p) ! i$  **for**  $i$ ]

*elim!*: *contrapos-np*[of - *False*])

**lemma** *enc-dec*:

$\llbracket \text{wf-word } n \text{ } w; \forall i \in \text{FO}. i < n \longrightarrow (\exists ! p. p < \text{length } w \wedge \text{snd } (w ! p) ! i) \rrbracket \implies$

*enc* (*dec-word*  $w$ , *dec-interp*  $n$  *FO*  $w$ ) =  $w$

**unfolding** *enc.simps dec-word-def*

by (intro trans[OF map-index-map map-index-congL] allI impI enc-atom-dec)  
assumption+

**lemma** *dec-word-enc*:  $\text{dec-word } (\text{enc } (w, I)) = w$   
unfolding *dec-word-def* by auto

**lemma** *enc-unique*:

assumes *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 (erule *exE*)  
fix  $p$  assume  $I ! i = \text{Inl } p$   
with *assms* show ?thesis by (auto simp: map-index all-set-conv-all-nth intro!: *exI*[of  $p$ ])  
qed

**lemma** *dec-interp-enc-Inl*:

$\llbracket \text{dec-interp } n \text{ FO } (\text{enc } (w, I)) ! i = \text{Inl } p'; I ! i = \text{Inl } p; i \in \text{FO}; i < n; \text{length } I = n; p < \text{length } w; \text{wf-interp } w I \rrbracket \implies p = p'$   
unfolding *dec-interp-def* using *nth-map*[of  $-\ [0..<n]$ ] *positions-in-row-unique*[OF *enc-unique*]  
by (auto intro: *sym*[OF *the-equality*])

**lemma** *dec-interp-enc-Inr*:

$\llbracket \text{dec-interp } n \text{ FO } (\text{enc } (w, I)) ! i = \text{Inr } P'; I ! i = \text{Inr } P; i \notin \text{FO}; i < n; \text{length } I = n; \forall p \in P. p < \text{length } w \rrbracket \implies P = P'$   
unfolding *dec-interp-def* *positions-in-row* by auto

**lemma** *length-dec-interp*[*simp*]:  $\text{length } (\text{dec-interp } n \text{ FO } x) = n$   
by (auto simp: *dec-interp-def*)

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

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

**lemma** *lang-ENC*:

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

**by** (*force simp: ENC-def dec-word-def wf-word*  
*in-set-conv-nth*[of - *dec-interp*  $n$  {}] *x for x*] *positions-in-row Ball-def*  
*intro!*: *enc-atom- $\sigma$  exI*[of - *dec-word*  $x$  :: 'a list *for x*] *exI*[of - *dec-interp*  $n$  {}]  
*x for x*]  
*enc-dec*[of  $n$  - {}, *symmetric, unfolded dec-word-def enc.simps map-index-map*])  
**next**  
**case** *False*  
**hence** *nonempty: valid-ENC*  $n$  ' *FO*  $\neq$  {} **by** *simp*  
**have** *finite: finite* (*valid-ENC*  $n$  ' *FO*)  
**by** (*intro finite-imageI*[*OF finite-subset*[*OF assms*(1)]]] *auto*  
**from** *False assms*(1) **have**  $0 < n$  **by** *auto*  
**with** *wf-rexp-valid-ENC assms*(1) **have** *wf-rexp*:  $\forall x \in \text{valid-ENC } n \text{ ' } FO. \text{wf } n$   
*x by auto*  
{ **fix**  $r w I$  **assume**  $r \in FO$  **and** \*: *length*  $I = n$  *wf-interp*  $w I$   
( $\forall i \in FO. \text{case } I ! i \text{ of } Inl - \Rightarrow \text{True} \mid Inr - \Rightarrow \text{False}$ )  
( $\forall i \in SO. \text{case } I ! i \text{ of } Inl - \Rightarrow \text{False} \mid Inr - \Rightarrow \text{True}$ )  
**then obtain**  $p$  **where**  $p: I ! r = Inl p$  **by** (*cases*  $I ! r$ ) *auto*  
**moreover from**  $\langle r \in FO \rangle \text{assms}(1) *(1)$  **have**  $r < \text{length } I$  **by** *auto*  
**ultimately have**  $Inl p \in \text{set } I$  **by** (*auto simp add: in-set-conv-nth*)  
**with** \*(2) **have**  $p < \text{length } w$  **by** *auto*  
**with** \*(2) **obtain**  $a$  **where**  $a: w ! p = a$   $a \in \text{set } \Sigma$  **by** *auto*  
**from**  $\langle r < \text{length } I \rangle p *(1) \langle a \in \text{set } \Sigma \rangle$   
**have** [*enc-atom*  $I p a$ ]  $\in \text{lang } n$  (*Atom* (*Arbitrary-Except*  $r$  *True*))  
**by** (*intro enc-atom-lang-Arbitrary-Except-True*) *auto*  
**moreover**  
**from**  $\langle r < \text{length } I \rangle p *(1) \langle a \in \text{set } \Sigma \rangle *(2) \langle p < \text{length } w \rangle$   
**have** *take*  $p$  (*enc* ( $w, I$ ))  $\in \text{star}$  (*lang*  $n$  (*Atom* (*Arbitrary-Except*  $r$  *False*)))  
**by** (*auto simp: in-set-conv-nth simp del: lang.simps*  
*intro!*: *Ball-starI enc-atom-lang-Arbitrary-Except-False*) *auto*  
**moreover**  
**from**  $\langle r < \text{length } I \rangle p *(1) \langle a \in \text{set } \Sigma \rangle *(2) \langle p < \text{length } w \rangle$   
**have** *drop* (*Suc*  $p$ ) (*enc* ( $w, I$ ))  $\in \text{star}$  (*lang*  $n$  (*Atom* (*Arbitrary-Except*  $r$  *False*)))  
*False*))  
**by** (*auto simp: in-set-conv-nth simp del: lang.simps*  
*intro!*: *Ball-starI enc-atom-lang-Arbitrary-Except-False*) *auto*  
**ultimately have** *take*  $p$  (*enc* ( $w, I$ )) @ [*enc-atom*  $I p a$ ] @ *drop* ( $p + 1$ ) (*enc*  
( $w, I$ ))  $\in$   
*lang*  $n$  (*valid-ENC*  $n$   $r$ ) **using**  $\langle 0 < n \rangle$  **unfolding** *valid-ENC-def* **by** (*auto*  
*simp del: append.simps*)  
**with**  $\langle p < \text{length } w \rangle a$  **have** *enc* ( $w, I$ )  $\in \text{lang } n$  (*valid-ENC*  $n$   $r$ )  
**using** *id-take-nth-drop*[of  $p$  *enc* ( $w, I$ )] **by** *auto*  
}  
**hence**  $?R \subseteq ?L$  **using** *lang-flatten-INTERSECT*[*OF finite nonempty wf-rexp*]  
**by** (*auto simp add: ENC-def*)  
**moreover**  
{ **fix**  $x$  **assume**  $x \in (\bigcap r \in \text{valid-ENC } n \text{ ' } FO. \text{lang } n r)$   
**hence**  $r: \forall r \in FO. x \in \text{lang } n$  (*valid-ENC*  $n$   $r$ ) **by** *blast*  
**have** *length* (*dec-interp*  $n$  *FO*  $x$ ) =  $n$  **unfolding** *dec-interp-def* **by** *simp*  
**moreover**

```

{ fix  $r$  assume  $r \in FO$ 
  with  $assms$  have  $r < n$  by  $auto$ 
  from  $\langle r \in FO \rangle r$  obtain  $u v w$  where  $uvw: x = u @ v @ w$ 
     $u \in star (lang\ n (Atom (Arbitrary-Except\ r\ False)))$ 
     $v \in lang\ n (Atom (Arbitrary-Except\ r\ True))$ 
     $w \in star (lang\ n (Atom (Arbitrary-Except\ r\ False)))$  using  $\langle 0 < n \rangle$  unfolding
   $valid-ENC-def$ 
    by  $(fastforce\ simp\ del: lang.simps(4))$ 
    hence  $length\ u < length\ x \wedge i. i < length\ x \longrightarrow snd\ (x\ !\ i) ! r \longleftrightarrow i = length\ u$ 
    by  $(auto\ simp: nth-append\ nth-Cons'\ split: if-split-asm\ simp\ del: lang.simps\ dest!: Arbitrary-ExceptD[OF - \langle r < n \rangle]\ dest: star-Arbitrary-ExceptD[OF - \langle r < n \rangle, of\ u]\ elim!: iffD1[OF star-Arbitrary-ExceptD[OF - \langle r < n \rangle, of\ w\ False]])$   $auto$ 
    hence  $\exists! p. p < length\ x \wedge snd\ (x\ !\ p) ! r$  by  $auto$ 
  } note  $*$  =  $this$ 
  have  $x-wf-word: wf-word\ n\ x$  using  $wf-lang-wf-word[OF\ wf-rexp-valid-ENC\ False\ r\ assms(1)]$ 
  by  $auto$ 
  with  $*$  have  $x = enc\ (dec-word\ x, dec-interp\ n\ FO\ x)$  by  $(intro\ sym[OF\ enc-dec])\ auto$ 
  moreover
  from  $*$  have  $wf-interp\ (dec-word\ x)\ (dec-interp\ n\ FO\ x)$ 
     $(\forall i \in FO. case\ dec-interp\ n\ FO\ x\ !\ i\ of\ Inl\ - \Rightarrow True\ | Inr\ - \Rightarrow False)$ 
     $(\forall i \in SO. case\ dec-interp\ n\ FO\ x\ !\ i\ of\ Inl\ - \Rightarrow False\ | Inr\ - \Rightarrow True)$ 
    using  $False\ x-wf-word[unfolding\ wf-word, unfolded\ \sigma-def\ o-apply\ set-concat\ set-map\ set-n-lists, simplified]\ assms$ 
     $Inl-dec-interp-length[OF\ ballI, of\ FO\ x - n]\ Inr-dec-interp-length[of - n\ FO\ x]$ 
     $dec-interp-Inl[of - FO\ n\ x]\ dec-interp-Inr[of - FO\ n\ x]$ 
    by  $(fastforce\ simp\ add: dec-word-def\ split: sum.split)+$ 
  ultimately have  $x \in ?R$  by  $blast$ 
  }
  with  $False\ lang-flatten-INTERSECT[OF\ finite\ nonempty\ wf-rexp]$  have  $?L \subseteq ?R$  by  $(auto\ simp\ add: ENC-def)$ 
  ultimately show  $?thesis$  by  $blast$ 
qed

```

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

```

assumes  $wf-formula\ n\ \varphi$ 
shows  $lang\ n\ (ENC\ n\ (FOV\ \varphi)) - \{\}\ = \{enc\ (w, I) \mid w\ I. length\ I = n \wedge wf-interp-for-formula\ (w, I)\ \varphi\}$ 
proof -
  from  $assms\ max-idx-vars$  have  $*$ :  $FOV\ \varphi \subseteq \{0 ..< n\}\ SOV\ \varphi \subseteq \{0 ..< n\} - FOV\ \varphi$  by  $auto$ 
  show  $?thesis$  unfolding  $lang-ENC[OF\ *]$  by  $simp$ 
qed

```

## 10.2 Welldefinedness of enc wrt. Models

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

$enc (w, x \# I) = map-index (\lambda n (a, bs). (a, (case x of Inl p \Rightarrow n = p \mid Inr P \Rightarrow n \in P) \# bs)) (enc (w, I))$   
**by** (*auto simp: comp-def*)

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

**unfolding** *enc-alt-def* **by** *auto*

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

$\llbracket wf-formula (length I) (FExists \varphi); w \neq [] \rrbracket \Longrightarrow$   
 $wf-interp-for-formula (w, I) (FExists \varphi) \longleftrightarrow$   
 $(\forall p < length w. wf-interp-for-formula (w, Inl p \# I) \varphi)$   
**by** (*auto simp: nth-Cons' split: if-split-asm*)

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

**by** (*auto simp: nth-Cons' split: if-split-asm*)

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

$\llbracket wf-formula (length I) (FEXISTS \varphi); w \neq [] \rrbracket \Longrightarrow$   
 $wf-interp-for-formula (w, I) (FEXISTS \varphi) \longleftrightarrow (\forall P \subseteq \{0 .. length w - 1\}.$   
 $wf-interp-for-formula (w, Inr P \# I) \varphi)$   
**by** (*auto simp: neq-Nil-conv nth-Cons'*)

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

**by** (*cases w*) (*auto simp: nth-Cons' split: if-split-asm*)

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

**by** (*auto elim: map-index-eq-imp-length-eq*)

**lemma** *enc-length*:

**assumes**  $w \neq []$   $enc (w, I) = enc (w', I')$

**shows**  $length I = length I'$

**using** *assms*

$length-map[of (\lambda x. case x of Inl p \Rightarrow 0 = p \mid Inr P \Rightarrow 0 \in P) I]$

$length-map[of (\lambda x. case x of Inl p \Rightarrow 0 = p \mid Inr P \Rightarrow 0 \in P) I']$

**by** (*induct rule: list-induct2[OF enc-word-length[OF assms(2)]]*) *auto*

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

$wf-interp-for-formula (w, I) (FOr \varphi1 \varphi2) =$

$(wf-interp-for-formula (w, I) \varphi1 \wedge wf-interp-for-formula (w, I) \varphi2)$

**by** *auto*

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

$wf-interp-for-formula (w, I) (FAnd \varphi1 \varphi2) =$

$(wf-interp-for-formula (w, I) \varphi1 \wedge wf-interp-for-formula (w, I) \varphi2)$

by *auto*

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

**assumes** *wf-formula* (length *I*)  $\varphi$  *wf-interp-for-formula* (*w*, *I*)  $\varphi$   
**shows** *wf-interp-for-formula* (*dec-word* (*enc* (*w*, *I*)), *dec-interp* (length *I*) (*FOV*  $\varphi$ ) (*enc* (*w*, *I*)))  $\varphi$   
(**is** *wf-interp-for-formula* (-, ?*dec*)  $\varphi$ )  
**unfolding** *dec-word-enc*  
**proof** –  
  { **fix** *i* **assume** *i*: *i*  $\in$  *FOV*  $\varphi$   
    **with** *assms*(2) **have**  $\exists p. I ! i = \text{Inl } p$  **by** (*cases* *I* ! *i*) *auto*  
    **with** *i* *assms* **have**  $\exists ! p. p < \text{length } (\text{enc } (w, I)) \wedge \text{snd } (\text{enc } (w, I) ! p) ! i$   
      **by** (*intro* *enc-unique*[of *w* *I* *i*]) (*auto* *intro*!: *bspec*[*OF* *max-idx-vars*] *split*:  
    *sum.splits*)  
  } **note** \* = *this*  
  **have**  $\forall x \in \text{set } ?dec. \text{case-sum } (\lambda p. p < \text{length } w) (\lambda P. \forall p \in P. p < \text{length } w) x$   
  **proof** (*intro* *ballI*, *split* *sum.split*, *safe*)  
    **fix** *p* **assume** *Inl* *p*  $\in$  *set* ?*dec*  
    **thus**  $p < \text{length } w$  **using** *Inl-dec-interp-length*[*OF* *ballI*[*OF* \*]] **by** *auto*  
  **next**  
    **fix** *p* *P* **assume** *Inr* *P*  $\in$  *set* ?*dec*  $p \in P$   
    **thus**  $p < \text{length } w$  **using** *Inr-dec-interp-length* **by** *fastforce*  
  **qed**  
  **thus** *wf-interp-for-formula* (*w*, ?*dec*)  $\varphi$   
  **using** *assms*  
    *dec-interp-Inl*[of - *FOV*  $\varphi$  length *I* *enc* (*w*, *I*), *OF* - *bspec*[*OF* *max-idx-vars*]]  
    *dec-interp-Inr*[of - *FOV*  $\varphi$  length *I* *enc* (*w*, *I*), *OF* - *bspec*[*OF* *max-idx-vars*]]  
  **by** (*fastforce* *split*: *sum.splits*)  
**qed**

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

**proof** (*induction*  $\varphi$  *arbitrary*: *I* *I'*)

**case** (*FQ* *a* *m*)  
  **let** ?*dec* =  $\lambda w I. (\text{dec-word } (\text{enc } (w, I)), \text{dec-interp } (\text{length } I) (\text{FOV } (\text{FQ } a \ m))$   
  (*enc* (*w*, *I*)))  
  **from** *FQ*(2,3) **have** *satisfies* (*w*, *I*) (*FQ* *a* *m*) = *satisfies* (?*dec* *w* *I*) (*FQ* *a* *m*)  
  **unfolding** *dec-word-enc*  
  **using** *dec-interp-enc-Inl*[of length *I* {*m*} *w* *I* *m*]  
  **by** (*auto* *intro*: *nth-mem* *dest*: *dec-interp-not-Inr* *split*: *sum.splits*) (*metis* *nth-mem*)  
  **moreover**  
  **from** *FQ*(3) **have** \*:  $w \neq []$  **by** *simp*  
  **from** *FQ*(2,4) **have** *satisfies* (*w'*, *I'*) (*FQ* *a* *m*) = *satisfies* (?*dec* *w'* *I'*) (*FQ* *a* *m*)  
  **unfolding** *dec-word-enc* *enc-length*[*OF* \* *FQ*(1)]  
  **using** *dec-interp-enc-Inl*[of length *I'* {*m*} *w'* *I'* *m*]  
  **by** (*auto* *dest*: *dec-interp-not-Inr* *split*: *sum.splits*) (*metis* *nth-mem*)  
  +



**ultimately show** *?case unfolding*  $FQ(1)$  *enc-length*[ $OF * FQ(1)$ ] ..  
**next**  
**case** ( $FLess\ m\ n$ )  
**let**  $?dec = \lambda w\ I.$  (*dec-word* (*enc* ( $w, I$ )), *dec-interp* (*length*  $I$ ) ( $FOV$  ( $FLess\ m\ n$ )) (*enc* ( $w, I$ )))  
**from**  $FLess(2,3)$  **have** *satisfies* ( $w, I$ ) ( $FLess\ m\ n$ ) = *satisfies* ( $?dec$  ( $TYPE$  ('a))  $w\ I$ ) ( $FLess\ m\ n$ )  
**unfolding** *dec-word-enc*  
**using** *dec-interp-enc-Inl*[*of length*  $I$  { $m, n$ }  $w\ I\ m$ ] *dec-interp-enc-Inl*[*of length*  $I$  { $m, n$ }  $w\ I\ n$ ]  
**by** (*fastforce intro: nth-mem dest: dec-interp-not-Inr split: sum.splits*)  
**moreover**  
**from**  $FLess(3)$  **have**  $*$ :  $w \neq []$  **by** *simp*  
**from**  $FLess(2,4)$  **have** *satisfies* ( $w', I'$ ) ( $FLess\ m\ n$ ) = *satisfies* ( $?dec$  ( $TYPE$  ('a))  $w'\ I'$ ) ( $FLess\ m\ n$ )  
**unfolding** *dec-word-enc enc-length*[ $OF * FLess(1)$ ]  
**using** *dec-interp-enc-Inl*[*of length*  $I'$  { $m, n$ }  $w'\ I'\ m$ ] *dec-interp-enc-Inl*[*of length*  $I'$  { $m, n$ }  $w'\ I'\ n$ ]  
**by** (*fastforce intro: nth-mem dest: dec-interp-not-Inr split: sum.splits*)  
**ultimately show** *?case unfolding*  $FLess(1)$  *enc-length*[ $OF * FLess(1)$ ] ..  
**next**  
**case** ( $FIn\ m\ M$ )  
**let**  $?dec = \lambda w\ I.$  (*dec-word* (*enc* ( $w, I$ )), *dec-interp* (*length*  $I$ ) ( $FOV$  ( $FIn\ m\ M$ )) (*enc* ( $w, I$ )))  
**from**  $FIn(2,3)$  **have** *satisfies* ( $w, I$ ) ( $FIn\ m\ M$ ) = *satisfies* ( $?dec$  ( $TYPE$  ('a))  $w\ I$ ) ( $FIn\ m\ M$ )  
**unfolding** *dec-word-enc*  
**using** *dec-interp-enc-Inl*[*of length*  $I$  { $m$ }  $w\ I\ m$ ] *dec-interp-enc-Inr*[*of length*  $I$  { $m$ }  $w\ I\ M$ ]  
**by** (*auto dest: dec-interp-not-Inr dec-interp-not-Inl split: sum.splits*) (*metis nth-mem*)+  
**moreover**  
**from**  $FIn(3)$  **have**  $*$ :  $w \neq []$  **by** *simp*  
**from**  $FIn(2,4)$  **have** *satisfies* ( $w', I'$ ) ( $FIn\ m\ M$ ) = *satisfies* ( $?dec$  ( $TYPE$  ('a))  $w'\ I'$ ) ( $FIn\ m\ M$ )  
**unfolding** *dec-word-enc enc-length*[ $OF * FIn(1)$ ]  
**using** *dec-interp-enc-Inl*[*of length*  $I'$  { $m$ }  $w'\ I'\ m$ ] *dec-interp-enc-Inr*[*of length*  $I'$  { $m$ }  $w'\ I'\ M$ ]  
**by** (*auto dest: dec-interp-not-Inr dec-interp-not-Inl split: sum.splits*) (*metis nth-mem*)+  
**ultimately show** *?case unfolding*  $FIn(1)$  *enc-length*[ $OF * FIn(1)$ ] ..  
**next**  
**case** ( $FOr\ \varphi1\ \varphi2$ ) **show** *?case unfolding* *satisfies.simps*(5)  
**proof** (*intro disj-cong*)  
**from**  $FOr(3-6)$  **show** *satisfies* ( $w, I$ )  $\varphi1$  = *satisfies* ( $w', I'$ )  $\varphi1$   
**by** (*intro FOr(1)*) *auto*  
**next**  
**from**  $FOr(3-6)$  **show** *satisfies* ( $w, I$ )  $\varphi2$  = *satisfies* ( $w', I'$ )  $\varphi2$   
**by** (*intro FOr(2)*) *auto*

```

qed
next
case (FAnd  $\varphi_1$   $\varphi_2$ ) show ?case unfolding satisfies.simps(6)
proof (intro conj-cong)
  from FAnd(3-6) show satisfies (w, I)  $\varphi_1$  = satisfies (w', I')  $\varphi_1$ 
  by (intro FAnd(1)) auto
next
  from FAnd(3-6) show satisfies (w, I)  $\varphi_2$  = satisfies (w', I')  $\varphi_2$ 
  by (intro FAnd(2)) auto
qed
next
case (FExists  $\varphi$ )
  hence  $w \neq []$   $w' \neq []$  by auto
  hence length: length w = length w' length I = length I'
  using enc-word-length[OF FExists.prem(1)] enc-length[OF - FExists.prem(1)]
by auto
  show ?case
  proof
    assume satisfies (w, I) (FExists  $\varphi$ )
    with FExists.prem(3) obtain p where  $p < \text{length } w$  satisfies (w, Inl p # I)
 $\varphi$ 
    by (auto intro: ord-less-eq-trans[OF le-imp-less-Suc Suc-pred])
    moreover
    with FExists.prem have satisfies (w', Inl p # I')  $\varphi$ 
    proof (intro iffD1[OF FExists.IH[of Inl p # I Inl p # I']])
      from FExists.prem(2,3) ( $p < \text{length } w$ ) show wf-interp-for-formula (w, Inl
      p # I)  $\varphi$ 
      by (blast dest: wf-interp-for-formula-FExists[of I, OF - ( $w \neq []$ )])
    next
      from FExists.prem(2,4) ( $p < \text{length } w$ ) show wf-interp-for-formula (w', Inl
      p # I')  $\varphi$ 
      unfolding length by (blast dest: wf-interp-for-formula-FExists[of I', OF -
      ( $w' \neq []$ )])
    qed (auto elim: enc-extend-interp simp del: enc.simps)
    ultimately show satisfies (w', I') (FExists  $\varphi$ ) using length by (auto intro!:
    exI[of - p])
  next
    assume satisfies (w', I') (FExists  $\varphi$ )
    with FExists.prem(1,2,4) obtain p where  $p < \text{length } w'$  satisfies (w', Inl p
    # I')  $\varphi$ 
    by (auto intro: ord-less-eq-trans[OF le-imp-less-Suc Suc-pred])
    moreover
    with FExists.prem have satisfies (w, Inl p # I)  $\varphi$ 
    proof (intro iffD2[OF FExists.IH[of Inl p # I Inl p # I']])
      from FExists.prem(2,3) ( $p < \text{length } w'$ ) show wf-interp-for-formula (w, Inl
      p # I)  $\varphi$ 
      unfolding length[symmetric] by (blast dest: wf-interp-for-formula-FExists[of
      I, OF - ( $w \neq []$ )])
    next

```

```

    from FExists.prem(2,4) ⟨p < length w'⟩ show wf-interp-for-formula (w', Inl
p # I') φ
    unfolding length by (blast dest: wf-interp-for-formula-FExists[of I', OF -
⟨w' ≠ []⟩])
    qed (auto elim: enc-extend-interp simp del: enc.simps)
    ultimately show satisfies (w, I) (FExists φ) using length by (auto intro!:
exI[of - p])
    qed
next
case (FEXISTS φ)
hence w ≠ [] w' ≠ [] by auto
hence length: length w = length w' length I = length I'
using enc-word-length[OF FEXISTS.prem(1)] enc-length[OF - FEXISTS.prem(1)]
by auto
show ?case
proof
assume satisfies (w, I) (FEXISTS φ)
then obtain P where P: P ⊆ {0 .. length w - 1} satisfies (w, Inr P # I)
φ by auto
moreover
with FEXISTS.prem have satisfies (w', Inr P # I') φ
proof (intro iffD1[OF FEXISTS.IH[of Inr P # I Inr P # I']])
from FEXISTS.prem(2,3) P(1) show wf-interp-for-formula (w, Inr P #
I) φ
by (blast dest: wf-interp-for-formula-FEXISTS[of I, OF - ⟨w ≠ []⟩])
next
from FEXISTS.prem(2,4) P(1) show wf-interp-for-formula (w', Inr P #
I') φ
unfolding length by (blast dest: wf-interp-for-formula-FEXISTS[of I', OF
- ⟨w' ≠ []⟩])
qed (auto elim: enc-extend-interp simp del: enc.simps)
ultimately show satisfies (w', I') (FEXISTS φ) using length by (auto intro!:
exI[of - P])
next
assume satisfies (w', I') (FEXISTS φ)
then obtain P where P: P ⊆ {0 .. length w' - 1} satisfies (w', Inr P # I')
φ by auto
moreover
with FEXISTS.prem have satisfies (w, Inr P # I) φ
proof (intro iffD2[OF FEXISTS.IH[of Inr P # I Inr P # I']])
from FEXISTS.prem(2,3) P(1) show wf-interp-for-formula (w, Inr P #
I) φ
unfolding length[symmetric] by (blast dest: wf-interp-for-formula-FEXISTS[of
I, OF - ⟨w ≠ []⟩])
next
from FEXISTS.prem(2,4) P(1) show wf-interp-for-formula (w', Inr P #
I') φ
unfolding length by (blast dest: wf-interp-for-formula-FEXISTS[of I', OF
- ⟨w' ≠ []⟩])

```

**qed** (*auto elim: enc-extend-interp simp del: enc.simps*)  
**ultimately show** *satisfies* ( $w, I$ ) (*FEXISTS*  $\varphi$ ) **using** *length by* (*auto intro!*:  
*exI[of - P]*)  
**qed**  
**qed** *auto*

**lemma** *lang<sub>M2L</sub>-FOr*:

**assumes** *wf-formula*  $n$  (*FOr*  $\varphi_1$   $\varphi_2$ )  
**shows** *lang<sub>M2L</sub>*  $n$  (*FOr*  $\varphi_1$   $\varphi_2$ )  $\subseteq$   
 $(\text{lang}_{M2L} \ n \ \varphi_1 \cup \text{lang}_{M2L} \ n \ \varphi_2) \cap \{enc \ (w, I) \mid w \ I. \ length \ I = n \wedge$   
*wf-interp-for-formula* ( $w, I$ ) (*FOr*  $\varphi_1$   $\varphi_2$ ) $\}$   
*(is -  $\subseteq$  (?L1  $\cup$  ?L2)  $\cap$  ?ENC)*  
**proof** (*intro equalityI subsetI*)  
**fix**  $x$  **assume**  $x \in \text{lang}_{M2L} \ n$  (*FOr*  $\varphi_1$   $\varphi_2$ )  
**then obtain**  $w \ I$  **where**  
 $*$ :  $x = enc \ (w, I)$  *wf-interp-for-formula* ( $w, I$ ) (*FOr*  $\varphi_1$   $\varphi_2$ ) *length*  $I = n$  *length*  
 $w > 0$  **and**  
*satisfies* ( $w, I$ )  $\varphi_1 \vee$  *satisfies* ( $w, I$ )  $\varphi_2$  **unfolding** *lang<sub>M2L</sub>-def* **by** *auto*  
**thus**  $x \in$   $(?L1 \cup ?L2) \cap ?ENC$   
**proof** (*elim disjE*)  
**assume** *satisfies* ( $w, I$ )  $\varphi_1$   
**with**  $*$  **have**  $x \in ?L1$  **using** *assms unfolding lang<sub>M2L</sub>-def* **by** (*fastforce*)  
**with**  $*$  **show** *?thesis* **by** *auto*  
**next**  
**assume** *satisfies* ( $w, I$ )  $\varphi_2$   
**with**  $*$  **have**  $x \in ?L2$  **using** *assms unfolding lang<sub>M2L</sub>-def* **by** (*fastforce*)  
**with**  $*$  **show** *?thesis* **by** *auto*  
**qed**  
**qed**

**lemma** *lang<sub>M2L</sub>-FAnd*:

**assumes** *wf-formula*  $n$  (*FAnd*  $\varphi_1$   $\varphi_2$ )  
**shows** *lang<sub>M2L</sub>*  $n$  (*FAnd*  $\varphi_1$   $\varphi_2$ )  $\subseteq$   
 $\text{lang}_{M2L} \ n \ \varphi_1 \cap \text{lang}_{M2L} \ n \ \varphi_2 \cap \{enc \ (w, I) \mid w \ I. \ length \ I = n \wedge$   
*wf-interp-for-formula* ( $w, I$ ) (*FAnd*  $\varphi_1$   $\varphi_2$ ) $\}$   
*(is -  $\subseteq$  ?L1  $\cap$  ?L2  $\cap$  ?ENC)*  
**using** *assms unfolding lang<sub>M2L</sub>-def* **by** (*fastforce*)

### 10.3 From M2L to Regular expressions

**fun** *rexp-of*  $:: \text{nat} \Rightarrow 'a \ \text{formula} \Rightarrow ('a \ \text{atom}) \ \text{rexp} \ \text{where}$

*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*  $m1 \ \text{True}$ ), *Full*, *Atom* (*Arbitrary-Except*  
 $m2 \ \text{True}$ ), *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*  $\varphi$ )))

|  $\text{rexp-of } n \text{ (FOr } \varphi_1 \varphi_2) = \text{Inter (Plus (rexp-of } n \varphi_1) \text{ (rexp-of } n \varphi_2)) \text{ (ENC } n \text{ (FOV (FOr } \varphi_1 \varphi_2)))}$   
|  $\text{rexp-of } n \text{ (FAnd } \varphi_1 \varphi_2) = \text{INTERSECT [rexp-of } n \varphi_1, \text{ rexp-of } n \varphi_2, \text{ ENC } n \text{ (FOV (FAnd } \varphi_1 \varphi_2))}]$   
|  $\text{rexp-of } n \text{ (FExists } \varphi) = \text{Pr (rexp-of (} n + 1) \varphi)$   
|  $\text{rexp-of } n \text{ (FEXISTS } \varphi) = \text{Pr (rexp-of (} n + 1) \varphi)$

**fun**  $\text{rexp-of-alt} :: \text{nat} \Rightarrow 'a \text{ formula} \Rightarrow ('a \text{ atom}) \text{ rexp where}$   
 $\text{rexp-of-alt } n \text{ (FQ } a \text{ m)} = \text{TIMES [Full, Atom (AQ } m \text{ a), Full]}$   
|  $\text{rexp-of-alt } n \text{ (FLess } m1 \text{ m2)} = \text{(if } m1 = m2 \text{ then Zero else}$   
 $\text{TIMES [Full, Atom (Arbitrary-Except } m1 \text{ True), Full, Atom (Arbitrary-Except}$   
 $m2 \text{ True), Full])}$   
|  $\text{rexp-of-alt } n \text{ (FIn } m \text{ M)} = \text{TIMES [Full, Atom (Arbitrary-Except2 } m \text{ M), Full]}$   
|  $\text{rexp-of-alt } n \text{ (FNot } \varphi) = \text{rexp.Not (rexp-of-alt } n \varphi)$   
|  $\text{rexp-of-alt } n \text{ (FOr } \varphi_1 \varphi_2) = \text{Plus (rexp-of-alt } n \varphi_1) \text{ (rexp-of-alt } n \varphi_2)$   
|  $\text{rexp-of-alt } n \text{ (FAnd } \varphi_1 \varphi_2) = \text{Inter (rexp-of-alt } n \varphi_1) \text{ (rexp-of-alt } n \varphi_2)$   
|  $\text{rexp-of-alt } n \text{ (FExists } \varphi) = \text{Pr (Inter (rexp-of-alt (} n + 1) \varphi) \text{ (ENC (} n + 1) \text{ (FOV } \varphi)))}$   
|  $\text{rexp-of-alt } n \text{ (FEXISTS } \varphi) = \text{Pr (Inter (rexp-of-alt (} n + 1) \varphi) \text{ (ENC (} n + 1) \text{ (FOV } \varphi)))}$

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

**fun**  $\text{rexp-of-alt}' :: \text{nat} \Rightarrow 'a \text{ formula} \Rightarrow ('a \text{ atom}) \text{ rexp where}$   
 $\text{rexp-of-alt}' n \text{ (FQ } a \text{ m)} = \text{TIMES [Full, Atom (AQ } m \text{ a), Full]}$   
|  $\text{rexp-of-alt}' n \text{ (FLess } m1 \text{ m2)} = \text{(if } m1 = m2 \text{ then Zero else}$   
 $\text{TIMES [Full, Atom (Arbitrary-Except } m1 \text{ True), Full, Atom (Arbitrary-Except}$   
 $m2 \text{ True), Full])}$   
|  $\text{rexp-of-alt}' n \text{ (FIn } m \text{ M)} = \text{TIMES [Full, Atom (Arbitrary-Except2 } m \text{ M), Full]}$   
|  $\text{rexp-of-alt}' n \text{ (FNot } \varphi) = \text{rexp.Not (rexp-of-alt}' n \varphi)$   
|  $\text{rexp-of-alt}' n \text{ (FOr } \varphi_1 \varphi_2) = \text{Plus (rexp-of-alt}' n \varphi_1) \text{ (rexp-of-alt}' n \varphi_2)$   
|  $\text{rexp-of-alt}' n \text{ (FAnd } \varphi_1 \varphi_2) = \text{Inter (rexp-of-alt}' n \varphi_1) \text{ (rexp-of-alt}' n \varphi_2)$   
|  $\text{rexp-of-alt}' n \text{ (FExists } \varphi) = \text{Pr (Inter (rexp-of-alt}' (} n + 1) \varphi) \text{ (ENC (} n + 1) \text{ \{0\})})}$   
|  $\text{rexp-of-alt}' n \text{ (FEXISTS } \varphi) = \text{Pr (rexp-of-alt}' (} n + 1) \varphi)$

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

**theorem**  $\text{lang}_{M2L}\text{-rexp-of: wf-formula } n \varphi \Longrightarrow \text{lang}_{M2L} n \varphi = \text{lang } n \text{ (rexp-of } n \varphi) - \{\{\}\}$

(is  $- \Longrightarrow - = ?L n \varphi$ )

**proof** (induct  $\varphi$  arbitrary:  $n$ )

**case** (FQ  $a \text{ m}$ )

**show**  $?case$

**proof** (intro equalityI subsetI)

**fix**  $x$  **assume**  $x \in \text{lang}_{M2L} n \text{ (FQ } a \text{ m)}$

**then obtain**  $w \text{ I where}$

$*: x = \text{enc } (w, I) \text{ wf-interp-for-formula } (w, I) \text{ (FQ } a \text{ m) satisfies } (w, I) \text{ (FQ } a \text{ m)}$

$length\ I = n$   
**unfolding**  $lang_{M2L}\text{-def}$  **by**  $blast$   
**with**  $FQ(1)$  **obtain**  $p$  **where**  $p: p < length\ w\ I ! m = Inl\ p\ w ! p = a$   
**by**  $(auto\ simp: all\text{-set}\text{-conv}\text{-all}\text{-nth}\ split: sum.\text{splits})$   
**with**  $*(1)$  **have**  $x = take\ p\ (enc\ (w, I)) @ [enc\text{-atom}\ I\ p\ a] @ drop\ (p + 1)$   
 $(enc\ (w, I))$   
**using**  $id\text{-take}\text{-nth}\text{-drop}[of\ p\ enc\ (w, I)]$  **by**  $auto$   
**moreover from**  $*(4)\ FQ(1)\ p(2)$   
**have**  $[enc\text{-atom}\ I\ p\ a] \in lang\ n\ (Atom\ (AQ\ m\ a))$   
**by**  $(intro\ enc\text{-atom}\text{-lang}\text{-AQ})\ auto$   
**moreover from**  $*(2,4)$  **have**  $take\ p\ (enc\ (w, I)) \in lang\ n\ (Full)$   
**by**  $(auto\ intro!: enc\text{-atom}\text{-}\sigma\ dest!: in\text{-set}\text{-take}D)$   
**moreover from**  $*(2,4)$  **have**  $drop\ (Suc\ p)\ (enc\ (w, I)) \in lang\ n\ (Full)$   
**by**  $(auto\ intro!: enc\text{-atom}\text{-}\sigma\ dest!: in\text{-set}\text{-drop}D)$   
**ultimately show**  $x \in ?L\ n\ (FQ\ a\ m)$  **using**  $*(1,2,4)$   
**unfolding**  $rexp\text{-of}.\text{sims}\ lang.\text{sims}(6,9)\ rexp\text{-of}\text{-list}.\text{sims}\ Int\text{-Diff}$   
 $lang\text{-ENC}\text{-formula}[OF\ FQ, unfolded\ FOV.\text{sims}]$   
**by**  $(auto\ elim: ssubst\ simp\ del: o\text{-apply}\ append.\text{sims}\ lang.\text{sims})$   
**next**  
**fix**  $x$  **assume**  $x: x \in ?L\ n\ (FQ\ a\ m)$   
**with**  $FQ$  **obtain**  $w\ I\ p$  **where**  $m: I ! m = Inl\ p\ m < length\ I$  **and**  
 $wI: x = enc\ (w, I)\ length\ I = n\ wf\text{-interp}\text{-for}\text{-formula}\ (w, I)\ (FQ\ a\ m)$   
**unfolding**  $rexp\text{-of}.\text{sims}\ lang.\text{sims}\ lang\text{-ENC}\text{-formula}[OF\ FQ, unfolded$   
 $FOV.\text{sims}] Int\text{-Diff}$   
**by**  $atomize\text{-elim}\ (auto\ split: sum.\text{splits})$   
**hence**  $wf\text{-interp}\text{-for}\text{-formula}\ (dec\text{-word}\ x, dec\text{-interp}\ n\ \{m\}\ x)\ (FQ\ a\ m)$  **un-**  
**folding**  $wI(1)$   
**using**  $enc\text{-wf}\text{-interp}[OF\ FQ(1)[folded\ wI(2)]]$  **by**  $auto$   
**moreover**  
**from**  $x$  **obtain**  $u1\ u\ u2$  **where**  $x = u1 @ u @ u2\ u \in lang\ n\ (Atom\ (AQ\ m$   
 $a))$   
**unfolding**  $rexp\text{-of}.\text{sims}\ lang.\text{sims}\ rexp\text{-of}\text{-list}.\text{sims}$  **using**  $concE$  **by**  $fast$   
**with**  $FQ(1)$  **obtain**  $v$  **where**  $v: x = u1 @ [v] @ u2\ snd\ v ! m\ fst\ v = a$   
**using**  $AQ\text{-D}[of\ u\ n\ m\ a]$  **by**  $fastforce$   
**hence**  $u: length\ u1 < length\ x$  **by**  $auto$   
**{ from**  $v$  **have**  $snd\ (x ! length\ u1) ! m$  **by**  $auto$   
**moreover**  
**from**  $m\ wI$  **have**  $p < length\ x\ snd\ (x ! p) ! m$   
**by**  $(fastforce\ intro: nth\text{-mem}\ split: sum.\text{splits})+$   
**moreover**  
**from**  $m\ wI$  **have**  $ex1: \exists!p. p < length\ x \wedge snd\ (x ! p) ! m$  **unfolding**  $wI(1)$   
**by**  $(intro\ enc\text{-unique})\ auto$   
**ultimately have**  $p = length\ u1$  **using**  $u$  **by**  $auto$   
**} note**  $* = this$   
**from**  $v$  **have**  $v = enc\ (w, I) ! length\ u1$  **unfolding**  $wI(1)$  **by**  $simp$   
**hence**  $a = w ! length\ u1$  **using**  $nth\text{-map}[OF\ u, of\ fst]$  **unfolding**  $wI(1)$   
 $v(3)[symmetric]$  **by**  $auto$   
**with**  $*\ m\ wI$  **have**  $satisfies\ (dec\text{-word}\ x, dec\text{-interp}\ n\ \{m\}\ x)\ (FQ\ a\ m)$   
**unfolding**  $dec\text{-word}\text{-enc}[of\ w\ I, folded\ wI(1)]$

**by** (*auto simp del: enc.simps dest: dec-interp-not-Inr split: sum.splits*)  
*(fastforce dest!: dec-interp-enc-Inl intro: nth-mem split: sum.splits)*  
**moreover from**  $wI$  **have** *wf-word*  $n$   $x$  **unfolding** *wf-word* **by** (*auto intro!: enc-atom- $\sigma$* )  
**ultimately show**  $x \in \text{lang}_{M2L} n$  ( $FQ$   $a$   $m$ ) **unfolding** *lang $_{M2L}$ -def* **using**  $m$   $wI(3)$   
**by** (*auto simp del: enc.simps intro!: exI[of - dec-word  $x$ ] exI[of - dec-interp  $n$  { $m$ }  $x$ ]*  
*intro: sym[OF enc-dec[OF - ballI[OF impI[OF enc-unique[of  $w$   $I$ , folded  $wI(1)]]]]]]]$ )  
**qed**  
**next**  
**case** ( $FLess\ m\ m'$ )  
**show** *?case*  
**proof** (*cases  $m = m'$* )  
**case** *False*  
**thus** *?thesis*  
**proof** (*intro equalityI subsetI*)  
**fix**  $x$  **assume**  $x \in \text{lang}_{M2L} n$  ( $FLess\ m\ m'$ )  
**then obtain**  $w\ I$  **where**  
 $*$ :  $x = enc(w, I)$  *wf-interp-for-formula* ( $w, I$ ) ( $FLess\ m\ m'$ ) *satisfies* ( $w, I$ ) ( $FLess\ m\ m'$ )  
 $length\ I = n$   
**unfolding** *lang $_{M2L}$ -def* **by** *blast*  
**with**  $FLess(1)$  **obtain**  $p\ q$  **where**  $pq$ :  $p < length\ w\ I$  !  $m = Inl\ p\ q < length\ w\ I$  !  $m' = Inl\ q\ p < q$   
**by** (*auto simp: all-set-conv-all-nth split: sum.splits*)  
**with**  $*(1)$  **have**  $x = take\ p\ (enc(w, I))$  @  $[enc-atom\ I\ p\ (w\ !\ p)]$  @  $drop\ (p + 1)\ (enc(w, I))$   
**using** *id-take-nth-drop*[of  $p\ enc(w, I)$ ] **by** *auto*  
**also have**  $drop\ (p + 1)\ (enc(w, I)) = take\ (q - p - 1)\ (drop\ (p + 1)\ (enc(w, I)))$  @  $[enc-atom\ I\ q\ (w\ !\ q)]$  @  $drop\ (q - p)\ (drop\ (p + 1)\ (enc(w, I)))$  (**is - = ?LHS**)  
**using** *id-take-nth-drop*[of  $q - p - 1\ drop\ (p + 1)\ (enc(w, I))$ ]  $pq$  **by** *auto*  
**finally have**  $x = take\ p\ (enc(w, I))$  @  $[enc-atom\ I\ p\ (w\ !\ p)]$  @ *?LHS* .  
**moreover from**  $*(2,4)$   $FLess(1)$   $pq(1,2)$   
**have**  $[enc-atom\ I\ p\ (w\ !\ p)] \in \text{lang}\ n$  (*Atom* (*Arbitrary-Except*  $m\ True$ ))  
**by** (*intro enc-atom-lang-Arbitrary-Except-True*) *auto*  
**moreover from**  $*(2,4)$   $FLess(1)$   $pq(3,4)$   
**have**  $[enc-atom\ I\ q\ (w\ !\ q)] \in \text{lang}\ n$  (*Atom* (*Arbitrary-Except*  $m'\ True$ ))  
**by** (*intro enc-atom-lang-Arbitrary-Except-True*) *auto*  
**moreover from**  $*(2,4)$  **have**  $take\ p\ (enc(w, I)) \in \text{lang}\ n$  (*Full*)  
**by** (*auto intro!: enc-atom- $\sigma$  dest!: in-set-takeD*)  
**moreover from**  $*(2,4)$  **have**  $take\ (q - p - 1)\ (drop\ (Suc\ p)\ (enc(w, I))) \in \text{lang}\ n$  (*Full*)  
**by** (*auto intro!: enc-atom- $\sigma$  dest!: in-set-dropD in-set-takeD*)  
**moreover from**  $*(2,4)$  **have**  $drop\ (q - p)\ (drop\ (Suc\ p)\ (enc(w, I))) \in \text{lang}\ n$  (*Full*)*

```

    by (auto intro!: enc-atom-σ dest!: in-set-dropD)
  ultimately show  $x \in ?L\ n\ (FLess\ m\ m')$  using  $*(1,2,4)$ 
    unfolding rexp-of.simps lang.simps(6,9) rexp-of-list.simps Int-Diff
      lang-ENC-formula[OF FLess, unfolded FOV.simps] if-not-P[OF False]
    by (auto elim: ssubst simp del: o-apply append.simps lang.simps)
next
fix  $x$  assume  $x: x \in ?L\ n\ (FLess\ m\ m')$ 
with FLess obtain  $w\ I$  where
   $wI: x = enc\ (w, I)\ length\ I = n\ wf\ interp\ for\ formula\ (w, I)\ (FLess\ m\ m')$ 
  unfolding rexp-of.simps lang.simps lang-ENC-formula[OF FLess, unfolded
FOV.simps] Int-Diff
    if-not-P[OF False]
  by (fastforce split: sum.splits)
with FLess obtain  $p\ p'$  where  $m: I\ !\ m = Inl\ p\ m < length\ I\ I\ !\ m' = Inl\ p'\ m' < length\ I$ 
  by (auto split: sum.splits)
with  $wI$  have  $wf\ interp\ for\ formula\ (dec\ word\ x, dec\ interp\ n\ \{m, m'\}\ x)$ 
 $(FLess\ m\ m')$  unfolding  $wI(1)$ 
  using  $enc\ wf\ interp[OF\ FLess(1)[folded\ wI(2)]]$  by auto
moreover
from  $x$  obtain  $u1\ u\ u2\ u'\ u3$  where  $x = u1\ @\ u\ @\ u2\ @\ u'\ @\ u3$ 
   $u \in lang\ n\ (Atom\ (Arbitrary\ Except\ m\ True))$ 
   $u' \in lang\ n\ (Atom\ (Arbitrary\ Except\ m'\ True))$ 
  unfolding rexp-of.simps lang.simps rexp-of-list.simps if-not-P[OF False]
using concE by fast
with FLess(1) obtain  $v\ v'$  where  $v: x = u1\ @\ [v]\ @\ u2\ @\ [v']\ @\ u3\ snd\ v\ !\ m\ snd\ v'\ !\ m'$ 
  using Arbitrary-ExceptD[of  $u\ n\ m\ True$ ] Arbitrary-ExceptD[of  $u'\ n\ m'\ True$ ]
by fastforce
hence  $u: length\ u1 < length\ x$  and  $u': Suc\ (length\ u1 + length\ u2) < length\ x$ 
(is  $?u' < -$ ) by auto
{ from  $v$  have  $snd\ (x\ !\ length\ u1)\ !\ m$  by auto
  moreover
  from  $m\ wI$  have  $p < length\ x\ snd\ (x\ !\ p)\ !\ m$ 
    by (fastforce intro: nth-mem split: sum.splits)+
  moreover
  from  $m\ wI$  have  $ex1: \exists!p. p < length\ x \wedge snd\ (x\ !\ p)\ !\ m$  unfolding  $wI(1)$ 
by (intro enc-unique) auto
  ultimately have  $p = length\ u1$  using  $u$  by auto
}
{ from  $v$  have  $snd\ (x\ !\ ?u')\ !\ m'$  by (auto simp: nth-append)
  moreover
  from  $m\ wI$  have  $p' < length\ x\ snd\ (x\ !\ p')\ !\ m'$ 
    by (fastforce intro: nth-mem split: sum.splits)+
  moreover
  from  $m\ wI$  have  $ex1: \exists!p. p < length\ x \wedge snd\ (x\ !\ p)\ !\ m'$  unfolding
 $wI(1)$  by (intro enc-unique) auto
  ultimately have  $p' = ?u'$  using  $u'$  by auto
} note  $* = this\ \langle p = length\ u1 \rangle$ 

```



```

with *  $m$   $wI$  have satisfies (dec-word  $x$ , dec-interp  $n$   $\{m, m'\}$   $x$ ) (FLess  $m$ 
 $m'$ )
  unfolding dec-word-enc[of  $w$   $I$ , folded  $wI(1)$ ]
  by (auto simp del: enc.simps dest: dec-interp-not-Inr split: sum.splits)
    (fastforce dest!: dec-interp-enc-Inl intro: nth-mem split: sum.splits)
  moreover from  $wI$  have wf-word  $n$   $x$  unfolding wf-word by (auto intro!:
enc-atom- $\sigma$ )
  ultimately show  $x \in \text{lang}_{M2L} n$  (FLess  $m$   $m'$ ) unfolding lang $_{M2L}$ -def
using  $m$   $wI(3)$ 
  by (auto simp del: enc.simps intro!: exI[of - dec-word  $x$ ] exI[of - dec-interp
 $n$   $\{m, m'\}$   $x$ ]
    intro: sym[OF enc-dec[OF - ballI[OF impI[OF enc-unique[of  $w$   $I$ , folded
 $wI(1)$ ]]]]]]])
  qed
qed (simp add: lang $_{M2L}$ -def del: o-apply)
next
case (FIN  $m$   $M$ )
show ?case
proof (intro equalityI subsetI)
  fix  $x$  assume  $x \in \text{lang}_{M2L} n$  (FIN  $m$   $M$ )
  then obtain  $w$   $I$  where
    * :  $x = \text{enc}(w, I)$  wf-interp-for-formula ( $w, I$ ) (FIN  $m$   $M$ ) satisfies ( $w, I$ )
(FIN  $m$   $M$ )
    length  $I = n$ 
  unfolding lang $_{M2L}$ -def by blast
  with FIN(1) obtain  $p$   $P$  where  $p : p < \text{length } w$   $I ! m = \text{Inl } p$   $I ! M = \text{Inr}$ 
 $P$   $p \in P$ 
  by (auto simp: all-set-conv-all-nth split: sum.splits)
  with *(1) have  $x = \text{take } p$  (enc ( $w, I$ )) @ [enc-atom  $I$   $p$  ( $w ! p$ )] @ drop ( $p$ 
+ 1) (enc ( $w, I$ ))
  using id-take-nth-drop[of  $p$  (enc ( $w, I$ ))] by auto
moreover
from *(2,4) FIN(1)  $p$  have [enc-atom  $I$   $p$  ( $w ! p$ )]  $\in \text{lang } n$  (Atom (Arbitrary-Except2
 $m$   $M$ ))
  by (intro enc-atom-lang-Arbitrary-Except2) auto
moreover from *(2,4) have  $\text{take } p$  (enc ( $w, I$ ))  $\in \text{lang } n$  (Full)
  by (auto intro!: enc-atom- $\sigma$  dest!: in-set-takeD)
moreover from *(2,4) have  $\text{drop } (\text{Suc } p)$  (enc ( $w, I$ ))  $\in \text{lang } n$  (Full)
  by (auto intro!: enc-atom- $\sigma$  dest!: in-set-dropD)
ultimately show  $x \in ?L n$  (FIN  $m$   $M$ ) using *(1,2,4)
  unfolding rexp-of.simps lang.simps(6,9) rexp-of-list.simps Int-Diff
lang-ENC-formula[OF FIN, unfolded FOV.simps]
  by (auto elim: ssubst simp del: o-apply append.simps lang.simps)
next
fix  $x$  assume  $x : x \in ?L n$  (FIN  $m$   $M$ )
with FIN obtain  $w$   $I$  where  $wI : x = \text{enc}(w, I)$  length  $I = n$  wf-interp-for-formula
( $w, I$ ) (FIN  $m$   $M$ )
  unfolding rexp-of.simps lang.simps lang-ENC-formula[OF FIN, unfolded
FOV.simps] Int-Diff

```

by (*fastforce split: sum.splits*)  
 with *FIn* obtain  $p$   $P$  where  $m: I ! m = \text{Inl } p \ m < \text{length } I \ I ! M = \text{Inr } P$   
 $M < \text{length } I$  by (*auto split: sum.splits*)  
 with  $wI$  have *wf-interp-for-formula* (*dec-word*  $x$ , *dec-interp*  $n \ \{m\} \ x$ ) (*FIn*  $m$   
 $M$ ) **unfolding**  $wI(1)$   
 using *enc-wf-interp*[*OF FIn(1)[folded wI(2)]*] by *auto*  
**moreover**  
 from  $x$  obtain  $u1 \ u \ u2$  where  $x = u1 \ @ \ u \ @ \ u2$   
 $u \in \text{lang } n \ (\text{Atom } (\text{Arbitrary-Except2 } m \ M))$   
**unfolding** *rexp-of.simps lang.simps rexp-of-list.simps* using *concE* by *fast*  
 with *FIn(1)* obtain  $v$  where  $v: x = u1 \ @ \ [v] \ @ \ u2 \ \text{snd } v ! m \ \text{snd } v ! M$   
 using *Arbitrary-Except2D*[*of u n m M*] by *fastforce*  
 from  $v$  have  $u: \text{length } u1 < \text{length } x$  by *auto*  
 { from  $v$  have  $\text{snd } (x ! \text{length } u1) ! m$  by *auto*  
**moreover**  
 from  $m \ wI$  have  $p < \text{length } x \ \text{snd } (x ! p) ! m$   
 by (*fastforce intro: nth-mem split: sum.splits*)+  
**moreover**  
 from  $m \ wI$  have  $ex1: \exists ! p. p < \text{length } x \wedge \text{snd } (x ! p) ! m$  **unfolding**  $wI(1)$   
 by (*intro enc-unique*) *auto*  
 ultimately have  $p = \text{length } u1$  using  $u$  by *auto*  
 } note  $*$  = *this*  
 from  $v$  have  $v = \text{enc } (w, I) ! \text{length } u1$  **unfolding**  $wI(1)$  by *simp*  
 with  $v(3) \ m(3,4) \ u \ wI(1)$  have  $\text{length } u1 \in P$  by *auto*  
 with  $* \ m \ wI$  have *satisfies* (*dec-word*  $x$ , *dec-interp*  $n \ \{m\} \ x$ ) (*FIn*  $m \ M$ )  
**unfolding** *dec-word-enc*[*of w I, folded wI(1)*]  
 by (*auto simp del: enc.simps dest: dec-interp-not-Inr dec-interp-not-Inl split:*  
*sum.splits*)  
 (*auto simp del: enc.simps dest!: dec-interp-enc-Inl dec-interp-enc-Inr dest:*  
*nth-mem split: sum.splits*)  
**moreover** from  $wI$  have *wf-word*  $n \ x$  **unfolding** *wf-word* by (*auto intro!:*  
*enc-atom- $\sigma$* )  
 ultimately show  $x \in \text{lang}_{M2L} \ n \ (FIn \ m \ M)$  **unfolding** *lang<sub>M2L</sub>-def* using  
 $m \ wI(3)$   
 by (*auto simp del: enc.simps intro!: exI[of - dec-word x] exI[of - dec-interp n*  
 $\{m\} \ x]$   
*intro: sym[OF enc-dec[OF - ballI[OF impI[OF enc-unique[of w I, folded*  
 $wI(1)]]]]]]$ )  
**qed**  
**next**  
**case** (*FOr*  $\varphi_1 \ \varphi_2$ )  
 from *FOr(3)* have *IH1*:  $\text{lang}_{M2L} \ n \ \varphi_1 = \text{lang } n \ (\text{rexp-of } n \ \varphi_1) - \{\ \}$   
 by (*intro FOr(1)*) *auto*  
 from *FOr(3)* have *IH2*:  $\text{lang}_{M2L} \ n \ \varphi_2 = \text{lang } n \ (\text{rexp-of } n \ \varphi_2) - \{\ \}$   
 by (*intro FOr(2)*) *auto*  
**show** *?case*  
**proof** (*intro equalityI subsetI*)  
 fix  $x$  assume  $x \in \text{lang}_{M2L} \ n \ (FOr \ \varphi_1 \ \varphi_2)$  **thus**  $x \in \text{lang } n \ (\text{rexp-of } n \ (FOr$   
 $\varphi_1 \ \varphi_2)) - \{\ \}$

```

    using langM2L-FOr[OF FOr(3)] unfolding lang-ENC-formula[OF FOr(3)]
  rexp-of.simps lang.simps
    IH1 IH2 Int-Diff by auto
  next
    fix x assume x ∈ lang n (rexp-of n (FOr φ1 φ2)) - {}
    then obtain w I where or: x ∈ langM2L n φ1 ∨ x ∈ langM2L n φ2 and wI:
  x = enc (w, I) length I = n
    wf-interp-for-formula (w, I) (FOr φ1 φ2)
    unfolding lang-ENC-formula[OF FOr(3)] rexp-of.simps lang.simps IH1 IH2
  Int-Diff by auto
    have satisfies (w, I) φ1 ∨ satisfies (w, I) φ2
    proof (intro mp[OF disj-mono[OF impI impI] or])
      assume x ∈ langM2L n φ1
      with wI(2,3) FOr(3) show satisfies (w, I) φ1
        unfolding langM2L-def wI(1) wf-interp-for-formula-FOr
        by (auto simp del: enc.simps dest!: iffD2[OF enc-welldef[of - - - φ1]])
    next
      assume x ∈ langM2L n φ2
      with wI(2,3) FOr(3) show satisfies (w, I) φ2
        unfolding langM2L-def wI(1) wf-interp-for-formula-FOr
        by (auto simp del: enc.simps dest!: iffD2[OF enc-welldef[of - - - φ2]])
    qed
    with wI show x ∈ langM2L n (FOr φ1 φ2) unfolding langM2L-def by auto
  qed
next
  case (FAnd φ1 φ2)
  from FAnd(3) have IH1: langM2L n φ1 = lang n (rexp-of n φ1) - {}
    by (intro FAnd(1)) auto
  from FAnd(3) have IH2: langM2L n φ2 = lang n (rexp-of n φ2) - {}
    by (intro FAnd(2)) auto
  show ?case
  proof (intro equalityI subsetI)
    fix x assume x ∈ langM2L n (FAnd φ1 φ2) thus x ∈ lang n (rexp-of n (FAnd
  φ1 φ2)) - {}
    using langM2L-FAnd[OF FAnd(3)] unfolding lang-ENC-formula[OF FAnd(3)]
  rexp-of.simps
    rexp-of-list.simps lang.simps IH1 IH2 Int-Diff by auto
  next
    fix x assume x ∈ lang n (rexp-of n (FAnd φ1 φ2)) - {}
    then obtain w I where and: x ∈ langM2L n φ1 ∧ x ∈ langM2L n φ2 and
  wI: x = enc (w, I) length I = n
    wf-interp-for-formula (w, I) (FAnd φ1 φ2)
    unfolding lang-ENC-formula[OF FAnd(3)] rexp-of.simps rexp-of-list.simps
  lang.simps IH1 IH2
    Int-Diff by auto
    have satisfies (w, I) φ1 ∧ satisfies (w, I) φ2
    proof (intro mp[OF conj-mono[OF impI impI] and])
      assume x ∈ langM2L n φ1
      with wI(2,3) FAnd(3) show satisfies (w, I) φ1

```

```

      unfolding langM2L-def wI(1) wf-interp-for-formula-FAnd
      by (auto simp del: enc.simps dest!: iffD2[OF enc-welldef[of - - - φ1]])
    next
      assume x ∈ langM2L n φ2
      with wI(2,3) FAnd(3) show satisfies (w, I) φ2
      unfolding langM2L-def wI(1) wf-interp-for-formula-FAnd
      by (auto simp del: enc.simps dest!: iffD2[OF enc-welldef[of - - - φ2]])
    qed
    with wI show x ∈ langM2L n (FAnd φ1 φ2) unfolding langM2L-def by auto
  qed
next
  case (FNot φ)
  hence IH: ?L n φ = langM2L n φ by simp
  show ?case
  proof (intro equalityI subsetI)
    fix x assume x ∈ langM2L n (FNot φ)
    then obtain w I where
      *: x = enc (w, I) wf-interp-for-formula (w, I) φ length I = n length w > 0
      and unsat: ¬ (satisfies (w, I) φ)
      unfolding langM2L-def by auto
    { assume x ∈ ?L n φ
      with IH have satisfies (w, I) φ using enc-welldef[of - - w I φ, OF - - -
      *(2)] FNot(2)
      unfolding *(1,3) langM2L-def by auto
    }
    with unsat have x ∉ ?L n φ by blast
    with * show x ∈ ?L n (FNot φ) unfolding rexp-of.simps lang.simps
      using lang-ENC-formula[OF FNot(2)] by (auto simp: comp-def intro!:
      enc-atom-σ)
  next
    fix x assume x ∈ ?L n (FNot φ)
    with IH have x ∈ lang n (ENC n (FOV (FNot φ))) - {} and x: x ∉
    langM2L n φ by (auto simp del: o-apply)
    then obtain w I where *: x = enc (w, I) wf-interp-for-formula (w, I) (FNot
    φ) length I = n
      unfolding lang-ENC-formula[OF FNot(2)] by blast
    { assume ¬ satisfies (w, I) (FNot φ)
      with * have x ∈ langM2L n φ unfolding langM2L-def by auto
    }
    with x * show x ∈ langM2L n (FNot φ) unfolding langM2L-def by blast
  qed
next
  case (FExists φ)
  show ?case
  proof (intro equalityI subsetI)
    fix x assume x ∈ langM2L n (FExists φ)
    then obtain w I p where
      *: x = enc (w, I) wf-interp-for-formula (w, I) (FExists φ)
      length I = n length w > 0 p ∈ {0 .. length w - 1} satisfies (w, Inl p # I) φ

```

**unfolding** *lang<sub>M2L</sub>-def* **by** *auto*  
**with** *FExists(2)* **have**  $enc(w, Inl\ p \# I) \in ?L(Suc\ n)\ \varphi$   
**by** (*intro subsetD[OF equalityD1[OF FExists(1)], of Suc n enc(w, Inl p # I)]*)  
*(auto simp: lang<sub>M2L</sub>-def nth-Cons' ord-less-eq-trans[OF le-imp-less-Suc Suc-pred[OF \*(4)]]*  
*split: if-split-asm sum.splits intro!: exI[of - w] exI[of - Inl p # I])*  
**with** *\*(1)* **show**  $x \in ?L\ n\ (FExists\ \varphi)$   
**by** (*auto simp: map-index intro!: image-eqI[of - map  $\pi$ ] simp del: o-apply*)  
*(auto simp:  $\pi$ -def)*  
**next**  
**fix**  $x$  **assume**  $x \in ?L\ n\ (FExists\ \varphi)$   
**then obtain**  $x'$  **where**  $x = map\ \pi\ x'$  **and**  $x' \in ?L(Suc\ n)\ \varphi$  **by** (*auto simp del: o-apply*)  
**with** *FExists(2)* **have**  $x' \in lang_{M2L}(Suc\ n)\ \varphi$   
**by** (*intro subsetD[OF equalityD2[OF FExists(1)], of Suc n x']*)  
*(auto split: if-split-asm sum.splits)*  
**then obtain**  $w\ I'$  **where**  
 $*$ :  $x' = enc(w, I')$  *wf-interp-for-formula*  $(w, I')\ \varphi$  *length*  $I' = Suc\ n$  *satisfies*  
 $(w, I')\ \varphi$   
**unfolding** *lang<sub>M2L</sub>-def* **by** *auto*  
**moreover then obtain**  $I_0\ I$  **where**  $I' = I_0 \# I$  **by** (*cases*  $I'$ ) *auto*  
**moreover with** *FExists(2)* *\*(2)* **obtain**  $p$  **where**  $I_0 = Inl\ p$   $p < length\ w$   
**by** (*auto simp: nth-Cons' split: sum.splits if-split-asm*)  
**ultimately have**  $x = enc(w, I)$  *wf-interp-for-formula*  $(w, I)\ (FExists\ \varphi)$   
*length*  $I = n$   
*length*  $w > 0$  *satisfies*  $(w, I)\ (FExists\ \varphi)$  **using** *FExists(2)* **unfolding**  $x$   
**by** (*auto simp: map-tl nth-Cons' split: if-split-asm simp del: o-apply*) (*auto simp:  $\pi$ -def*)  
**thus**  $x \in lang_{M2L}\ n\ (FExists\ \varphi)$  **unfolding** *lang<sub>M2L</sub>-def* **by** (*auto intro!: exI[of - w] exI[of - I]*)  
**qed**  
**next**  
**case** (*FEXISTS*  $\varphi$ )  
**show** *?case*  
**proof** (*intro equalityI subsetI*)  
**fix**  $x$  **assume**  $x \in lang_{M2L}\ n\ (FEXISTS\ \varphi)$   
**then obtain**  $w\ I\ P$  **where**  
 $*$ :  $x = enc(w, I)$  *wf-interp-for-formula*  $(w, I)\ (FEXISTS\ \varphi)$   
*length*  $I = n$  *length*  $w > 0$   $P \subseteq \{0 .. length\ w - 1\}$  *satisfies*  $(w, Inr\ P \# I)$   
 $\varphi$   
**unfolding** *lang<sub>M2L</sub>-def* **by** *auto*  
**from** *\*(4,5)* **have**  $\forall p \in P. p < length\ w$  **by** (*cases*  $w$ ) *auto*  
**with** *\*(2-4,6)* *FEXISTS(2)* **have**  $enc(w, Inr\ P \# I) \in ?L(Suc\ n)\ \varphi$   
**by** (*intro subsetD[OF equalityD1[OF FEXISTS(1)], of Suc n enc(w, Inr P # I)]*)  
*(auto simp: lang<sub>M2L</sub>-def nth-Cons' split: if-split-asm sum.splits intro!: exI[of - w] exI[of - Inr P # I])*  
**with** *\*(1)* **show**  $x \in ?L\ n\ (FEXISTS\ \varphi)$

**by** (*auto simp: map-index intro!: image-eqI[of - map  $\pi$ ] simp del: o-apply*)  
(*auto simp:  $\pi$ -def*)  
**next**  
**fix**  $x$  **assume**  $x \in ?L\ n$  (*FEXISTS  $\varphi$* )  
**then obtain**  $x'$  **where**  $x: x = \text{map } \pi\ x'$  **and**  $x': \text{length } x' > 0$  **and**  $x' \in ?L$   
(*Suc n*)  $\varphi$  **by** (*auto simp del: o-apply*)  
**with** *FEXISTS(2)* **have**  $x' \in \text{lang}_{M2L}\ (\text{Suc } n)\ \varphi$   
**by** (*intro subsetD[OF equalityD2[OF FEXISTS(1)], of Suc n x']*)  
(*auto split: if-split-asm sum.splits*)  
**then obtain**  $w\ I'$  **where**  
 $*$ :  $x' = \text{enc } (w, I')$  *wf-interp-for-formula*  $(w, I')\ \varphi$   $\text{length } I' = \text{Suc } n$  *satisfies*  
 $(w, I')\ \varphi$   
**unfolding** *lang<sub>M2L</sub>-def* **by** *auto*  
**moreover then obtain**  $I_0\ I$  **where**  $I' = I_0 \# I$  **by** (*cases I'*) *auto*  
**moreover with** *FEXISTS(2) \*(2)* **obtain**  $P$  **where**  $I_0 = \text{Inr } P$   
**by** (*auto simp: nth-Cons' split: sum.splits if-split-asm*)  
**moreover have**  $\text{length } w \geq 1$  **using**  $x' *(1)$  **by** (*cases w*) *auto*  
**ultimately have**  $x = \text{enc } (w, I)$  *wf-interp-for-formula*  $(w, I)$  (*FEXISTS  $\varphi$* )  
 $\text{length } I = n$   
 $\text{length } w > 0$  *satisfies*  $(w, I)$  (*FEXISTS  $\varphi$* ) **using** *FEXISTS(2)* **unfolding**  $x$   
**by** (*auto simp add: map-tl nth-Cons' split: if-split-asm*)  
(*intro!: exI[of - P] simp del: o-apply*) (*auto simp:  $\pi$ -def*)  
**thus**  $x \in \text{lang}_{M2L}\ n$  (*FEXISTS  $\varphi$* ) **unfolding** *lang<sub>M2L</sub>-def* **by** (*auto intro!:*  
*exI[of - w] exI[of - I]*)  
**qed**  
**qed**

**lemma** *wf-rexp-of: wf-formula n  $\varphi \implies \text{wf } n$  (rexp-of n  $\varphi$ )*  
**by** (*induct  $\varphi$  arbitrary: n*) (*auto intro: wf-rexp-ENC simp: finite-FOV max-idx-vars*)

**lemma** *wf-rexp-of-alt: wf-formula n  $\varphi \implies \text{wf } n$  (rexp-of-alt n  $\varphi$ )*  
**by** (*induct  $\varphi$  arbitrary: n*) (*auto simp: wf-rexp-ENC finite-FOV max-idx-vars*)

**lemma** *wf-rexp-of': wf-formula n  $\varphi \implies \text{wf } n$  (rexp-of' n  $\varphi$ )*  
**unfolding** *rexp-of'-def* **by** (*auto simp: wf-rexp-ENC wf-rexp-of-alt finite-FOV*  
*max-idx-vars*)

**lemma** *wf-rexp-of-alt': wf-formula n  $\varphi \implies \text{wf } n$  (rexp-of-alt' n  $\varphi$ )*  
**by** (*induct  $\varphi$  arbitrary: n*) (*auto simp: wf-rexp-ENC*)

**lemma** *wf-rexp-of'': wf-formula n  $\varphi \implies \text{wf } n$  (rexp-of'' n  $\varphi$ )*  
**unfolding** *rexp-of''-def* **by** (*auto simp: wf-rexp-ENC wf-rexp-of-alt' finite-FOV*  
*max-idx-vars*)

**lemma** *ENC-Not: ENC n (FOV (FNot  $\varphi$ )) = ENC n (FOV  $\varphi$ )*  
**unfolding** *ENC-def* **by** *auto*

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

$n \text{ (ENC } n \text{ (FOV } \varphi)) \cap \text{lang } n \text{ (ENC } n \text{ (FOV } \psi)) - \{\emptyset\}$

**proof**

**fix**  $x$  **assume**  $wf: wf\text{-formula } n \text{ (FAnd } \varphi \psi)$  **and**  $x: x \in \text{lang } n \text{ (ENC } n \text{ (FOV (FAnd } \varphi \psi))) - \{\emptyset\}$

**hence**  $wf1: wf\text{-formula } n \varphi$  **and**  $wf2: wf\text{-formula } n \psi$  **by auto**

**from**  $x$  **obtain**  $w I$  **where**  $wI: x = \text{enc } (w, I) wf\text{-interp-for-formula } (w, I) \text{ (FAnd } \varphi \psi) \text{ length } I = n$

**using**  $\text{lang-ENC-formula}[OF wf]$  **by blast**

**hence**  $wf\text{-interp-for-formula } (w, I) \varphi wf\text{-interp-for-formula } (w, I) \psi$

**unfolding**  $wf\text{-interp-for-formula-FAnd}$  **by auto**

**hence**  $x \in (\text{lang } n \text{ (ENC } n \text{ (FOV } \varphi)) - \{\emptyset\}) \cap (\text{lang } n \text{ (ENC } n \text{ (FOV } \psi)) - \{\emptyset\})$

**unfolding**  $\text{lang-ENC-formula}[OF wf1]$   $\text{lang-ENC-formula}[OF wf2]$  **using**  $wI$  **by auto**

**thus**  $x \in \text{lang } n \text{ (ENC } n \text{ (FOV } \varphi)) \cap \text{lang } n \text{ (ENC } n \text{ (FOV } \psi)) - \{\emptyset\}$  **by blast**  
**qed**

**lemma** *ENC-Or*:

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

**proof**

**fix**  $x$  **assume**  $wf: wf\text{-formula } n \text{ (FOr } \varphi \psi)$  **and**  $x: x \in \text{lang } n \text{ (ENC } n \text{ (FOV (FOr } \varphi \psi))) - \{\emptyset\}$

**hence**  $wf1: wf\text{-formula } n \varphi$  **and**  $wf2: wf\text{-formula } n \psi$  **by auto**

**from**  $x$  **obtain**  $w I$  **where**  $wI: x = \text{enc } (w, I) wf\text{-interp-for-formula } (w, I) \text{ (FOr } \varphi \psi) \text{ length } I = n$

**using**  $\text{lang-ENC-formula}[OF wf]$  **by blast**

**hence**  $wf\text{-interp-for-formula } (w, I) \varphi wf\text{-interp-for-formula } (w, I) \psi$

**unfolding**  $wf\text{-interp-for-formula-FOr}$  **by auto**

**hence**  $x \in (\text{lang } n \text{ (ENC } n \text{ (FOV } \varphi)) - \{\emptyset\}) \cap (\text{lang } n \text{ (ENC } n \text{ (FOV } \psi)) - \{\emptyset\})$

**unfolding**  $\text{lang-ENC-formula}[OF wf1]$   $\text{lang-ENC-formula}[OF wf2]$  **using**  $wI$  **by auto**

**thus**  $x \in \text{lang } n \text{ (ENC } n \text{ (FOV } \varphi)) \cap \text{lang } n \text{ (ENC } n \text{ (FOV } \psi)) - \{\emptyset\}$  **by blast**  
**qed**

**lemma** *project-enc*:  $\text{map } \pi \text{ (enc } (w, x \# I)) = \text{enc } (w, I)$

**unfolding**  $\pi\text{-def}$  **by auto**

**lemma** *list-list-eqI*:

**assumes**  $\forall (-, x) \in \text{set } xs. x \neq [] \forall (-, y) \in \text{set } ys. y \neq []$

$\text{map } (\lambda(-, x). \text{hd } x) xs = \text{map } (\lambda(-, x). \text{hd } x) ys \text{ map } \pi xs = \text{map } \pi ys$

**shows**  $xs = ys$

**proof** –

**from**  $\text{assms}(4)$  **have**  $\text{length } xs = \text{length } ys$  **by**  $(\text{metis length-map})$

**then show**  $?thesis$  **using**  $\text{assms}$  **by**  $(\text{induct rule: list-induct2})$   $(\text{auto simp: } \pi\text{-def neq-Nil-conv})$

**qed**

**lemma** *project-enc-extend*:  
**assumes**  $\text{map } \pi \ x = \text{enc } (w, I) \ \forall (-, x) \in \text{set } x. \ x \neq []$   
**shows**  $x = \text{enc } (w, \text{Inr } (\text{positions-in-row } x \ 0) \ # \ I)$   
**proof** –  
**from**  $\text{arg-cong}[\text{OF } \text{assms}(1), \text{ of } \text{map } \text{fst}]$  **have**  $w: w = \text{map } \text{fst } x$  **by**  $(\text{auto } \text{simp}: \pi\text{-def})$   
**show** *?thesis*  
**proof**  $(\text{rule } \text{list-list-eqI}[\text{OF } \text{assms}(2)], \text{ unfold } \text{project-enc})$   
**show**  $\text{map } (\lambda(-, x). \text{hd } x) \ x = \text{map } (\lambda(-, x). \text{hd } x) (\text{enc } (w, \text{Inr } (\text{positions-in-row } x \ 0) \ # \ I))$   
**using**  $\text{assms}(2)$  **unfolding**  $\text{enc.simps } \text{map-index } \text{positions-in-row } w$   
**by**  $(\text{intro } \text{nth-equalityI}) (\text{auto } \text{dest!}: \text{nth-mem } \text{simp}: \text{hd-conv-nth})$   
**qed**  $(\text{auto } \text{simp}: \text{assms}(1))$   
**qed**

**lemma** *ENC-Exists*:  
 $\text{wf-formula } n \ (\text{FExists } \varphi) \implies \text{lang } n \ (\text{ENC } n \ (\text{FOV } (\text{FExists } \varphi))) - \{\{\}\} = \text{map } \pi \ ' \ \text{lang } (\text{Suc } n) \ (\text{ENC } (\text{Suc } n) \ (\text{FOV } \varphi)) - \{\{\}\}$   
**proof**  $(\text{intro } \text{equalityI } \text{subsetI})$   
**fix**  $x$  **assume**  $\text{wf}: \text{wf-formula } n \ (\text{FExists } \varphi)$  **and**  $x: x \in \text{lang } n \ (\text{ENC } n \ (\text{FOV } (\text{FExists } \varphi))) - \{\{\}\}$   
**hence**  $\text{wf1}: \text{wf-formula } (\text{Suc } n) \ \varphi$  **by** *auto*  
**from**  $x$  **obtain**  $w \ I$  **where**  $wI: x = \text{enc } (w, I) \ \text{wf-interp-for-formula } (w, I) \ (\text{FExists } \varphi) \ \text{length } I = n$   
**using**  $\text{lang-ENC-formula}[\text{OF } \text{wf}]$  **by** *blast*  
**with**  $x$  **have**  $w \neq []$  **by**  $(\text{cases } w) \ \text{auto}$   
**from**  $wI(2)$  **obtain**  $p$  **where**  $p < \text{length } w \ \text{wf-interp-for-formula } (w, \text{Inl } p \ # \ I) \ \varphi$   
**using**  $\text{wf-interp-for-formula-FExists}[\text{OF } \text{wf}[\text{folded } wI(3)] \ \langle w \neq [] \rangle]$  **by** *auto*  
**with**  $wI(3)$  **have**  $x \in \text{map } \pi \ ' \ (\text{lang } (\text{Suc } n) \ (\text{ENC } (\text{Suc } n) \ (\text{FOV } \varphi)) - \{\{\}\})$   
**unfolding**  $wI(1) \ \text{lang-ENC-formula}[\text{OF } \text{wf1}] \ \text{project-enc}[\text{symmetric}, \text{ of } w \ I \ \text{Inl } p]$   
**by**  $(\text{intro } \text{imageI } \text{CollectI } \text{exI}[\text{of } - \ w] \ \text{exI}[\text{of } - \ \text{Inl } p \ # \ I]) \ \text{auto}$   
**thus**  $x \in \text{map } \pi \ ' \ \text{lang } (\text{Suc } n) \ (\text{ENC } (\text{Suc } n) \ (\text{FOV } \varphi)) - \{\{\}\}$  **by** *blast*  
**next**  
**fix**  $x$  **assume**  $\text{wf}: \text{wf-formula } n \ (\text{FExists } \varphi)$  **and**  $x \in \text{map } \pi \ ' \ \text{lang } (\text{Suc } n) \ (\text{ENC } (\text{Suc } n) \ (\text{FOV } \varphi)) - \{\{\}\}$   
**hence**  $\text{wf1}: \text{wf-formula } (\text{Suc } n) \ \varphi$  **and**  $0 \in \text{FOV } \varphi$  **and**  $x: x \in \text{map } \pi \ ' \ (\text{lang } (\text{Suc } n) \ (\text{ENC } (\text{Suc } n) \ (\text{FOV } \varphi)) - \{\{\}\})$  **by** *auto*  
**from**  $x$  **obtain**  $w \ I$  **where**  $wI: x = \text{map } \pi \ (\text{enc } (w, I)) \ \text{wf-interp-for-formula } (w, I) \ \varphi \ \text{length } I = \text{Suc } n$   
**using**  $\text{lang-ENC-formula}[\text{OF } \text{wf1}]$  **by** *auto*  
**with**  $\langle 0 \in \text{FOV } \varphi \rangle$  **obtain**  $p \ I'$  **where**  $I: I = \text{Inl } p \ # \ I'$  **by**  $(\text{cases } I) \ (\text{fastforce } \text{split}: \text{sum.splits})+$   
**with**  $wI$  **have**  $wI': x = \text{enc } (w, I') \ \text{length } I' = n$  **unfolding**  $\pi\text{-def}$  **by** *auto*  
**with**  $x$  **have**  $w \neq []$  **by**  $(\text{cases } w) \ \text{auto}$   
**have**  $\text{wf-interp-for-formula } (w, I') \ (\text{FExists } \varphi)$   
**using**  $\text{wf-interp-for-formula-FExists}[\text{OF } \text{wf}[\text{folded } wI(2)] \ \langle w \neq [] \rangle]$   
 $\text{wf-interp-for-formula-any-Inl}[\text{OF } wI(2)[\text{unfolded } I]] \ ..$



**with** *wtlI* **show**  $x \in \text{lang } n \text{ (ENC } n \text{ (FOV (FExists } \varphi))) - \{\square\}$  **unfolding**  
*lang-ENC-formula[OF wf]* **by** *blast*  
**qed**

**lemma** *ENC-EXISTS*:

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

**proof** (*intro equalityI subsetI*)

**fix**  $x$  **assume** *wf*: *wf-formula*  $n \text{ (FEXISTS } \varphi)$  **and**  $x: x \in \text{lang } n \text{ (ENC } n \text{ (FOV (FEXISTS } \varphi))) - \{\square\}$

**hence** *wf1*: *wf-formula* (Suc  $n$ )  $\varphi$  **by** *auto*

**from**  $x$  **obtain**  $w$   $I$  **where** *wI*:  $x = \text{enc } (w, I) \text{ wf-interp-for-formula } (w, I)$   
*(FEXISTS } \varphi) \text{ length } I = n*

**using** *lang-ENC-formula[OF wf]* **by** *blast*

**with**  $x$  **have**  $w \neq \square$  **by** (*cases w*) *auto*

**from** *wI*(2) **obtain**  $P$  **where**  $\forall p \in P. p < \text{length } w \text{ wf-interp-for-formula } (w,$   
 $\text{Inr } P \# I) \varphi$

**using** *wf-interp-for-formula-FEXISTS[OF wf[folded wI(3)]*  $\langle w \neq \square \rangle$  **by** *auto*

**with** *wI*(3) **have**  $x \in \text{map } \pi \text{ ' (lang (Suc } n \text{) (ENC (Suc } n \text{) (FOV } \varphi)) - \{\square\})$

**unfolding** *wI*(1) *lang-ENC-formula[OF wf1]* *project-enc[symmetric, of w I Inr*  
 $P]$

**by** (*intro imageI CollectI exI[of - w] exI[of - Inr P # I]*) *auto*

**thus**  $x \in \text{map } \pi \text{ ' lang (Suc } n \text{) (ENC (Suc } n \text{) (FOV } \varphi)) - \{\square\}$  **by** *blast*

**next**

**fix**  $x$  **assume** *wf*: *wf-formula*  $n \text{ (FEXISTS } \varphi)$  **and**  $x \in \text{map } \pi \text{ ' lang (Suc } n \text{) (ENC (Suc } n \text{) (FOV } \varphi)) - \{\square\}$

**hence** *wf1*: *wf-formula* (Suc  $n$ )  $\varphi$  **and**  $0 \in \text{SOV } \varphi$  **and**  $x: x \in \text{map } \pi \text{ ' (lang (Suc } n \text{) (ENC (Suc } n \text{) (FOV } \varphi)) - \{\square\})$  **by** *auto*

**from**  $x$  **obtain**  $w$   $I$  **where** *wI*:  $x = \text{map } \pi \text{ (enc } (w, I) \text{ wf-interp-for-formula (w, I) } \varphi \text{ length } I = \text{Suc } n$

**using** *lang-ENC-formula[OF wf1]* **by** *auto*

**with**  $\langle 0 \in \text{SOV } \varphi \rangle$  **obtain**  $P$   $I'$  **where**  $I: I = \text{Inr } P \# I'$  **by** (*cases I*) (*fastforce split: sum.splits*) $+$

**with** *wI* **have** *wtlI*:  $x = \text{enc } (w, I') \text{ length } I' = n$  **unfolding** *pi-def* **by** *auto*

**with**  $x$  **have**  $w \neq \square$  **by** (*cases w*) *auto*

**have** *wf-interp-for-formula* (w,  $I'$ ) (FEXISTS  $\varphi$ )

**using** *wf-interp-for-formula-FEXISTS[OF wf[folded wtlI(2)]*  $\langle w \neq \square \rangle$

*wf-interp-for-formula-any-Inr[OF wI(2)[unfolded I]] ..*

**with** *wtlI* **show**  $x \in \text{lang } n \text{ (ENC } n \text{ (FOV (FEXISTS } \varphi))) - \{\square\}$  **unfolding**  
*lang-ENC-formula[OF wf]* **by** *blast*

**qed**

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

**by** *auto*

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

*wf-formula*  $n \varphi \implies \text{lang } n \text{ (rexp-of } n \varphi) - \{\square\} = \text{lang } n \text{ (rexp-of' } n \varphi) - \{\square\}$

**unfolding** *rexp-of'-def* **proof** (*induction*  $\varphi$  *arbitrary: n*)

```

case (FNot  $\varphi$ )
hence wf-formula  $n$   $\varphi$  by simp
with FNot.IH show ?case unfolding rexp-of.simps rexp-of-alt.simps lang.simps
ENC-Not by blast
next
case (FAnd  $\varphi_1$   $\varphi_2$ )
hence wf1: wf-formula  $n$   $\varphi_1$  and wf2: wf-formula  $n$   $\varphi_2$  by force+
from FAnd.IH(1)[OF wf1] FAnd.IH(2)[OF wf2] show ?case using ENC-And[OF
FAnd.premis]
unfolding rexp-of.simps rexp-of-alt.simps lang.simps rexp-of-list.simps by blast
next
case (FOr  $\varphi_1$   $\varphi_2$ )
hence wf1: wf-formula  $n$   $\varphi_1$  and wf2: wf-formula  $n$   $\varphi_2$  by force+
from FOr.IH(1)[OF wf1] FOr.IH(2)[OF wf2] show ?case using ENC-Or[OF
FOr.premis]
unfolding rexp-of.simps rexp-of-alt.simps lang.simps by blast
next
case (FExists  $\varphi$ )
hence wf: wf-formula  $(n + 1)$   $\varphi$  by auto
show ?case using ENC-Exists[OF FExists.premis]
unfolding rexp-of.simps rexp-of-alt.simps lang.simps map-project-empty FEx-
ists.IH[OF wf] by auto
next
case (FEXISTS  $\varphi$ )
hence wf: wf-formula  $(n + 1)$   $\varphi$  by auto
show ?case using ENC-EXISTS[OF FEXISTS.premis]
unfolding rexp-of.simps rexp-of-alt.simps lang.simps map-project-empty FEX-
ISTS.IH[OF wf] by auto
qed auto

```

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

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

```

assumes finite  $X$   $X = Y1 \cup Y2$   $n = 0 \vee (\forall p \in X. p < n)$ 
shows lang  $n$  (ENC  $n$   $X$ ) = lang  $n$  (ENC  $n$   $Y1$ )  $\cap$  lang  $n$  (ENC  $n$   $Y2$ )
unfolding ENC-def lang-INTERSECT using assms lang-subset-lists[OF wf-rexp-valid-ENC,
of n] by auto

```

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

```

**proof** –

```

let ? $Y = \{0 ..< n + 1\} - X$ 
let ? $fX = (\lambda x. x - 1) ' X$ 
let ? $fY = \{0 ..< n\} - (\lambda x. x - 1) ' X$ 
from assms have *:  $(\lambda x. x - 1) ' X \subseteq \{0 ..< n\}$  by (cases  $n$ ) auto
show ?thesis unfolding Int-Diff lang-ENC[OF assms(2) subset-refl] lang-ENC[OF

```

```

* subset-refl]
proof (safe elim!: imageI)
  fix w I
  assume *: length I = n + 1 w ≠ []
    ∀ i ∈ X. case I ! i of Inl x ⇒ True | Inr x ⇒ False
    ∀ i ∈ ?Y. case I ! i of Inl x ⇒ False | Inr x ⇒ True
    ∀ a ∈ set w. a ∈ set Σ Ball (set I) (case-sum (λp. p < length w) (λP. ∀ p ∈ P.
p < length w))
  then obtain p Is where I = p # Is by (cases I) auto
  then show ∃ w' I'.
    map π (enc (w, I)) = enc (w', I') ∧
    length I' = n ∧ (0 < length w' ∧ (∀ a ∈ set w'. a ∈ set Σ) ∧
    Ball (set I') (case-sum (λp. p < length w') (λP. ∀ p ∈ P. p < length w'))) ∧
    (∀ i ∈ ?fX. case I' ! i of Inl x ⇒ True | Inr x ⇒ False) ∧
    (∀ i ∈ ?fY. case I' ! i of Inl x ⇒ False | Inr x ⇒ True)
  proof (hypsubst, intro exI[of - w] exI[of - Is] conjI ballI project-enc)
  fix i assume i ∈ ?fY
  then show case Is ! i of Inl x ⇒ False | Inr x ⇒ True
  using *[unfolded ⟨I = p # Is⟩] assms(1)
  by (cases i = 0) (fastforce simp: nth-Cons' image-iff split: sum.splits
if-splits)+
  qed (insert *[unfolded ⟨I = p # Is⟩] assms(1), auto simp: nth-Cons' split:
sum.splits if-splits)
  next
  fix x w I
  assume *: w ≠ [] x ∈ Z map π x = enc (w, I)
    ∀ i ∈ ?fX. case I ! i of Inl x ⇒ True | Inr x ⇒ False
    ∀ i ∈ {0 ..< length I} - ?fX. case I ! i of Inl x ⇒ False | Inr x ⇒ True
    ∀ a ∈ set w. a ∈ set Σ Ball (set I) (case-sum (λp. p < length w) (λP. ∀ p ∈ P.
p < length w))
  moreover from assms(1) have ∀ x ∈ X. x > 0 ∧ x y. x - Suc 0 = y - Suc
0 ⟷
    x = y ∨ (x = 0 ∧ y = Suc 0) ∨ (x = Suc 0 ∧ y = 0)
  by (metis neq0-conv) (metis One-nat-def Suc-diff-1 diff-0-eq-0 diff-self-eq-0
neq0-conv)
  moreover from *(2) assms(3) have x = enc (w, Inr (positions-in-row x 0)
# I)
  apply (intro project-enc-extend [OF *(3)])
  apply (simp only: σ-def)
  apply auto
  done
  moreover from arg-cong[OF *(3), of length] have length w = length x by
simp
  ultimately show map π x ∈ map π ‘
    (Z ∩ {enc (w, I') | w I'. length I' = length I + 1 ∧ (0 < length w ∧ (∀ a ∈ set
w. a ∈ set Σ) ∧
    Ball (set I') (case-sum (λp. p < length w) (λP. ∀ p ∈ P. p < length w))) ∧
    (∀ i ∈ X. case I' ! i of Inl x ⇒ True | Inr x ⇒ False) ∧
    (∀ i ∈ {0 ..< length I + 1} - X. case I' ! i of Inl x ⇒ False | Inr x ⇒

```

*True*))  
**by** (*intro imageI CollectI conjI IntI exI[of - w] exI[of - Inr (positions-in-row x 0) # I]*)  
*(auto simp: nth-Cons' positions-in-row elim!: bspec simp del: enc.simps)*  
**qed**  
**qed**

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

**assumes**  $X \subseteq \{0 \dots n + 1\}$   $Z \subseteq \text{lists } ((\text{set } o \sigma \Sigma) (n + 1))$   
**shows**  $\text{map } \pi \text{ ' } (Z \cap \text{lang } (n + 1) (\text{ENC } (n + 1) X) - \{\}) =$   
*(if*  $0 \in X$   
*then*  $\text{map } \pi \text{ ' } (Z \cap \text{lang } (n + 1) (\text{ENC } (n + 1) \{0\})) \cap \text{lang } n (\text{ENC } n ((\lambda x. x - 1) \text{ ' } (X - \{0\}))) - \{\}$   
*else*  $\text{map } \pi \text{ ' } Z \cap \text{lang } n (\text{ENC } n ((\lambda x. x - 1) \text{ ' } (X - \{0\}))) - \{\}$   
**is**  $?L = (\text{if - then ?R1 else ?R2})$

**proof** (*split if-splits, intro conjI impI*)

**assume**  $0: 0 \notin X$

**from** *assms* **have** *fin: finite X finite ((λx. x - 1) ' X)*

**by** (*auto elim: finite-subset intro!: finite-imageI[of X]*)

**from**  $0$  **show**  $?L = ?R2$  **using** *map-project-Int-ENC[OF 0 assms]*

**unfolding** *lists-image[symmetric] π-σ*

*Int-absorb1[OF lang-subset-lists[OF wf-rexp-ENC[OF fin(1)], of n + 1]*

*Int-absorb1[OF lang-subset-lists[OF wf-rexp-ENC[OF fin(2)], of n]*

**by** *auto*

**next**

**assume**  $0 \in X$

**hence**  $0: 0 \notin X - \{0\}$  **and**  $X: X = \{0\} \cup (X - \{0\})$  **by** *auto*

**from** *assms* **have** *fin: finite X*

**by** (*auto elim: finite-subset intro!: finite-imageI[of X]*)

**have**  $?L = \text{map } \pi \text{ ' } ((Z \cap \text{lang } (n + 1) (\text{ENC } (n + 1) \{0\})) \cap \text{lang } (n + 1) (\text{ENC } (n + 1) (X - \{0\}))) - \{\}$

**unfolding** *Int-assoc* **using** *assms* **by** (*subst lang-ENC-split[OF fin X, of n + 1] auto*)

**also** **have**  $\dots = ?R1$

**using**  $0$  *assms* **by** (*elim map-project-Int-ENC*) *auto*

**finally** **show**  $?L = ?R1$  .

**qed**

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

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

*wf-formula n φ ⇒ lang n (rexp-of' n φ) - {\} = lang n (rexp-of'' n φ) - {\}*

**unfolding** *rexp-of'-def rexp-of''-def*

**proof** (*induction φ arbitrary: n*)

**case** (*FNot φ*)

**hence** *wf-formula n φ* **by** *simp*

**with** *FNot.IH* **show** *?case* **unfolding** *rexp-of-alt.simps rexp-of-alt'.simps lang.simps ENC-Not* **by** *blast*

**next**

```

case (FAnd  $\varphi_1$   $\varphi_2$ )
hence wf1: wf-formula  $n$   $\varphi_1$  and wf2: wf-formula  $n$   $\varphi_2$  by force+
from FAnd.IH(1)[OF wf1] FAnd.IH(2)[OF wf2] show ?case using ENC-And[OF FAnd.prems]
  unfolding rexp-of-alt.simps rexp-of-alt'.simps lang.simps rexp-of-list.simps by
  blast
next
  case (FOr  $\varphi_1$   $\varphi_2$ )
  hence wf1: wf-formula  $n$   $\varphi_1$  and wf2: wf-formula  $n$   $\varphi_2$  by force+
  from FOr.IH(1)[OF wf1] FOr.IH(2)[OF wf2] show ?case using ENC-Or[OF FOr.prems]
  unfolding rexp-of-alt.simps rexp-of-alt'.simps lang.simps rexp-of-list.simps by
  blast
next
  case (FExists  $\varphi$ )
  hence wf: wf-formula  $(n + 1)$   $\varphi$  and  $0$ :  $0 \in \text{FOV } \varphi$  by auto
  then show ?case
    using ENC-Exists[OF FExists.prems] map-project-ENC[of FOV  $\varphi$   $n$ ] max-idx-vars[of
     $n + 1$   $\varphi$ ]
    wf-rexp-of-alt'[OF wf]  $0$ 
    unfolding rexp-of-alt.simps rexp-of-alt'.simps lang.simps Int-Diff-both
    unfolding map-project-empty FExists.IH[OF wf, unfolded lang.simps]
    by (intro trans[OF arg-cong2[of - - - ( $\cap$ ), OF map-project-ENC[OF - lang-subset-lists]
    refl]])
    fastforce+
next
  case (FEXISTS  $\varphi$ )
  hence wf: wf-formula  $(n + 1)$   $\varphi$  and  $0$ :  $0 \notin \text{FOV } \varphi$  by auto
  then show ?case
    using ENC-EXISTS[OF FEXISTS.prems] map-project-ENC[of FOV  $\varphi$   $n$ ]
    max-idx-vars[of  $n + 1$   $\varphi$ ]
    wf-rexp-of-alt'[OF wf]  $0$ 
    unfolding rexp-of-alt.simps rexp-of-alt'.simps lang.simps Int-Diff-both
    unfolding map-project-empty FEXISTS.IH[OF wf, unfolded lang.simps]
    by (intro trans[OF arg-cong2[of - - - ( $\cap$ ), OF map-project-ENC[OF - lang-subset-lists]
    refl]])
    fastforce+
qed simp-all

theorem langM2L-rexp-of': wf-formula  $n$   $\varphi \implies \text{lang}_{M2L} n \varphi = \text{lang } n (\text{rexp-of}'$ 
 $n \varphi) - \{\emptyset\}$ 
  unfolding langM2L-rexp-of-rexp-of'[symmetric] by (rule langM2L-rexp-of')

theorem langM2L-rexp-of'': wf-formula  $n$   $\varphi \implies \text{lang}_{M2L} n \varphi = \text{lang } n (\text{rexp-of}''$ 
 $n \varphi) - \{\emptyset\}$ 
  unfolding langM2L-rexp-of'-rexp-of''[symmetric] by (rule langM2L-rexp-of')

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]*:  $satisfies (w, I) (nNot \varphi) = satisfies (w, I) (FNot \varphi)$   
**by** (*induct*  $\varphi$  *rule: nNot.induct*) *auto*

**lemma** *FOV-nNot[simp]*:  $FOV (nNot \varphi) = FOV (FNot \varphi)$   
**by** (*induct*  $\varphi$  *rule: nNot.induct*) *auto*

**lemma** *SOV-nNot[simp]*:  $SOV (nNot \varphi) = SOV (FNot \varphi)$   
**by** (*induct*  $\varphi$  *rule: nNot.induct*) *auto*

**lemma** *pre-wf-formula-nNot[simp]*:  $pre-wf-formula n (nNot \varphi) = pre-wf-formula n (FNot \varphi)$   
**by** (*induct*  $\varphi$  *rule: nNot.induct*) *auto*

**lemma** *FOV-norm[simp]*:  $FOV (norm \varphi) = FOV \varphi$   
**by** (*induct*  $\varphi$ ) *auto*

**lemma** *SOV-norm[simp]*:  $SOV (norm \varphi) = SOV \varphi$   
**by** (*induct*  $\varphi$ ) *auto*

**lemma** *pre-wf-formula-norm[simp]*:  $pre-wf-formula n (norm \varphi) = pre-wf-formula n \varphi$   
**by** (*induct*  $\varphi$  *arbitrary: n*) *auto*

**lemma** *satisfies-norm[simp]*:  $satisfies (w, I) (norm \varphi) = satisfies (w, I) \varphi$   
**by** (*induct*  $\varphi$  *arbitrary: I*) *auto*

**lemma** *lang<sub>M2L</sub>-norm[simp]*:  $lang_{M2L} n (norm \varphi) = lang_{M2L} n \varphi$

**unfolding** *lang<sub>M2L</sub>-def* **by** *auto*

**end**

## 12 Deciding Equivalence of M2L Formulas

**global-interpretation** *embed set o σ Σ wf-atom Σ π lookup ε Σ*

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

**defines**

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

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

**by** *unfold-locales (auto simp: σ-def π-def ε-def set-n-lists)*

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

**proof** (*rule notI*)

**assume**  $?enum = []$

**hence**  $\text{set } ?enum = \{\}$  **by** *simp*

**thus** *False* **unfolding** *UNIV-enum[symmetric]* **by** *simp*

**qed**

**global-interpretation**  $\Phi$ : *formula Enum.enum :: 'a :: {enum, 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**  $\text{rexp-of}''' = \Phi.\text{rexp-of}'''$

**and**  $\text{valid-ENC} = \Phi.\text{valid-ENC}$

**and**  $\text{ENC} = \Phi.\text{ENC}$

**and**  $\text{dec-interp} = \Phi.\text{dec-interp}$

**by** *unfold-locales (auto simp: σ-def π-def set-n-lists)*

**lemma** *lang-Plus-Zero: lang Σ n (Plus r One) = lang Σ n (Plus s One)  $\longleftrightarrow$  lang*

$\Sigma n r - \{\}\} = \text{lang } \Sigma n s - \{\}\}$

**by** *auto*

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

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

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

**setup**  $\langle \text{Sign.map-naming (Name-Space.mandatory-path slow)} \rangle$

**global-interpretation**  $D$ : *rexp-DFA σ Σ wf-atom Σ π lookup λx.  $\ll \text{pnorm (inorm } x) \gg$*

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

**for**  $\Sigma :: 'a :: \text{linorder list}$  **and**  $n :: \text{nat}$   
**defines**  
 $\text{test} = \text{rexp-DA.test}$  ( $\text{final} :: 'a \text{ atom rexp} \Rightarrow \text{bool}$ )  
**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**  $\text{check-equivRE} = \text{rexp-DA.check-equiv}$  ( $\sigma \Sigma$ ) ( $\lambda x. \llbracket \text{pnorm } (\text{inorm } x) \rrbracket$ ) ( $\lambda a r. \llbracket \mathfrak{D} \Sigma a r \rrbracket$ )  $\text{final pnorm } n$   
**and**  $\text{test-invariant} = \text{rexp-DA.test-invariant}$  ( $\text{final} :: 'a \text{ atom rexp} \Rightarrow \text{bool}$ ) ::  
 $(('a \times \text{bool list}) \text{ list} \times -) \text{ list} \times - \Rightarrow \text{bool}$   
**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**  $\text{counterexampleRE} = \text{rexp-DA.counterexample}$  ( $\sigma \Sigma$ ) ( $\lambda x. \llbracket \text{pnorm } (\text{inorm } x) \rrbracket$ ) ( $\lambda a r. \llbracket \mathfrak{D} \Sigma a r \rrbracket$ )  $\text{final pnorm } n$   
**and**  $\text{reachable} = \text{rexp-DA.reachable}$  ( $\sigma \Sigma$ ) ( $\lambda x. \llbracket \text{pnorm } (\text{inorm } x) \rrbracket$ ) ( $\lambda a r. \llbracket \mathfrak{D} \Sigma a r \rrbracket$ )  $\text{pnorm } n$   
**and**  $\text{automaton} = \text{rexp-DA.automaton}$  ( $\sigma \Sigma$ ) ( $\lambda x. \llbracket \text{pnorm } (\text{inorm } x) \rrbracket$ ) ( $\lambda a r. \llbracket \mathfrak{D} \Sigma a r \rrbracket$ )  $\text{pnorm } n$   
**by**  $\text{unfold-locales}$  ( $\text{auto simp only: comp-apply}$   
 $\text{ACI-norm-wf ACI-norm-lang wf-inorm lang-inorm wf-pnorm lang-pnorm wf-lderiv}$   
 $\text{lang-lderiv}$   
 $\text{lang-final finite-fold-lderiv dest!: lang-subset-lists}$ )

**definition check-equiv where**

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

**definition counterexample where**

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

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

**by** ( $\text{rule box-equals}[\text{OF iffD1}[\text{OF lang-Plus-Zero}, \text{OF slow.D.check-equiv-sound}]$   
 $\text{sym}[\text{OF trans}[\text{OF lang}_{M2L}\text{-rexp-of''-norm}]] \text{sym}[\text{OF trans}[\text{OF lang}_{M2L}\text{-rexp-of''-norm}]]]$ )  
 $(\text{auto simp: slow.check-equiv-def intro!: } \Phi.\text{wf-rexp-of''})$

**lemma completeness:**

**assumes**  $\Phi.\text{lang}_{M2L} n \varphi = \Phi.\text{lang}_{M2L} n \psi \text{ wf-formula } n (\text{FOr } \varphi \psi)$   
**shows**  $\text{slow.check-equiv } n \varphi \psi$   
**using**  $\text{assms}(2)$  **unfolding**  $\text{slow.check-equiv-def}$   
**by** ( $\text{intro conjI}[\text{OF assms}(2) \text{slow.D.check-equiv-complete}[\text{OF iffD2}[\text{OF lang-Plus-Zero}],$   
 $\text{OF box-equals}[\text{OF assms}(1) \text{lang}_{M2L}\text{-rexp-of''-norm lang}_{M2L}\text{-rexp-of''-norm}]]]$ )  
 $(\text{auto intro!: } \Phi.\text{wf-rexp-of''})$

**setup** ( $\text{Sign.map-naming Name-Space.parent-path}$ )



**setup**  $\langle \text{Sign.map-naming } (\text{Name-Space.mandatory-path fast}) \rangle$

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

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

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

**defines**

$\text{test} = \text{rexp-DA.test } (\text{final} :: 'a \text{ atom rexp} \Rightarrow \text{bool})$

**and**  $\text{step} = \text{rexp-DA.step } (\sigma \Sigma) (\lambda a r. \text{pnorm } (\mathfrak{D} \Sigma a r)) \text{ id } n$

**and**  $\text{closure} = \text{rexp-DA.closure } (\sigma \Sigma) (\lambda a r. \text{pnorm } (\mathfrak{D} \Sigma a r)) \text{ final id } n$

**and**  $\text{check-eqvRE} = \text{rexp-DA.check-eqv } (\sigma \Sigma) (\lambda x. \text{pnorm } (\text{inorm } x)) (\lambda a r. \text{pnorm } (\mathfrak{D} \Sigma a r)) \text{ final id } n$

**and**  $\text{test-invariant} = \text{rexp-DA.test-invariant } (\text{final} :: 'a \text{ atom rexp} \Rightarrow \text{bool}) ::$

$(('a \times \text{bool list}) \text{ list} \times -) \text{ list} \times - \Rightarrow \text{bool}$

**and**  $\text{step-invariant} = \text{rexp-DA.step-invariant } (\sigma \Sigma) (\lambda a r. \text{pnorm } (\mathfrak{D} \Sigma a r)) \text{ id } n$

**and**  $\text{closure-invariant} = \text{rexp-DA.closure-invariant } (\sigma \Sigma) (\lambda a r. \text{pnorm } (\mathfrak{D} \Sigma a r)) \text{ final id } n$

**and**  $\text{counterexampleRE} = \text{rexp-DA.counterexample } (\sigma \Sigma) (\lambda x. \text{pnorm } (\text{inorm } x)) (\lambda a r. \text{pnorm } (\mathfrak{D} \Sigma a r)) \text{ final id } n$

**and**  $\text{reachable} = \text{rexp-DA.reachable } (\sigma \Sigma) (\lambda x. \text{pnorm } (\text{inorm } x)) (\lambda a r. \text{pnorm } (\mathfrak{D} \Sigma a r)) \text{ id } n$

**and**  $\text{automaton} = \text{rexp-DA.automaton } (\sigma \Sigma) (\lambda x. \text{pnorm } (\text{inorm } x)) (\lambda a r. \text{pnorm } (\mathfrak{D} \Sigma a r)) \text{ id } n$

**by**  $\text{unfold-locales } (\text{auto simp only: comp-apply})$

$\text{ACI-norm-wf ACI-norm-lang wf-inorm lang-inorm wf-pnorm lang-pnorm wf-lderiv lang-lderiv id-apply}$

$\text{lang-final dest!: lang-subset-lists})$

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

$\text{check-eqv } n \varphi \psi \longleftrightarrow \text{wf-formula } n (\text{FOr } \varphi \psi) \wedge$

$\text{fast.check-eqvRE Enum.enum } n (\text{Plus } (\text{rexp-of'' } n (\text{norm } \varphi)) \text{ One}) (\text{Plus } (\text{rexp-of'' } n (\text{norm } \psi)) \text{ One})$

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

$\text{counterexample } n \varphi \psi =$

$\text{map-option } (\lambda w. \text{dec-interp } n (\text{FOV } (\text{FOr } \varphi \psi)) w)$

$(\text{fast.counterexampleRE Enum.enum } n (\text{Plus } (\text{rexp-of'' } n (\text{norm } \varphi)) \text{ One}) (\text{Plus } (\text{rexp-of'' } n (\text{norm } \psi)) \text{ One}))$

**lemma**  $\text{soundness: fast.check-eqv } n \varphi \psi \Longrightarrow \Phi.\text{lang}_{M2L} n \varphi = \Phi.\text{lang}_{M2L} n \psi$

**by**  $(\text{rule box-equals}[\text{OF iffD1}[\text{OF lang-Plus-Zero, OF fast.D.check-eqv-sound}]$

$\text{sym}[\text{OF trans}[\text{OF lang}_{M2L}\text{-rexp-of''-norm}]] \text{sym}[\text{OF trans}[\text{OF lang}_{M2L}\text{-rexp-of''-norm}]])$

$(\text{auto simp: fast.check-eqv-def intro!: } \Phi.\text{wf-rexp-of''})$

**setup**  $\langle \text{Sign.map-naming } (\text{Name-Space.parent-path}) \rangle$

**setup**  $\langle \text{Sign.map-naming } (\text{Name-Space.mandatory-path dual}) \rangle$

**global-interpretation** *D*: *rexp-DA-no-post*  $\sigma \Sigma$  *wf-atom*  $\Sigma \pi$  *lookup*  
 $\lambda x. \text{pnorm-dual } (\text{rexp-dual-of } (\text{inorm } x)) \lambda a r. \text{pnorm-dual } (\text{CoD } \Sigma a r)$  *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 atom rexp-dual*  $\Rightarrow$  *bool*)  
**and** *step* = *rexp-DA.step* ( $\sigma \Sigma$ ) ( $\lambda a r. \text{pnorm-dual } (\text{CoD } \Sigma a r)$ ) *id n*  
**and** *closure* = *rexp-DA.closure* ( $\sigma \Sigma$ ) ( $\lambda a r. \text{pnorm-dual } (\text{CoD } \Sigma a r)$ ) *final-dual*  
*id n*  
**and** *check-equivRE* = *rexp-DA.check-equiv* ( $\sigma \Sigma$ ) ( $\lambda x. \text{pnorm-dual } (\text{rexp-dual-of } (\text{inorm } x))$ ) ( $\lambda a r. \text{pnorm-dual } (\text{CoD } \Sigma a r)$ ) *final-dual id n*  
**and** *test-invariant* = *rexp-DA.test-invariant* (*final-dual* :: *'a atom rexp-dual*  $\Rightarrow$  *bool*) ::  
(*'a*  $\times$  *bool list*) *list*  $\times$  -) *list*  $\times$  -  $\Rightarrow$  *bool*  
**and** *step-invariant* = *rexp-DA.step-invariant* ( $\sigma \Sigma$ ) ( $\lambda a r. \text{pnorm-dual } (\text{CoD } \Sigma a r)$ ) *id n*  
**and** *closure-invariant* = *rexp-DA.closure-invariant* ( $\sigma \Sigma$ ) ( $\lambda a r. \text{pnorm-dual } (\text{CoD } \Sigma a r)$ ) *final-dual id n*  
**and** *counterexampleRE* = *rexp-DA.counterexample* ( $\sigma \Sigma$ ) ( $\lambda x. \text{pnorm-dual } (\text{rexp-dual-of } (\text{inorm } x))$ ) ( $\lambda a r. \text{pnorm-dual } (\text{CoD } \Sigma a r)$ ) *final-dual id n*  
**and** *reachable* = *rexp-DA.reachable* ( $\sigma \Sigma$ ) ( $\lambda x. \text{pnorm-dual } (\text{rexp-dual-of } (\text{inorm } x))$ ) ( $\lambda a r. \text{pnorm-dual } (\text{CoD } \Sigma a r)$ ) *id n*  
**and** *automaton* = *rexp-DA.automaton* ( $\sigma \Sigma$ ) ( $\lambda x. \text{pnorm-dual } (\text{rexp-dual-of } (\text{inorm } x))$ ) ( $\lambda a r. \text{pnorm-dual } (\text{CoD } \Sigma a r)$ ) *id n*  
**by** *unfold-locales* (*auto simp only: comp-apply id-apply*  
*wf-inorm lang-inorm*  
*wf-dual-pnorm-dual lang-dual-pnorm-dual*  
*wf-dual-rexp-dual-of lang-dual-rexp-dual-of*  
*wf-dual-lderv-dual lang-dual-lderv-dual*  
*lang-dual-final-dual dest!: lang-dual-subset-lists*)

**definition** *check-equiv where*

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

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

**by** (*rule* *box-equals*[*OF iffD1*[*OF lang-Plus-Zero*, *OF dual.D.check-equiv-sound*]  
*sym*[*OF trans*[*OF lang\_{M2L}-rexp-of''-norm*]] *sym*[*OF trans*[*OF lang\_{M2L}-rexp-of''-norm*]]])  
(*auto simp: dual.check-equiv-def intro!: \Phi.wf-rexp-of''*)

**setup** (*Sign.map-naming Name-Space.parent-path*)

## 13 WS1S

### 13.1 Encodings

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

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

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

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

$\llbracket \exists n. P\ n; \exists n. Q\ n \rrbracket \Longrightarrow \text{min } (\text{Least } P)\ (\text{Least } Q) = (\text{LEAST } n. P\ n \vee Q\ n)$

**proof** (*intro sym[OF Least-equality]*)

**fix**  $y$  **assume**  $P\ y \vee Q\ y$

**thus**  $\text{min } (\text{Least } P)\ (\text{Least } Q) \leq y$

**proof** (*elim disjE*)

**assume**  $P\ y$

**hence**  $\text{Least } P \leq y$  **by** (*auto intro: LeastI2-wellorder*)

**thus**  $\text{min } (\text{Least } P)\ (\text{Least } Q) \leq y$  **unfolding** *min-def* **by** *auto*

**next**

**assume**  $Q\ y$

**hence**  $\text{Least } Q \leq y$  **by** (*auto intro: LeastI2-wellorder*)

**thus**  $\text{min } (\text{Least } P)\ (\text{Least } Q) \leq y$  **unfolding** *min-def* **by** *auto*

**qed**

**qed** (*metis LeastI-ex min-def*)

**lemma** *sconst-collapse*:  $y \#\# \text{sconst } y = \text{sconst } y$

**by** (*subst (2) siterate.ctr*) *auto*

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

**by** (*induct rule: list-induct2*) *auto*

**context** *formula*

**begin**

**definition** *any*  $\equiv \text{hd } \Sigma$

**lemma** *any- $\Sigma$* [*simp*]:  $\text{any} \in \text{set } \Sigma$

**unfolding** *any-def* **by** (*auto simp: nonempty intro: someI[of - hd  $\Sigma$ ]*)

**lemma** *any- $\sigma$* [*simp*]:  $\text{length } bs = n \Longrightarrow (\text{any}, bs) \in \text{set } (\sigma\ \Sigma\ n)$

**by** (*auto simp:  $\sigma$ -def set-n-lists*)

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

*stream-enc*  $(w, I) = \text{smap2 } (\text{enc-atom } I) \text{ nats } (w \text{ @- } \text{sconst } \text{any})$

**lemma** *tl-stream-enc*[*simp*]:  $\text{smap } \pi (\text{stream-enc } (w, x \#\# I)) = \text{stream-enc } (w, I)$

**by** (*auto simp: comp-def  $\pi$ -def*)

**lemma** *enc-atom-max*:  $\llbracket \forall x \in \text{set } I. \text{ case } x \text{ of } \text{Inl } p \Rightarrow p \leq n \mid \text{Inr } P \Rightarrow \forall p \in P. p \leq n; n \leq n' \rrbracket \Longrightarrow$   
*enc-atom I (Suc n') a = (a, replicate (length I) False)*  
**by** (*induct I*) (*auto split: sum.splits*)

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

**proof** –

**from** *assms* **have**  $\exists n > \text{length } w. \forall x \in \text{set } I. \text{ case } x \text{ of } \text{Inl } p \Rightarrow p \leq n \mid \text{Inr } P \Rightarrow \forall p \in P. p \leq n$

**proof** (*induct I*)

**case** (*Cons x I*)

**then obtain** *n* **where** *IH*: *length w < n*

$\forall x \in \text{set } I. \text{ case } x \text{ of } \text{Inl } p \Rightarrow p \leq n \mid \text{Inr } P \Rightarrow \forall p \in P. p \leq n$  **by** *auto*

**thus** *?case*

**proof** (*cases x*)

**case** (*Inl p*)

**with** *IH* **show** *?thesis*

**by** (*intro exI[of - max p n]*) (*fastforce split: sum.splits*)

**next**

**case** (*Inr P*)

**with** *IH Cons(2)* **show** *?thesis*

**by** (*intro exI[of - max (Max P) n]*) (*fastforce dest: Max-ge split: sum.splits*)

**qed**

**qed** *auto*

**then obtain** *n* **where** *length w < n*  $\forall x \in \text{set } I. \text{ case } x \text{ of } \text{Inl } p \Rightarrow p \leq n \mid \text{Inr } P \Rightarrow \forall p \in P. p \leq n$

**by** (*elim exE conjE*)

**hence** *sdrop (Suc n) (stream-enc (w, I)) = sconst (any, replicate (length I) False)*

(*is ?s1 n = ?s2*)

**by** (*intro stream.coinduct[of  $\lambda s1 s2. \exists n' \geq n. s1 = ?s1 n' \wedge s2 = ?s2$ ]*)

(*auto simp: enc-atom-max dest: le-SucI*)

**thus** *?thesis* **by** *blast*

**qed**

**lemma** *length-snth-enc[simp]*: *length (snd (stream-enc (w, I) !! n)) = length I*

**by** *auto*

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

**by** (*cases s*) *auto*

**lemma** *drop-sconstE*:  $\llbracket \text{drop } n \text{ w } @- \text{sconst } y = \text{sconst } y; p < \text{length } w; \neg p < n \rrbracket \Longrightarrow w ! p = y$

**unfolding** *not-less sconst-alt* **proof** (*induct p arbitrary: w n*)

**case** (*Suc p*)

**with** *Suc(1)[of 0 tl w]* **show** *?case*

**by** (*cases w n rule: list.exhaust[case-product nat.exhaust]*) *auto*

**qed** (*auto simp add: neq-Nil-conv*)

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

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

**unfolding** *cut-same-def length-stake*

**by** (*rule LeastI2-ex[OF exI[of - length w]]*)

(*auto simp: sdrop-shift shift-snth split: if-split-asm elim!: drop-sconstE*)

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

$\llbracket (\forall x \in \text{set } I. \text{case } x \text{ of } \text{Inr } P \Rightarrow \text{finite } P \mid - \Rightarrow \text{True}); r < \text{length } I; I ! r = \text{Inl } p \rrbracket \implies$

$p < \text{length } (\text{cut-same } (\text{any}, \text{replicate } (\text{length } I) \text{ False}) (\text{stream-enc } (w, I)))$

**unfolding** *cut-same-def length-stake*

**by** (*erule LeastI2-ex[OF ex-Loop-stream-enc ccontr]*,

*auto simp: smap2-alt list-eq-iff-nth-eq add commute dest!: add-diff-inverse split: sum.splits,*

*metis*)

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

$\llbracket (\forall x \in \text{set } I. \text{case } x \text{ of } \text{Inr } P \Rightarrow \text{finite } P \mid - \Rightarrow \text{True}); r < \text{length } I; I ! r = \text{Inr } p \rrbracket \implies$

$\forall p \in P. p < \text{length } (\text{cut-same } (\text{any}, \text{replicate } (\text{length } I) \text{ False}) (\text{stream-enc } (w, I)))$

**unfolding** *cut-same-def length-stake*

**by** (*rule ballI, erule LeastI2-ex[OF ex-Loop-stream-enc ccontr]*,

*auto simp: smap2-alt list-eq-iff-nth-eq add commute dest!: add-diff-inverse split: sum.splits,*

*metis*)

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

*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]*: *cut-same* *x* (*sconst* *x*) = []

**unfolding** *cut-same-def* **by** (*auto intro: Least-equality*)

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

**assumes**  $x \neq y$

**shows** *cut-same* *x* (*xs* @- *y* ## *sconst* *x*) = *xs* @ [*y*] (**is** *cut-same* *x* ?*s* = -)

**proof** -

**have** (*LEAST* *n*. *sdrop* *n* ?*s* = *sconst* *x*) = *Suc* (*length* *xs*)

**proof** (*rule Least-equality*)

**show** *sdrop* (*Suc* (*length* *xs*)) ?*s* = *sconst* *x* **by** (*auto simp: sdrop-shift*)

**next**

**fix** *m* **assume** \*: *sdrop* *m* ?*s* = *sconst* *x*

{ **assume**  $m < \text{Suc } (\text{length } xs)$

**hence**  $m \leq \text{length } xs$  **by** *simp*

```

then obtain  $ys$  where  $sdrop\ m\ ?s = ys\ @-\ y\ \#\#\ sconst\ x$ 
  by atomize-elim (induct m arbitrary: xs, auto)
with * obtain  $ys\ @-\ y\ \#\#\ sconst\ x = sconst\ x$  by simp
from arg-cong[OF this, of sdrop (length ys)] have  $y\ \#\#\ sconst\ x = sconst\ x$ 
  by (auto simp: sdrop-shift)
with assms have False by (metis siterate.code stream.inject)
}
thus  $Suc\ (length\ xs) \leq m$  by (blast intro: leI)
qed
thus ?thesis unfolding cut-same-def stake-shift by simp
qed

```

```

lemma cut-same-shift-sconst:  $\exists n. w = cut-same\ x\ (w\ @-\ sconst\ x)\ @\ replicate\ n\ x$ 
proof (induct w rule: rev-induct)
  case (snoc a w)
  then obtain  $n$  where  $w = cut-same\ x\ (w\ @-\ sconst\ x)\ @\ replicate\ n\ x$  by blast
  thus ?case
    by (cases a = x)
    (auto simp: id-def[symmetric] siterate.code[of id, simplified id-apply, symmetric]
      replicate-append-same[symmetric] intro!: exI[of - Suc n])
qed (simp add: id-def[symmetric])

```

```

lemma set-cut-same:  $set\ (cut-same\ x\ (w\ @-\ sconst\ x)) \subseteq set\ w$ 
proof (induct w rule: rev-induct)
  case (snoc a w)
  thus ?case by (cases a = x)
  (auto simp: id-def[symmetric] siterate.code[of id, simplified id-apply, symmetric])
qed (simp add: id-def[symmetric])

```

```

lemma stream-enc-cut-same:
  assumes  $(\forall x \in set\ I. case\ x\ of\ Inr\ P \Rightarrow finite\ P \mid - \Rightarrow True)$ 
  shows  $stream-enc\ (w, I) = cut-same\ (any, replicate\ (length\ I)\ False)\ (stream-enc\ (w, I))\ @-\ sconst\ (any, replicate\ (length\ I)\ False)$ 
  unfolding cut-same-def
  by (rule trans[OF sym[OF stake-sdrop] arg-cong2[of - - - (@-), OF refl]])
  (rule LeastI-ex[OF ex-Loop-stream-enc[OF assms]])

```

```

lemma stream-enc-enc:
  assumes  $(\forall x \in set\ I. case\ x\ of\ Inr\ P \Rightarrow finite\ P \mid - \Rightarrow True)$  and  $v: v \in enc\ (w, I)$ 
  shows  $stream-enc\ (w, I) = v\ @-\ sconst\ (any, replicate\ (length\ I)\ False)$ 
  (is  $?s = ?v\ @-\ sconst\ ?F$ )
proof -
  from assms(1) obtain  $n$  where  $sdrop\ n\ (stream-enc\ (w, I)) = sconst\ ?F$  by
  (metis ex-Loop-stream-enc)

```

**moreover from  $v$  obtain  $m$  where  $?v = \text{cut-same } ?F ?s @ \text{replicate } m ?F$  by *auto***  
**ultimately show  $?s = v @ - \text{sconst } ?F$**   
**by (*auto simp del: stream-enc.simps intro: stream-enc-cut-same*[*OF assms*(1)])**  
**qed**

**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**  $\text{stream-enc } (w, I) = (\text{SOME } v. v \in \text{enc } (w, I)) @ - \text{sconst } (\text{any, replicate } (\text{length } I) \text{ False})$   
**by** (*rule stream-enc-enc*[*OF assms*], *rule someI-ex*) *auto*

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

**by *auto***

**lemma *sdrop-sconst*:**  $\text{sdrop } n s = \text{sconst } x \Longrightarrow n \leq m \Longrightarrow s !! m = x$

**by** (*metis le-iff-add sdrop-snth snth-siterate*[*of id, simplified id-funpow id-apply*])

**lemma *fin-cut-same-tl*:**

**assumes**  $\exists n. \text{sdrop } n s = \text{sconst } x$

**shows**  $\text{fin-cut-same } (\pi x) (\text{map } \pi (\text{cut-same } x s)) = \text{cut-same } (\pi x) (\text{smap } \pi s)$

**proof** –

**define *min* where  $\text{min} = (\text{LEAST } n. \text{sdrop } n s = \text{sconst } x)$**

**from *assms* have *min*:**  $\text{sdrop } \text{min } s = \text{sconst } x \wedge m. \text{sdrop } m s = \text{sconst } x \Longrightarrow \text{min} \leq m$

**unfolding *min-def* by** (*auto intro: LeastI Least-le*)

**have *Ex*:**  $\exists n. \text{drop } n (\text{map } \pi (\text{stake } \text{min } s)) = \text{replicate } (\text{length } (\text{map } \pi (\text{stake } \text{min } s)) - n) (\pi x)$

**by** (*auto intro: exI*[*of - length (map pi (stake min s))*])

**have**  $\text{fin-cut-same } (\pi x) (\text{map } \pi (\text{cut-same } x s)) =$

$\text{map } \pi (\text{stake } (\text{LEAST } n.$

$\text{map } \pi (\text{stake } (\text{min} - n) (\text{sdrop } n s)) = \text{replicate } (\text{min} - n) (\pi x) \vee \text{sdrop } n s = \text{sconst } x) s)$

**unfolding *fin-cut-same-def cut-same-def take-map take-stake min-Least***[*OF Ex assms, folded min-def*]

$\text{min-def}$ [*symmetric*] **by** (*auto simp: drop-map drop-stake*)

**also have**  $(\lambda n. \text{map } \pi (\text{stake } (\text{min} - n) (\text{sdrop } n s)) = \text{replicate } (\text{min} - n) (\pi x) \vee \text{sdrop } n s = \text{sconst } x) =$

$(\lambda n. \text{smap } \pi (\text{sdrop } n s) = \text{sconst } (\pi x))$

**proof** (*rule ext, unfold smap-alt snth-siterate*[*of id, simplified id-funpow id-apply*], *safe*)

**fix**  $n m$

**assume**  $\text{map } \pi (\text{stake } (\text{min} - n) (\text{sdrop } n s)) = \text{replicate } (\text{min} - n) (\pi x)$

**hence**  $\forall y \in \text{set } (\text{stake } (\text{min} - n) (\text{sdrop } n s)). \pi y = \pi x$

**by** (*intro iffD1*[*OF map-eq-conv*]) (*metis length-stake map-replicate-const*)

**hence**  $\forall i < \text{min} - n. \pi (\text{sdrop } n s !! i) = \pi x$

**unfolding *all-set-conv-all-nth* by** (*auto simp: sdrop-snth*)

**thus**  $\pi (\text{sdrop } n s !! m) = \pi x$

```

proof (cases m < min - n)
  case False
  hence min ≤ n + m by linarith
  hence sdrop n s !! m = x unfolding sdrop-snth by (rule sdrop-sconst[OF
min(I)])
  thus ?thesis by simp
  qed auto
next
  fix n
  assume ∀ m. π (sdrop n s !! m) = π x
  thus map π (stake (min - n) (sdrop n s)) = replicate (min - n) (π x)
  unfolding stake-smap[symmetric] smap-alt[symmetric, of π sdrop n s sconst
(π x), simplified]
  by (auto simp: map-replicate-const)
  qed auto
  finally show ?thesis unfolding cut-same-def sdrop-smap stake-smap .
qed

```

```

lemma tl-enc[simp]:
  assumes ∀ x ∈ set (x # I). case x of Inr P ⇒ finite P | - ⇒ True
  shows SAMEQUOT (any, replicate (length I) False) (map π ‘ enc (w, x # I))
= enc (w, I)
  unfolding SAMEQUOT-def
  by (fastforce simp: assms π-def
  fin-cut-same-tl[OF ex-Loop-stream-enc[OF assms], unfolded π-def, simplified,
symmetric])

```

```

lemma encD:
  [[v ∈ enc (w, I); (∀ x ∈ set I. case x of Inr P ⇒ finite P | - ⇒ True)] ⇒
v = map (case-prod (enc-atom I)) (zip [0 ..< length v] (stake (length v) (w @-
sconst any)))
  by (erule box-equals[OF sym[OF arg-cong[of - - stake (length v) , OF stream-enc-enc]]])
  (auto simp: stake-shift sdrop-shift stake-add[symmetric] map-replicate-const)

```

```

lemma enc-Inl: [[x ∈ enc (w, I); (∀ x ∈ set I. case x of Inr P ⇒ finite P | - ⇒
True);
m < length I; I ! m = Inl p] ⇒ p < length x ∧ snd (x ! p) ! m
by (auto dest!: less-length-cut-same-Inl[of - - w] simp: nth-append cut-same-def)

```

```

lemma enc-Inr: assumes x ∈ enc (w, I) ∀ x ∈ set I. case x of Inr P ⇒ finite P
| - ⇒ True

```

```

  M < length I I ! M = Inr P
  shows p ∈ P ↔ p < length x ∧ snd (x ! p) ! M

```

**proof**

```

  assume p ∈ P with assms show p < length x ∧ snd (x ! p) ! M

```

```

  by (auto dest!: less-length-cut-same-Inr[of - - w] simp: nth-append cut-same-def)

```

**next**

```

  assume p < length x ∧ snd (x ! p) ! M

```

```

  thus p ∈ P using assms by (subst (asm) (2) encD[OF assms(1,2)]) auto

```



qed

**lemma** *enc-length*:

**assumes**  $enc(w, I) = enc(w', I')$

**shows**  $length I = length I'$

**proof** –

**let**  $?cL = \lambda w I. cut\_same (any, replicate (length I) False) (stream\_enc (w, I))$

**let**  $?w = \lambda w I m. ?cL w I @ replicate (m - length (?cL w I)) (any, replicate (length I) False)$

**let**  $?max = max (length (?cL w I)) (length (?cL w' I')) + 1$

**from** *assms* **have**  $?w w I ?max \in enc(w, I) \quad ?w w' I' ?max \in enc(w', I')$  **by** *auto*

**hence**  $?w w I ?max = ?w w' I' ?max$  **using** *enc-unique-length assms* **by** (*simp del: enc.simps*)

**moreover** **have**  $last (?w w I ?max) = (any, replicate (length I) False)$

$last (?w w' I' ?max) = (any, replicate (length I') False)$  **by** *auto*

**ultimately show**  $length I = length I'$  **by** *auto*

qed

**lemma** *enc-stream-enc*:

$[(\forall x \in set I. case x of Inr P \Rightarrow finite P \mid - \Rightarrow True);$

$(\forall x \in set I'. case x of Inr P \Rightarrow finite P \mid - \Rightarrow True);$

$enc(w, I) = enc(w', I')]] \Longrightarrow stream\_enc(w, I) = stream\_enc(w', I')$

**by** (*rule box-equals[OF - sym[OF stream-enc-enc-some] sym[OF stream-enc-enc-some]]*)  
(*auto dest: enc-length simp del: enc.simps*)

**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*  $(w, I) \varphi =$

$(wf\_interp w I \wedge$

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

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

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

$(w, I) \models FQ a m = ((case I ! m of Inl p \Rightarrow if p < length w then w ! p else any) = a)$

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

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

$(w, I) \models FNot \varphi = (\neg (w, I) \models \varphi)$

$(w, I) \models FOr \varphi_1 \varphi_2 = ((w, I) \models \varphi_1 \vee (w, I) \models \varphi_2)$

$(w, I) \models FAnd \varphi_1 \varphi_2 = ((w, I) \models \varphi_1 \wedge (w, I) \models \varphi_2)$

$(w, I) \models FExists \varphi = (\exists p. (w, Inl p \# I) \models \varphi)$

$(w, I) \models FEXISTS \varphi = (\exists P. finite P \wedge (w, Inr P \# I) \models \varphi)$

**definition** *lang<sub>W<sub>S</sub>I<sub>S</sub></sub>*  $:: nat \Rightarrow 'a formula \Rightarrow ('a \times bool list) list set$  **where**

*lang<sub>W<sub>S</sub>I<sub>S</sub></sub>*  $n \varphi = \bigcup \{enc(w, I) \mid w I. length I = n \wedge wf\_interp\_for\_formula (w,$

$I) \varphi \wedge (w, I) \models \varphi\}$

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

**lemma** *enc-set-σ*:  $\llbracket x \in \text{enc } (w, I); (\forall x \in \text{set } I. \text{case } x \text{ of } \text{Inr } P \Rightarrow \text{finite } P \mid - \Rightarrow \text{True});$   
 $\text{length } I = n; a \in \text{set } x; \text{set } w \subseteq \text{set } \Sigma \rrbracket \Longrightarrow a \in \text{set } (\sigma \Sigma n)$   
**by** (*force dest: encD-ex intro: enc-atom-σ simp: in-set-zip shift-snth simp del: enc.simps*)

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

**lemma** *positions-in-row*:  $\text{positions-in-row } s \ i = \{p. \text{snd } (s !! p) ! i\}$   
**unfolding** *positions-in-row-def Option.these-def smap2-szip stream.set-map sset-range*  
**by** (*auto split: if-split-asm intro!: image-eqI[of - the] split: prod.splits*)

**lemma** *positions-in-row-unique*:  $\exists ! p. \text{snd } (s !! p) ! i \Longrightarrow$   
 $\text{the-elem } (\text{positions-in-row } s \ i) = (\text{THE } p. \text{snd } (s !! p) ! i)$   
**by** (*rule the1I2*) (*auto simp: the-elem-def positions-in-row*)

**lemma** *positions-in-row-nth*:  $\exists ! p. \text{snd } (s !! p) ! i \Longrightarrow$   
 $\text{snd } (s !! \text{the-elem } (\text{positions-in-row } s \ i)) ! i$   
**unfolding** *positions-in-row-unique* **by** (*rule the1I2*) *auto*

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

**lemma** *dec-word-stream-enc*:  $\text{dec-word } (\text{stream-enc } (w, I)) = \text{cut-same any } (w @- \text{sconst any})$   
**unfolding** *dec-word-def* **by** (*auto intro!: arg-cong[of - - cut-same any] simp: smap2-alt*)

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

**lemma** *stream-dec-Inl*:  $\llbracket i \in FO; i < n \rrbracket \Longrightarrow \exists p. \text{stream-dec } n \ FO \ s ! i = \text{Inl } p$   
**unfolding** *stream-dec-def* **using** *nth-map[of n [0..<n]]* **by** *auto*

**lemma** *stream-dec-not-Inr*:  $\llbracket \text{stream-dec } n \ FO \ s ! i = \text{Inr } P; i \in FO; i < n \rrbracket \Longrightarrow$   
 $\text{False}$   
**unfolding** *stream-dec-def* **using** *nth-map[of n [0..<n]]* **by** *auto*

**lemma** *stream-dec-Inr*:  $\llbracket i \notin FO; i < n \rrbracket \implies \exists P. \text{stream-dec } n \text{ } FO \text{ } s ! i = \text{Inr } P$   
**unfolding** *stream-dec-def* **using** *nth-map*[of *n* [0..*n*]] **by** *auto*

**lemma** *stream-dec-not-Inl*:  $\llbracket \text{stream-dec } n \text{ } FO \text{ } s ! i = \text{Inl } p; i \notin FO; i < n \rrbracket \implies \text{False}$   
**unfolding** *stream-dec-def* **using** *nth-map*[of *n* [0..*n*]] **by** *auto*

**lemma** *Inr-dec-finite*:  $\llbracket \forall i < n. \text{finite } \{p. \text{snd } (s !! p) ! i\}; \text{Inr } P \in \text{set } (\text{stream-dec } n \text{ } FO \text{ } s) \rrbracket \implies \text{finite } P$   
**unfolding** *stream-dec-def* **by** (*auto simp: positions-in-row*)

**lemma** *enc-atom-dec*:  
 $\llbracket \forall p. \text{length } (\text{snd } (s !! p)) = n; \forall i \in FO. i < n \longrightarrow (\exists !p. \text{snd } (s !! p) ! i); a = \text{fst } (s !! p) \rrbracket \implies \text{enc-atom } (\text{stream-dec } n \text{ } FO \text{ } s) \text{ } p \text{ } a = s !! p$   
**unfolding** *stream-dec-def*  
**by** (*rule sym, subst surjective-pairing*[of *s !! p*])  
(*auto intro!: nth-equalityI simp: positions-in-row simp del: prod.collapse split: if-split-asm,*  
*(metis positions-in-row positions-in-row-nth)+*)

**lemma** *length-stream-dec*[*simp*]:  $\text{length } (\text{stream-dec } n \text{ } FO \text{ } x) = n$   
**unfolding** *stream-dec-def* **by** *auto*

**lemma** *stream-enc-dec*:  
 $\llbracket \exists n. \text{sdrop } n \text{ } (\text{smap } \text{fst } s) = \text{sconst } \text{any}; \text{stream-all } (\lambda x. \text{length } (\text{snd } x) = n) \text{ } s; \forall i \in FO. (\exists !p. \text{snd } (s !! p) ! i) \rrbracket \implies \text{stream-enc } (\text{dec-word } s, \text{stream-dec } n \text{ } FO \text{ } s) = s$   
**unfolding** *dec-word-def*  
**by** (*drule LeastI-ex*)  
(*auto intro!: enc-atom-dec simp: smap2-alt cut-same-def simp del: stake-smap sdrop-smap intro!: trans[OF arg-cong2[of - - - - (!)] snth-smap] trans[OF arg-cong2[of - - - - (@-)] stake-sdrop]*)

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

**lemma** *stream-dec-enc-Inl*:  
 $\llbracket \text{stream-dec } n \text{ } FO \text{ } (\text{stream-enc } (w, I)) ! i = \text{Inl } p'; I ! i = \text{Inl } p; i \in FO; i < n; \text{length } I = n \rrbracket \implies p = p'$   
**unfolding** *stream-dec-def*  
**by** (*auto intro!: trans[OF - sym[OF positions-in-row-unique[OF stream-enc-unique]]] simp del: stream-enc.simps*) *simp*

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

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

$P = P'$

**unfolding** *stream-dec-def positions-in-row* **by** *auto*

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

**unfolding** *image-def* **by** (*auto simp: gr0-conv-Suc*)

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

**by** (*induct w*) (*auto simp: id-def[symmetric] intro: finite-subset[OF Collect-snth]*)

**lemma** *lang-ENC*:

**assumes**  $\text{FO} \subseteq \{0 ..< n\}$   $\text{SO} \subseteq \{0 ..< n\} - \text{FO}$

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

$\wedge$

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

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

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

**proof** (*intro equalityI subsetI*)

**fix**  $x$  **assume**  $L: x \in ?L$

**from** *assms(1)* **have** *fin*: *finite FO* **by** (*auto simp: finite-subset*)

**have**  $*$ :  $\text{set } x \subseteq \text{set } (\sigma \Sigma n)$  **using** *subsetD[OF assms(1)]*

*bspec[OF wf-lang-wf-word[OF wf-rexp-ENC[OF fin]] L]*

**by** (*cases n*) (*auto simp: wf-word*)

**let**  $?s = x @- \text{sconst } (\text{any}, \text{replicate } n \text{ False})$

**from** *assms* **have** *list-all*  $(\lambda bs. \text{length } (\text{snd } bs) = n)$   $x$

**using**  $*$  **by** (*auto simp: list-all-iff  $\sigma$ -def set-n-lists*)

**hence** *stream-all*  $(\lambda x. \text{length } (\text{snd } x) = n)$   $(x @- \text{sconst } (\text{any}, \text{replicate } n \text{ False}))$

**by** (*auto simp only: stream-all-shift sset-sconst length-replicate snd-conv*)

**moreover**

**{ fix**  $m$  **assume**  $m \in \text{FO}$

**with** *assms* **have**  $m < n$  **by** (*auto simp: max-idx-vars*)

**with**  $L \langle m \in \text{FO} \rangle$  *assms* **obtain**  $u \ z \ v$  **where**  $uzv: x = u @ z @ v$

$u \in \text{star } (\text{lang } n (\text{Atom } (\text{Arbitrary-Except } m \text{ False})))$

$z \in \text{lang } n (\text{Atom } (\text{Arbitrary-Except } m \text{ True}))$

$v \in \text{star } (\text{lang } n (\text{Atom } (\text{Arbitrary-Except } m \text{ False})))$  **unfolding** *ENC-def*

**by** (*cases n*)

(*auto simp: not-less max-idx-vars valid-ENC-def fin intro!: wf-rexp-valid-ENC*)

*finite-FOV*

*dest!*: *iffD1[OF lang-flatten-INTERSECT, rotated -1], fast*)

**with**  $\langle m < n \rangle$  **have**  $\exists ! p. \text{snd } (x ! p) ! m \wedge p < \text{length } x$

**proof** (*intro ex1I[of - length u]*)

**fix**  $p$  **assume**  $m < n$   $\text{snd } (x ! p) ! m \wedge p < \text{length } x$

**with** *star-Arbitrary-ExceptD[OF uzv(2)] Arbitrary-ExceptD[OF uzv(3)] star-Arbitrary-ExceptD[OF uzv(4)]*

**show**  $p = \text{length } u$  **by** (*cases rule: linorder-cases*) (*auto simp: nth-append uzv(1)*)

**qed** (*auto dest!: Arbitrary-ExceptD*)

```

then obtain  $p$  where  $p: p < \text{length } x \text{ snd } (x ! p) ! m$ 
 $\wedge q. \text{snd } (x ! q) ! m \wedge q < \text{length } x \longrightarrow q = p$  by auto
from this(1,2) have  $\exists ! p. \text{snd } (?s !! p) ! m$ 
proof (intro ex1I[of - p])
  fix  $q$  from  $p(1,2) p(3)[\text{of } q] \langle m < n \rangle$  show  $\text{snd } (?s !! q) ! m \implies q = p$ 
  by (cases  $q < \text{length } x$ ) auto
qed auto
}
moreover have  $\text{sdrop } (\text{length } x) (\text{smap } \text{fst } (x @- \text{sconst } (\text{any}, \text{replicate } n \text{ False}))) = \text{sconst } \text{any}$ 
unfolding sdrop-smap by (simp add: sdrop-shift)
ultimately have  $\text{enc-dec}: \text{stream-enc } (\text{dec-word } ?s, \text{stream-dec } n \text{ FO } ?s) =$ 
 $x @- \text{sconst } (\text{any}, \text{replicate } n \text{ False})$  by (intro stream-enc-dec) auto
define  $I$  where  $I = \text{stream-dec } n \text{ FO } ?s$ 
with assms have wf-interp (dec-word  $?s$ )  $I \wedge$ 
 $(\forall i \in \text{FO}. \text{case } I ! i \text{ of } \text{Inl } - \Rightarrow \text{True} \mid \text{Inr } - \Rightarrow \text{False}) \wedge$ 
 $(\forall i \in \text{SO}. \text{case } I ! i \text{ of } \text{Inl } - \Rightarrow \text{False} \mid \text{Inr } - \Rightarrow \text{True})$  unfolding I-def dec-word-def
by (auto dest: stream-dec-not-Inr stream-dec-not-Inl simp:  $\sigma$ -def max-idx-vars
dest!: subsetD[OF set-cut-same[of any map fst x]] subsetD[OF *] split:
sum.splits)
(auto simp: stream-dec-def positions-in-row finite-True-in-row)
moreover have  $\text{length } I = n$  unfolding I-def by simp
moreover have  $x \in \text{enc } (\text{dec-word } ?s, I)$  unfolding I-def
by (simp add: enc-dec cut-same-shift-sconst del: stream-enc.simps)
ultimately show  $x \in ?R$  by blast
next
fix  $x$  assume  $x \in ?R$ 
then obtain  $w I$  where  $I: x \in \text{enc } (w, I) \text{ wf-interp } w I \wedge$ 
 $(\forall i \in \text{FO}. \text{case } I ! i \text{ of } \text{Inl } - \Rightarrow \text{True} \mid \text{Inr } - \Rightarrow \text{False}) \wedge$ 
 $(\forall i \in \text{SO}. \text{case } I ! i \text{ of } \text{Inl } - \Rightarrow \text{False} \mid \text{Inr } - \Rightarrow \text{True})$   $\text{length } I = n$  by blast
{ fix  $i$  from  $I(2)$  have  $(w @- \text{sconst } \text{any}) !! i \in \text{set } \Sigma$  by (cases  $i < \text{length } w$ )
auto } note  $* = \text{this}$ 
from  $I$  have  $x @- \text{sconst } (\text{any}, \text{replicate } (\text{length } I) \text{ False}) = \text{stream-enc } (w, I)$ 
(is  $x @- ?F = ?s$ )
by (intro stream-enc-enc[symmetric]) auto
with  $* \langle \text{length } I = n \rangle$  have  $\forall x \in \text{set } x. \text{length } (\text{snd } x) = n \wedge \text{fst } x \in \text{set } \Sigma$ 
by (auto dest!: shift-snth-less[of - - ?F, symmetric] simp: in-set-conv-nth)
thus  $x \in ?L$ 
proof (cases  $\text{FO} = \{\}$ )
  case False
  hence nonempty: valid-ENC  $n \text{ ' FO } \neq \{\}$  by simp
have finite: finite (valid-ENC  $n \text{ ' FO}$ ) by (rule finite-imageI[OF finite-subset[OF
assms(1)]] simp)
from False assms(1) have  $0 < n$  by (cases  $n$ ) (auto split: dest!: max-idx-vars)
with wf-rexp-valid-ENC assms have wf-rexp:  $\forall x \in \text{valid-ENC } n \text{ ' FO}. \text{wf } n \ x$ 
by (auto simp: max-idx-vars)
{ fix  $r$  assume  $r \in \text{FO}$ 
  with  $I(2)$  obtain  $p$  where  $p: I ! r = \text{Inl } p$  by (cases  $I ! r$ ) auto
from  $\langle r \in \text{FO} \rangle$  assms  $I(2,3)$  have  $r: r < \text{length } I$  by (auto dest!: max-idx-vars)

```

```

from  $p I(1,2) r$  have  $p < \text{length } x$ 
  using less-length-cut-same-Inl[of  $I r p w$ ] by auto
with  $p I r *$ 
  have  $[x ! p] \in \text{lang } n$  (Atom (Arbitrary-Except r True))
  by (subst encD[of  $x w I$ ]) (auto simp del: lang.simps intro!: enc-atom-lang-Arbitrary-Except-True)
moreover
from  $p I r *$  have  $\text{take } p x \in \text{star } (\text{lang } n$  (Atom (Arbitrary-Except r False)))
  by (subst encD[of  $x$ ]) (auto simp del: lang.simps simp: in-set-conv-nth
intro!: Ball-starI enc-atom-lang-Arbitrary-Except-False)
moreover
from  $p I r *$  have  $\text{drop } (\text{Suc } p) x \in \text{star } (\text{lang } n$  (Atom (Arbitrary-Except r
False)))
  by (subst encD[of  $x$ ]) (auto simp: in-set-conv-nth simp del: lang.simps
snth.simps intro!: Ball-starI enc-atom-lang-Arbitrary-Except-False)
  ultimately have  $\text{take } p x @ [x ! p] @ \text{drop } (p + 1) x \in \text{lang } n$  (valid-ENC
n r)
  using  $\langle 0 < n \rangle$  unfolding valid-ENC-def by (auto simp del: append.simps)
  hence  $x \in \text{lang } n$  (valid-ENC n r) using id-take-nth-drop[OF  $\langle p < \text{length } x \rangle$ ]
by auto
  }
with False lang-flatten-INTERSECT[OF finite nonempty wf-rexp] show ?thesis
by (auto simp: ENC-def)
  qed (simp add: ENC-def, auto simp:  $\sigma$ -def set-n-lists image-iff)
qed

```

```

lemma lang-ENC-formula:
  assumes wf-formula n  $\varphi$ 
  shows  $\text{lang } n$  (ENC n (FOV  $\varphi$ )) =  $\bigcup \{ \text{enc } (w, I) \mid w I . \text{length } I = n \wedge$ 
wf-interp-for-formula (w, I)  $\varphi$   $\}$ 
proof –
  from assms max-idx-vars have  $*$ : FOV  $\varphi \subseteq \{0 ..< n\}$  SOV  $\varphi \subseteq \{0 ..< n\}$  –
FOV  $\varphi$  by auto
  show ?thesis unfolding lang-ENC[OF  $*$ ] by simp
qed

```

## 13.2 Welldefinedness of enc wrt. Models

```

lemma wf-interp-for-formula-FExists:
   $\llbracket \text{wf-formula } (\text{length } I) \text{ (FExists } \varphi) \rrbracket \implies$ 
   $\text{wf-interp-for-formula } (w, I) \text{ (FExists } \varphi) \iff (\forall p. \text{wf-interp-for-formula } (w, \text{Inl } p \# I) \varphi)$ 
  by (auto simp: nth-Cons' split: if-split-asm)

```

```

lemma wf-interp-for-formula-any-Inl:  $\text{wf-interp-for-formula } (w, \text{Inl } p \# I) \varphi \implies$ 
 $\forall p. \text{wf-interp-for-formula } (w, \text{Inl } p \# I) \varphi$ 
by (auto simp: nth-Cons' split: if-split-asm)

```

```

lemma wf-interp-for-formula-FEXISTS:
   $\llbracket \text{wf-formula } (\text{length } I) \text{ (FEXISTS } \varphi) \rrbracket \implies$ 

```

*wf-interp-for-formula* ( $w, I$ ) (*FEXISTS*  $\varphi$ )  $\longleftrightarrow$  ( $\forall P. \text{finite } P \longrightarrow \text{wf-interp-for-formula } (w, \text{Inr } P \# I) \varphi$ )

**by** (*auto simp: nth-Cons' split: if-split-asm*)

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

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

**by** (*auto simp: nth-Cons' split: if-split-asm*)

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

*wf-interp-for-formula* ( $w, I$ ) (*FOr*  $\varphi 1 \varphi 2$ ) =

(*wf-interp-for-formula* ( $w, I$ )  $\varphi 1 \wedge \text{wf-interp-for-formula } (w, I) \varphi 2$ )

**by** *auto*

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

*wf-interp-for-formula* ( $w, I$ ) (*FAnd*  $\varphi 1 \varphi 2$ ) =

(*wf-interp-for-formula* ( $w, I$ )  $\varphi 1 \wedge \text{wf-interp-for-formula } (w, I) \varphi 2$ )

**by** *auto*

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

$\llbracket \text{wf-formula } (\text{length } I) \varphi; \text{wf-interp-for-formula } (w, I) \varphi; x \in \text{enc } (w, I) \rrbracket \implies$

*wf-interp-for-formula* (*dec-word* ( $x @- \text{sconst } (\text{any}, \text{replicate } (\text{length } I) \text{False})$ ),

*stream-dec* ( $\text{length } I$ ) (*FOV*  $\varphi$ ) ( $x @- \text{sconst } (\text{any}, \text{replicate } (\text{length } I) \text{False})$ ))

$\varphi$

**using**

*stream-dec-Inl*[*of - FOV*  $\varphi$   $\text{length } I$  *stream-enc* ( $w, I$ ), *OF - bspec*[*OF max-idx-vars*]]

*stream-dec-Inr*[*of - FOV*  $\varphi$   $\text{length } I$  *stream-enc* ( $w, I$ ), *OF - bspec*[*OF max-idx-vars*]]

**by** (*auto split: sum.splits intro: Inr-dec-finite*[*OF finite-True-in-row*] *simp: max-idx-vars dec-word-def*)

*dest!*: *stream-dec-not-Inl stream-dec-not-Inr subsetD*[*OF set-cut-same*] *simp del: stream-enc.simps*)

(*auto simp: cut-same-def in-set-zip smap2-alt shift-snth*)

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

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

**proof** (*induct length I arbitrary: I I' m*)

**case** (*Suc*  $n I I'$ )

**then obtain**  $x xs x' xs'$  **where**  $*$ :  $I = x \# xs \ I' = x' \# xs'$

**by** (*fastforce simp: Suc-length-conv map-eq-Cons-conv*)

**with** *Suc show ?case*

**proof** (*cases m*)

**case 0 thus ?thesis using** *Suc(3) unfolding \**

**by** (*cases x x' rule: sum.exhaust*[*case-product sum.exhaust*]) *auto*

**qed** *auto*

**qed** *simp*

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

```

wf-interp-for-formula (w, I)  $\varphi$ ; wf-interp-for-formula (w', I')  $\varphi$ ]  $\implies$ 
(w, I)  $\models \varphi \iff (w', I') \models \varphi$ 
proof (induction  $\varphi$  arbitrary: w w' I I')
  case (FQ a m) thus ?case using enc-atom-welldef[of I I' m]
    by (simp split: sum.splits add: smap2-alt shift-snth)
      (metis snth-siterate[of id, simplified id-funpow id-apply])
next
  case (FLess m1 m2) thus ?case using enc-atom-welldef[of I I' m1] enc-atom-welldef[of
I I' m2]
    by (auto split: sum.splits simp add: smap2-alt)
next
  case (FIn m M) thus ?case using enc-atom-welldef[of I I' m] enc-atom-welldef[of
I I' M]
    by (auto split: sum.splits simp add: smap2-alt)
next
  case (FOr  $\varphi_1 \varphi_2$ ) show ?case unfolding satisfies.simps(5)
proof (intro disj-cong)
  from FOr(3-6) show (w, I)  $\models \varphi_1 \iff (w', I') \models \varphi_1$ 
    by (intro FOr(1)) auto
next
  from FOr(3-6) show (w, I)  $\models \varphi_2 \iff (w', I') \models \varphi_2$ 
    by (intro FOr(2)) auto
qed
next
  case (FAnd  $\varphi_1 \varphi_2$ ) show ?case unfolding satisfies.simps(6)
proof (intro conj-cong)
  from FAnd(3-6) show (w, I)  $\models \varphi_1 \iff (w', I') \models \varphi_1$ 
    by (intro FAnd(1)) auto
next
  from FAnd(3-6) show (w, I)  $\models \varphi_2 \iff (w', I') \models \varphi_2$ 
    by (intro FAnd(2)) auto
qed
next
  case (FExists  $\varphi$ )
hence length: length I' = length I by (metis length-snth-enc)
show ?case
proof
  assume (w, I)  $\models$  FExists  $\varphi$ 
with FExists.prem(3) obtain p where (w, Inl p # I)  $\models \varphi$  by auto
moreover
with FExists.prem(1,2) have (w', Inl p # I')  $\models \varphi$ 
proof (intro iffD1[OF FExists.IH[of w Inl p # I w' Inl p # I']])
  from FExists.prem(2,3) show wf-interp-for-formula (w, Inl p # I)  $\varphi$ 
    by (blast dest: wf-interp-for-formula-FExists[of I])
next
  from FExists.prem(2,4) show wf-interp-for-formula (w', Inl p # I')  $\varphi$ 
    by (blast dest: wf-interp-for-formula-FExists[of I', unfolded length])
qed (auto simp: smap2-alt split: sum.splits if-split-asm)
ultimately show (w', I')  $\models$  FExists  $\varphi$  by auto

```



```

next
  assume (w', I')  $\models$  FExists  $\varphi$ 
  with FExists.prem(1,2,4) obtain p where (w', Inl p # I')  $\models$   $\varphi$  by auto
  moreover
  with FExists.prem(1,2) have (w, Inl p # I)  $\models$   $\varphi$ 
  proof (intro iffD2[OF FExists.IH[of w Inl p # I w' Inl p # I']])
    from FExists.prem(2,3) show wf-interp-for-formula (w, Inl p # I)  $\varphi$ 
      by (blast dest: wf-interp-for-formula-FExists[of I])
  next
  from FExists.prem(2,4) show wf-interp-for-formula (w', Inl p # I')  $\varphi$ 
    by (blast dest: wf-interp-for-formula-FExists[of I', unfolded length])
  qed (auto simp: smap2-alt split: sum.splits if-split-asm)
  ultimately show (w, I)  $\models$  FExists  $\varphi$  by auto
qed
next
case (FEXISTS  $\varphi$ )
hence length: length I' = length I by (metis length-snth-enc)
show ?case
proof
  assume (w, I)  $\models$  FEXISTS  $\varphi$ 
  with FEXISTS.prem(3) obtain P where finite P (w, Inr P # I)  $\models$   $\varphi$  by
  auto
  moreover
  with FEXISTS.prem(1,2) have (w', Inr P # I')  $\models$   $\varphi$ 
  proof (intro iffD1[OF FEXISTS.IH[of w Inr P # I w' Inr P # I']])
    from FEXISTS.prem(2,3)  $\langle$ finite P $\rangle$  show wf-interp-for-formula (w, Inr P
  # I)  $\varphi$ 
      by (blast dest: wf-interp-for-formula-FEXISTS[of I])
  next
  from FEXISTS.prem(2,4)  $\langle$ finite P $\rangle$  show wf-interp-for-formula (w', Inr P
  # I')  $\varphi$ 
      by (blast dest: wf-interp-for-formula-FEXISTS[of I', unfolded length])
  qed (auto simp: smap2-alt split: sum.splits if-split-asm)
  ultimately show (w', I')  $\models$  FEXISTS  $\varphi$  by auto
next
  assume (w', I')  $\models$  FEXISTS  $\varphi$ 
  with FEXISTS.prem(1,2,4) obtain P where finite P (w', Inr P # I')  $\models$   $\varphi$ 
  by auto
  moreover
  with FEXISTS.prem have (w, Inr P # I)  $\models$   $\varphi$ 
  proof (intro iffD2[OF FEXISTS.IH[of w Inr P # I w' Inr P # I']])
    from FEXISTS.prem(2,3)  $\langle$ finite P $\rangle$  show wf-interp-for-formula (w, Inr P
  # I)  $\varphi$ 
      by (blast dest: wf-interp-for-formula-FEXISTS[of I])
  next
  from FEXISTS.prem(2,4)  $\langle$ finite P $\rangle$  show wf-interp-for-formula (w', Inr P
  # I')  $\varphi$ 
      by (blast dest: wf-interp-for-formula-FEXISTS[of I', unfolded length])
  qed (auto simp: smap2-alt split: sum.splits if-split-asm)

```

ultimately show  $(w, I) \models FEXISTS \varphi$  by auto  
qed  
qed auto

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

assumes wf-formula  $n$  (FOr  $\varphi_1 \varphi_2$ )  
shows lang<sub>WS1S</sub>  $n$  (FOr  $\varphi_1 \varphi_2$ )  $\subseteq$   
(lang<sub>WS1S</sub>  $n \varphi_1 \cup$  lang<sub>WS1S</sub>  $n \varphi_2$ )  $\cap \bigcup \{enc(w, I) \mid w I. length I = n \wedge$   
wf-interp-for-formula  $(w, I)$  (FOr  $\varphi_1 \varphi_2)\}$   
(is  $\subseteq$  (?L1  $\cup$  ?L2)  $\cap$  ?ENC)  
**proof** (intro equalityI subsetI)  
**fix**  $x$  **assume**  $x \in$  lang<sub>WS1S</sub>  $n$  (FOr  $\varphi_1 \varphi_2$ )  
**then obtain**  $w I$  **where**  
 $*$ :  $x \in$  enc  $(w, I)$  wf-interp-for-formula  $(w, I)$  (FOr  $\varphi_1 \varphi_2$ ) length  $I = n$  **and**  
satisfies  $(w, I) \varphi_1 \vee$  satisfies  $(w, I) \varphi_2$  **unfolding** lang<sub>WS1S</sub>-def **by** auto  
**thus**  $x \in$  (?L1  $\cup$  ?L2)  $\cap$  ?ENC  
**proof** (elim disjE)  
**assume** satisfies  $(w, I) \varphi_1$   
**with**  $*$  **have**  $x \in$  ?L1 **using** assms **unfolding** lang<sub>WS1S</sub>-def **by** (fastforce  
simp del: enc.simps)  
**with**  $*$  **show** ?thesis **by** auto  
**next**  
**assume** satisfies  $(w, I) \varphi_2$   
**with**  $*$  **have**  $x \in$  ?L2 **using** assms **unfolding** lang<sub>WS1S</sub>-def **by** (fastforce simp  
del: enc.simps)  
**with**  $*$  **show** ?thesis **by** auto  
**qed**  
**qed**

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

assumes wf-formula  $n$  (FAnd  $\varphi_1 \varphi_2$ )  
shows lang<sub>WS1S</sub>  $n$  (FAnd  $\varphi_1 \varphi_2$ )  $\subseteq$   
lang<sub>WS1S</sub>  $n \varphi_1 \cap$  lang<sub>WS1S</sub>  $n \varphi_2 \cap \bigcup \{enc(w, I) \mid w I. length I = n \wedge$   
wf-interp-for-formula  $(w, I)$  (FAnd  $\varphi_1 \varphi_2)\}$   
**using** assms **unfolding** lang<sub>WS1S</sub>-def **by** (fastforce simp del: enc.simps)

### 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))))

|  $\text{rexp-of } n \text{ (FNot } \varphi) = \text{Inter } (\text{rexp.Not } (\text{rexp-of } n \varphi)) \text{ (ENC } n \text{ (FOV (FNot } \varphi)))$   
 |  $\text{rexp-of } n \text{ (FOr } \varphi_1 \varphi_2) = \text{Inter } (\text{Plus } (\text{rexp-of } n \varphi_1) (\text{rexp-of } n \varphi_2)) \text{ (ENC } n \text{ (FOV (FOr } \varphi_1 \varphi_2)))$   
 |  $\text{rexp-of } n \text{ (FAnd } \varphi_1 \varphi_2) = \text{INTERSECT } [\text{rexp-of } n \varphi_1, \text{rexp-of } n \varphi_2, \text{ENC } n \text{ (FOV (FAnd } \varphi_1 \varphi_2))]$   
 |  $\text{rexp-of } n \text{ (FExists } \varphi) = \text{samequot-exec } (\text{any, replicate } n \text{ False}) \text{ (Pr } (\text{rexp-of } (n + 1) \varphi))$   
 |  $\text{rexp-of } n \text{ (FEXISTS } \varphi) = \text{samequot-exec } (\text{any, replicate } n \text{ False}) \text{ (Pr } (\text{rexp-of } (n + 1) \varphi))$

**fun**  $\text{rexp-of-alt} :: \text{nat} \Rightarrow 'a \text{ formula} \Rightarrow ('a \text{ atom}) \text{ rexp where}$   
 $\text{rexp-of-alt } n \text{ (FQ } a \text{ m)} =$   
 $\text{TIMES } [\text{rexp.Not Zero, Atom (AQ } m \text{ a), rexp.Not Zero}]$   
 |  $\text{rexp-of-alt } n \text{ (FLess } m1 \text{ m2)} = (\text{if } m1 = m2 \text{ then Zero else}$   
 $\text{TIMES } [\text{rexp.Not Zero, Atom (Arbitrary-Except } m1 \text{ True),}$   
 $\text{rexp.Not Zero, Atom (Arbitrary-Except } m2 \text{ True),}$   
 $\text{rexp.Not Zero}]$   
 |  $\text{rexp-of-alt } n \text{ (FIn } m \text{ M)} =$   
 $\text{TIMES } [\text{rexp.Not Zero, Atom (Arbitrary-Except2 } m \text{ M), rexp.Not Zero}]$   
 |  $\text{rexp-of-alt } n \text{ (FNot } \varphi) = \text{rexp.Not } (\text{rexp-of-alt } n \varphi)$   
 |  $\text{rexp-of-alt } n \text{ (FOr } \varphi_1 \varphi_2) = \text{Plus } (\text{rexp-of-alt } n \varphi_1) (\text{rexp-of-alt } n \varphi_2)$   
 |  $\text{rexp-of-alt } n \text{ (FAnd } \varphi_1 \varphi_2) = \text{Inter } (\text{rexp-of-alt } n \varphi_1) (\text{rexp-of-alt } n \varphi_2)$   
 |  $\text{rexp-of-alt } n \text{ (FExists } \varphi) = \text{samequot-exec } (\text{any, replicate } n \text{ False}) \text{ (Pr } (\text{Inter } (\text{rexp-of-alt } (n + 1) \varphi) \text{ (ENC } (\text{Suc } n) \text{ (FOV } \varphi))))$   
 |  $\text{rexp-of-alt } n \text{ (FEXISTS } \varphi) = \text{samequot-exec } (\text{any, replicate } n \text{ False}) \text{ (Pr } (\text{Inter } (\text{rexp-of-alt } (n + 1) \varphi) \text{ (ENC } (\text{Suc } n) \text{ (FOV } \varphi))))$

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

**fun**  $\text{rexp-of-alt}' :: \text{nat} \Rightarrow 'a \text{ formula} \Rightarrow ('a \text{ atom}) \text{ rexp where}$   
 $\text{rexp-of-alt}' n \text{ (FQ } a \text{ m)} = \text{TIMES } [\text{Full, Atom (AQ } m \text{ a), Full}]$   
 |  $\text{rexp-of-alt}' n \text{ (FLess } m1 \text{ m2)} = (\text{if } m1 = m2 \text{ then Zero else}$   
 $\text{TIMES } [\text{Full, Atom (Arbitrary-Except } m1 \text{ True), Full, Atom (Arbitrary-Except } m2 \text{ True), Full}]$   
 |  $\text{rexp-of-alt}' n \text{ (FIn } m \text{ M)} = \text{TIMES } [\text{Full, Atom (Arbitrary-Except2 } m \text{ M), Full}]$   
 |  $\text{rexp-of-alt}' n \text{ (FNot } \varphi) = \text{rexp.Not } (\text{rexp-of-alt}' n \varphi)$   
 |  $\text{rexp-of-alt}' n \text{ (FOr } \varphi_1 \varphi_2) = \text{Plus } (\text{rexp-of-alt}' n \varphi_1) (\text{rexp-of-alt}' n \varphi_2)$   
 |  $\text{rexp-of-alt}' n \text{ (FAnd } \varphi_1 \varphi_2) = \text{Inter } (\text{rexp-of-alt}' n \varphi_1) (\text{rexp-of-alt}' n \varphi_2)$   
 |  $\text{rexp-of-alt}' n \text{ (FExists } \varphi) = \text{samequot-exec } (\text{any, replicate } n \text{ False}) \text{ (Pr } (\text{Inter } (\text{rexp-of-alt}' (n + 1) \varphi) \text{ (ENC } (n + 1) \{0\})))$   
 |  $\text{rexp-of-alt}' n \text{ (FEXISTS } \varphi) = \text{samequot-exec } (\text{any, replicate } n \text{ False}) \text{ (Pr } (\text{rexp-of-alt}' (n + 1) \varphi))$

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

**lemma**  $\text{enc-eqI}$ :

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

**shows**  $enc (w, I) = enc (w', I')$   
**proof** –  
**from** *assms* **have**  $stream-enc (w, I) = stream-enc (w', I')$   
**by** (*intro* *box-equals*[*OF* - *stream-enc-enc*[*symmetric*] *stream-enc-enc*[*symmetric*]])  
*auto*  
**thus** *?thesis* **using** *assms*(5) **by** *auto*  
**qed**

**lemma** *enc-eq-welldef*:  
 $\llbracket enc (w, I) = enc (w', I'); wf-formula (length I) \varphi; wf-interp-for-formula (w, I) \varphi; wf-interp-for-formula (w', I') \varphi \rrbracket \implies$   
 $(w, I) \models \varphi \longleftrightarrow (w', I') \models \varphi$   
**by** (*intro* *stream-enc-welldef*) (*auto* *simp* *del*: *stream-enc.simps* *intro*!: *enc-stream-enc*)

**lemma** *enc-welldef*:  
 $\llbracket x \in enc (w, I); x \in enc (w', I'); length I = length I'; wf-formula (length I) \varphi; wf-interp-for-formula (w, I) \varphi; wf-interp-for-formula (w', I') \varphi \rrbracket \implies$   
 $(w, I) \models \varphi \longleftrightarrow (w', I') \models \varphi$   
**by** (*intro* *enc-eq-welldef*[*OF* *enc-eqI*])

**lemma** *wf-rexp-of*:  $wf-formula n \varphi \implies wf n (rexp-of n \varphi)$   
**by** (*induct*  $\varphi$  *arbitrary*:  $n$ )  
(*auto* *intro*!: *wf-samequot-exec* *wf-rexp-ENC*,  
*auto* *simp*: *max-idx-vars* *finite-FOV*)

**theorem** *lang<sub>WS1S</sub>-rexp-of*:  $wf-formula n \varphi \implies lang_{WS1S} n \varphi = lang n (rexp-of n \varphi)$   
(*is* -  $\implies$  - = *?L*  $n \varphi$ )  
**proof** (*induct*  $\varphi$  *arbitrary*:  $n$ )  
**case** (*FQ*  $a m$ )  
**show** *?case*  
**proof** (*intro* *equalityI* *subsetI*)  
**fix**  $x$  **assume**  $x \in lang_{WS1S} n (FQ a m)$   
**then obtain**  $w I$  **where**  
 $*: x \in enc (w, I) wf-interp-for-formula (w, I) (FQ a m) length I = n (w, I)$   
 $\models FQ a m$   
**unfolding** *lang<sub>WS1S</sub>-def* **by** *blast*  
**hence**  $x-alt: x = map (case-prod (enc-atom I)) (zip [0 ..< length x] (stake (length x) (w @- sconst any)))$   
**by** (*intro* *encD*) *auto*  
**from** *FQ*(1) \*(2,4) **obtain**  $p$  **where**  $p: I ! m = Inl p$   
**by** (*auto* *simp*: *all-set-conv-all-nth* *split*: *sum.splits*)  
**with** *FQ*(1) \* **have**  $p-less: p < length x$   
**by** (*auto* *simp* *del*: *stream-enc.simps* *intro*: *trans-less-add1*[*OF* *less-length-cut-same-Inl*])  
**hence**  $enc-atom: x ! p = enc-atom I p ((w @- sconst any) !! p)$  (*is* - = *enc-atom* - - *?p*)  
**by** (*subst*  $x-alt$ , *simp*)  
**with** \*(1)  $p-less$ (1) **have**  $x = take p x @ [enc-atom I p ?p] @ drop (p + 1) x$   
**using** *id-take-nth-drop*[*of*  $p x$ ] **by** *auto*

```

moreover
from  $*(2,3,4)$   $FQ(1)$   $p$  have  $[enc\text{-}atom\ I\ p\ ?p] \in lang\ n\ (Atom\ (AQ\ m\ a))$ 
  by  $(auto\ simp\ del:\ lang.\ simps\ intro!:\ enc\text{-}atom\ lang\text{-}AQ)$ 
moreover from  $*(2,3)$  have  $take\ p\ x \in lang\ n\ (rexp.\ Not\ Zero)$ 
  by  $(subst\ x\text{-}alt)\ (auto\ simp:\ in\text{-}set\text{-}zip\ shift\text{-}snth\ intro!:\ enc\text{-}atom\ \sigma\ dest!:\ in\text{-}set\text{-}takeD)$ 
moreover from  $*(2,3)$  have  $drop\ (Suc\ p)\ x \in lang\ n\ (rexp.\ Not\ Zero)$ 
  by  $(subst\ x\text{-}alt)\ (auto\ simp:\ in\text{-}set\text{-}zip\ shift\text{-}snth\ intro!:\ enc\text{-}atom\ \sigma\ dest!:\ in\text{-}set\text{-}dropD)$ 
ultimately show  $x \in ?L\ n\ (FQ\ a\ m)$  using  $*(1,2,3)$ 
unfolding  $rexp\text{-}of.\ simps\ lang.\ simps(6,9)\ rexp\text{-}of\text{-}list.\ simps\ lang\text{-}ENC\text{-}formula[OF\ FQ]$ 
  by  $(auto\ elim:\ ssubst\ simp\ del:\ o\text{-}apply\ append.\ simps\ lang.\ simps\ enc.\ simps)$ 
next
fix  $x$  let  $?x = x\ @\text{-}\ sconst\ (any,\ replicate\ n\ False)$ 
assume  $x: x \in ?L\ n\ (FQ\ a\ m)$ 
with  $FQ$  obtain  $w\ I$  where
   $I: x \in enc\ (w,\ I)\ length\ I = n\ wf\text{-}interp\text{-}for\text{-}formula\ (w,\ I)\ (FQ\ a\ m)$ 
  unfolding  $rexp\text{-}of.\ simps\ lang.\ simps\ lang\text{-}ENC\text{-}formula[OF\ FQ]$  by  $fastforce$ 
hence  $stream\text{-}enc:\ stream\text{-}enc\ (w,\ I) = ?x$  using  $stream\text{-}enc\text{-}enc$  by  $auto$ 
from  $I\ FQ$  obtain  $p$  where  $m: I\ !\ m = Inl\ p\ m < length\ I$  by  $(auto\ split:\ sum.\ splits)$ 
with  $I$  have  $wf\text{-}interp\text{-}for\text{-}formula\ (dec\text{-}word\ ?x,\ stream\text{-}dec\ n\ \{m\}\ ?x)\ (FQ\ a\ m)$ 
unfolding  $I(1)$ 
  using  $enc\text{-}wf\text{-}interp[OF\ FQ(1)[folded\ I(2)]]$  by  $auto$ 
moreover
from  $x$  obtain  $u1\ u\ u2$  where  $x = u1\ @\ u\ @\ u2\ u \in lang\ n\ (Atom\ (AQ\ m\ a))$ 
  unfolding  $rexp\text{-}of.\ simps\ lang.\ simps\ rexp\text{-}of\text{-}list.\ simps$  using  $concE$  by  $fast$ 
with  $FQ(1)$  obtain  $v$  where  $v: x = u1\ @\ [v]\ @\ u2\ snd\ v\ !\ m\ fst\ v = a$ 
  using  $AQ\text{-}D[of\ u\ n\ m\ a]$  by  $fastforce$ 
from  $v$  have  $u: length\ u1 < length\ x$  by  $auto$ 
{ from  $v$  have  $snd\ (x\ !\ length\ u1)\ !\ m$  by  $auto$ 
  moreover
  from  $m\ I$  have  $p < length\ x\ snd\ (x\ !\ p)\ !\ m$  by  $(auto\ dest:\ enc\text{-}Inl\ simp\ del:\ enc.\ simps)$ 
  moreover
  from  $m\ I$  have  $ex1: \exists !p.\ snd\ (stream\text{-}enc\ (w,\ I)\ !!\ p)\ !\ m$  by  $(intro\ stream\text{-}enc\text{-}unique)\ auto$ 
  ultimately have  $p = length\ u1$  unfolding  $stream\text{-}enc$  using  $u\ I(3)$  by  $auto$ 
} note  $* = this$ 
from  $v$  have  $v = x\ !\ length\ u1$  by  $simp$ 
with  $u$  have  $?x\ !!\ length\ u1 = v$  by  $(auto\ simp:\ shift\text{-}snth)$ 
with  $*\ m\ I\ v(3)$  have  $(dec\text{-}word\ ?x,\ stream\text{-}dec\ n\ \{m\}\ ?x) \models FQ\ a\ m$ 
  using  $stream\text{-}enc\text{-}enc[OF\ -\ I(1),\ symmetric]\ less\text{-}length\text{-}cut\text{-}same[of\ w\ any\ length\ u1\ a]$ 
  by  $(auto\ simp\ del:\ enc.\ simps\ stream\text{-}enc.\ simps\ simp:\ dec\text{-}word\text{-}stream\text{-}enc\ dest!:\ stream\text{-}dec\text{-}enc\text{-}Inl\ stream\text{-}dec\text{-}not\text{-}Inr\ split:\ sum.\ splits)$ 

```

```

      (auto simp: smap2-alt cut-same-def)
    moreover from m I(2)
      have stream-enc-dec: stream-enc (dec-word (stream-enc (w, I)), stream-dec n
{m}) (stream-enc (w, I)) = stream-enc (w, I)
      by (intro stream-enc-dec)
      (auto simp: smap2-alt sdrop-snth shift-snth intro: stream-enc-unique,
      auto simp: smap2-szip stream.set-map)
    moreover from I have wf-word n x unfolding wf-word by (auto elim:
enc-set-σ simp del: enc.simps)
    ultimately show x ∈ langWS1S n (FQ a m) unfolding langWS1S-def using
m I(1,3)
      by (auto simp del: enc.simps stream-enc.simps intro!: exI[of - enc (dec-word
?x, stream-dec n {m} ?x)],
      fastforce simp del: enc.simps stream-enc.simps,
      auto simp del: stream-enc.simps simp: stream-enc[symmetric] I(2))
  qed
next
case (FLess m m')
show ?case
proof (cases m = m')
case False
thus ?thesis
proof (intro equalityI subsetI)
fix x assume x ∈ langWS1S n (FLess m m')
then obtain w I where
  *: x ∈ enc (w, I) wf-interp-for-formula (w, I) (FLess m m') length I = n
(w, I) ⊨ FLess m m'
  unfolding langWS1S-def by blast
  hence x-alt: x = map (case-prod (enc-atom I)) (zip [0 ..< length x] (stake
(length x) (w @- sconst any)))
  by (intro encD) auto
  from FLess(1) *(2,4) obtain p q where pq: I ! m = Inl p I ! m' = Inl q p
< q
  by (auto simp: all-set-conv-all-nth split: sum.splits)
  with FLess(1) *(1,2,3) have pq-less: p < length x q < length x
  by (auto simp del: stream-enc.simps intro!: trans-less-add1[OF less-length-cut-same-Inl])
  hence enc-atom: x ! p = enc-atom I p ((w @- sconst any) !! p) (is - =
enc-atom - - ?p)
  x ! q = enc-atom I q ((w @- sconst any) !! q) (is - = enc-atom
- - ?q) by (subst x-alt, simp)+
  with *(1) pq-less(1) have x = take p x @ [enc-atom I p ?p] @ drop (p + 1)
x
  using id-take-nth-drop[of p x] by auto
  also have drop (p + 1) x = take (q - p - 1) (drop (p + 1) x) @
[enc-atom I q ?q] @ drop (q - p) (drop (p + 1) x) (is - = ?LHS)
  using id-take-nth-drop[of q - p - 1 drop (p + 1) x] pq pq-less(2) enc-atom(2)
by auto
  finally have x = take p x @ [enc-atom I p ?p] @ ?LHS .
  moreover from *(2,3) FLess(1) pq(1)

```

```

    have [enc-atom I p ?p] ∈ lang n (Atom (Arbitrary-Except m True))
      by (intro enc-atom-lang-Arbitrary-Except-True) (auto simp: shift-snth)
  moreover from *(2,3) FLess(1) pq(2)
    have [enc-atom I q ?q] ∈ lang n (Atom (Arbitrary-Except m' True))
      by (intro enc-atom-lang-Arbitrary-Except-True) (auto simp: shift-snth)
  moreover from *(2,3) have take p x ∈ lang n (rexp.Not Zero)
    by (subst x-alt) (auto simp: in-set-zip shift-snth intro!: enc-atom-σ dest!:
in-set-takeD)
  moreover from *(2,3) have take (q - p - 1) (drop (Suc p) x) ∈ lang n
(rexp.Not Zero)
    by (subst x-alt) (auto simp: in-set-zip shift-snth intro!: enc-atom-σ dest!:
in-set-dropD in-set-takeD)
  moreover from *(2,3) have drop (q - p) (drop (Suc p) x) ∈ lang n (rexp.Not
Zero)
    by (subst x-alt) (auto simp: in-set-zip shift-snth intro!: enc-atom-σ dest!:
in-set-dropD)
  ultimately show x ∈ ?L n (FLess m m') using *(1,2,3)
  unfolding rexp-of.simps lang.simps(6,9) rexp-of-list.simps Int-Diff lang-ENC-formula[OF
FLess] if-not-P[OF False]
    by (auto elim: ssubst simp del: o-apply append.simps lang.simps enc.simps)
next
fix x let ?x = x @- sconst (any, replicate n False)
assume x: x ∈ ?L n (FLess m m')
with FLess obtain w I where
  I: x ∈ enc (w, I) length I = n wf-interp-for-formula (w, I) (FLess m m')
unfolding rexp-of.simps lang.simps lang-ENC-formula[OF FLess] if-not-P[OF
False] by fastforce
hence stream-enc: stream-enc (w, I) = x @- sconst (any, replicate n False)
using stream-enc-enc by auto
from I FLess obtain p p' where m: I ! m = Inl p m < length I I ! m' =
Inl p' m' < length I
  by (auto split: sum.splits)
with I have wf-interp-for-formula (dec-word ?x, stream-dec n {m, m'} ?x)
(FLess m m') unfolding I(1)
  using enc-wf-interp[OF FLess(1)][folded I(2)] by auto
moreover
from x obtain u1 u2 u' u3 where x = u1 @ u @ u2 @ u' @ u3
  u ∈ lang n (Atom (Arbitrary-Except m True)) u' ∈ lang n (Atom (Arbitrary-Except
m' True))
  unfolding rexp-of.simps lang.simps rexp-of-list.simps if-not-P[OF False]
using concE by fast
with FLess(1) obtain v v' where v: x = u1 @ [v] @ u2 @ [v'] @ u3
  snd v ! m snd v' ! m' fst v ∈ set Σ fst v' ∈ set Σ
  using Arbitrary-ExceptD[of u n m True] Arbitrary-ExceptD[of u' n m' True]
  by simp (auto simp: σ-def)
hence u: length u1 < length x and u': Suc (length u1 + length u2) < length
x (is ?u' < -) by auto
{ from v have snd (x ! length u1) ! m by auto
  moreover

```

```

    from m I have p < length x snd (x ! p) ! m by (auto dest: enc-Inl simp
del: enc.simps)
    moreover
      from m I have ex1: ∃!p. snd (stream-enc (w, I) !! p) ! m by (intro
stream-enc-unique) auto
      ultimately have p = length u1 unfolding stream-enc using u I(3) by
auto
    }
    { from v have snd (x ! ?u') ! m' by (auto simp: nth-append)
      moreover
        from m I have p' < length x snd (x ! p') ! m' by (auto dest: enc-Inl simp
del: enc.simps)
        moreover
          from m I have ex1: ∃!p. snd (stream-enc (w, I) !! p) ! m' unfolding I(1)
by (intro stream-enc-unique) auto
          ultimately have p' = ?u' unfolding stream-enc using u' I(3) by auto
(metis shift-snth-less)
        } note * = this ⟨p = length u1⟩
        with m I have (dec-word ?x, stream-dec n {m, m'} ?x) |= FLess m m'
        using stream-enc-enc[OF - I(1), symmetric]
        by (auto dest: stream-dec-not-Inr stream-dec-enc-Inl split: sum.splits simp
del: stream-enc.simps)
        moreover from m I(2)
          have stream-enc-dec: stream-enc (dec-word (stream-enc (w, I)), stream-dec
n {m, m'} (stream-enc (w, I))) = stream-enc (w, I)
          by (intro stream-enc-dec)
             (auto simp: smap2-alt sdrop-snth shift-snth intro: stream-enc-unique,
auto simp: smap2-szip stream.set-map)
          moreover from I have wf-word n x unfolding wf-word by (auto elim:
enc-set-σ simp del: enc.simps)
          ultimately show x ∈ langWS1S n (FLess m m') unfolding langWS1S-def
using m I(1,3)
          by (auto simp del: enc.simps stream-enc.simps intro!: exI[of - enc (dec-word
?x, stream-dec n {m, m'} ?x)],
fastforce simp del: enc.simps stream-enc.simps,
auto simp del: stream-enc.simps simp: stream-enc[symmetric] I(2))
        qed
      qed (simp add: langWS1S-def del: o-apply)
    next
      case (FIn m M)
      show ?case
      proof (intro equalityI subsetI)
        fix x assume x ∈ langWS1S n (FIn m M)
        then obtain w I where
          *: x ∈ enc (w, I) wf-interp-for-formula (w, I) (FIn m M) length I = n (w,
I) |= FIn m M
          unfolding langWS1S-def by blast
          hence x-alt: x = map (case-prod (enc-atom I)) (zip [0 ..< length x] (stake
(length x) (w @- sconst any)))

```



**by** (*intro encD*) *auto*  
**from**  $FIn(1) * (2,4)$  **obtain**  $p P$  **where**  $p: I ! m = Inl p I ! M = Inr P p \in P$   
**by** (*auto simp: all-set-conv-all-nth split: sum.splits*)  
**with**  $FIn(1) * (1,2,3)$  **have**  $p\text{-less}: p < length\ x \ \forall p \in P. p < length\ x$   
**by** (*auto simp del: stream-enc.simps intro: trans-less-add1 [OF less-length-cut-same-Inl]*  
*trans-less-add1 [OF bspec [OF less-length-cut-same-Inr]]*)  
**hence**  $enc\text{-atom}: x ! p = enc\text{-atom}\ I\ p\ ((w\ @\ -\ sconst\ any) !! p)$  (**is**  $=$   
 $enc\text{-atom}\ -\ -\ ?p$ )  
 $\forall p \in P. x ! p = enc\text{-atom}\ I\ p\ ((w\ @\ -\ sconst\ any) !! p)$  (**is**  $Ball\ -\ (\lambda p. -$   
 $=\ enc\text{-atom}\ -\ -\ (?P\ p))$ )  
**by** (*subst x-alt, simp*)  
**with**  $* (1)\ p\text{-less}(1)$  **have**  $x = take\ p\ x\ @\ [enc\text{-atom}\ I\ p\ ?p]\ @\ drop\ (p + 1)\ x$   
**using** *id-take-nth-drop [of p x]* **by** *auto*  
**moreover**  
**from**  $* (2,3)\ FIn(1)\ p$  **have**  $[enc\text{-atom}\ I\ p\ ?p] \in lang\ n\ (Atom\ (Arbitrary\ Except2\ m\ M))$   
**by** (*intro enc-atom-lang-Arbitrary-Except2*) (*auto simp: shift-snth*)  
**moreover from**  $* (2,3)$  **have**  $take\ p\ x \in lang\ n\ (rexp.\ Not\ Zero)$   
**by** (*subst x-alt*) (*auto simp: in-set-zip shift-snth intro!: enc-atom- $\sigma$  dest!: in-set-takeD*)  
**moreover from**  $* (2,3)$  **have**  $drop\ (Suc\ p)\ x \in lang\ n\ (rexp.\ Not\ Zero)$   
**by** (*subst x-alt*) (*auto simp: in-set-zip shift-snth intro!: enc-atom- $\sigma$  dest!: in-set-dropD*)  
**ultimately show**  $x \in ?L\ n\ (FIn\ m\ M)$  **using**  $* (1,2,3)$   
**unfolding** *rexp-of.simps lang.simps(6,9) rexp-of-list.simps Int-Diff lang-ENC-formula [OF FIn]*  
**by** (*auto elim: ssubst simp del: o-apply append.simps lang.simps enc.simps*)  
**next**  
**fix**  $x$  **let**  $?x = x\ @\ -\ sconst\ (any, replicate\ n\ False)$   
**assume**  $x: x \in ?L\ n\ (FIn\ m\ M)$   
**with**  $FIn$  **obtain**  $w\ I$  **where**  
 $I: x \in enc\ (w, I)\ length\ I = n\ wf\text{-interp-for-formula}\ (w, I)\ (FIn\ m\ M)$   
**unfolding** *rexp-of.simps lang.simps lang-ENC-formula [OF FIn]* **by** *fastforce*  
**hence** *stream-enc: stream-enc*  $(w, I) = ?x$  **using** *stream-enc-enc* **by** *auto*  
**from**  $I\ FIn$  **obtain**  $p\ P$  **where**  $m: I ! m = Inl\ p\ m < length\ I\ I ! M = Inr\ P\ M < length\ I$   
**by** (*auto split: sum.splits*)  
**with**  $I$  **have** *wf-interp-for-formula*  $(dec\text{-word}\ ?x, stream\text{-dec}\ n\ \{m\}\ ?x)$   $(FIn\ m\ M)$  **unfolding**  $I(1)$   
**using** *enc-wf-interp [OF FIn(1) [folded I(2)]]* **by** *auto*  
**moreover**  
**from**  $x$  **obtain**  $u1\ u\ u2$  **where**  $x = u1\ @\ u\ @\ u2$   
 $u \in lang\ n\ (Atom\ (Arbitrary\ Except2\ m\ M))$   
**unfolding** *rexp-of.simps lang.simps rexp-of-list.simps* **using** *concE* **by** *fast*  
**with**  $FIn(1)$  **obtain**  $v$  **where**  $v: x = u1\ @\ [v]\ @\ u2\ snd\ v ! m\ snd\ v ! M$  **and**  
 $fst\ v \in set\ \Sigma$   
**using** *Arbitrary-Except2D [of u n m M]* **by** *simp (auto simp:  $\sigma$ -def)*  
**from**  $v$  **have**  $u: length\ u1 < length\ x$  **by** *auto*  
**{ from**  $v$  **have**  $snd\ (x ! length\ u1) ! m$  **by** *auto*

**moreover**  
**from**  $m I$  **have**  $p < \text{length } x \text{ snd } (x ! p) ! m$  **by** (auto dest: enc-Inl simp del: enc.simps)  
**moreover**  
**from**  $m I$  **have**  $ex1: \exists ! p. \text{snd } (\text{stream-enc } (w, I) !! p) ! m$  **by** (intro stream-enc-unique) auto  
**ultimately have**  $p = \text{length } u1$  **unfolding** stream-enc **using**  $u I(3)$  **by** auto  
**} note**  $* = \text{this}$   
**from**  $v$  **have**  $v = x ! \text{length } u1$  **by** simp  
**with**  $v(3) m(3,4) u I(1,3)$  **have**  $\text{length } u1 \in P$  **by** (auto dest!: enc-Inr simp del: enc.simps)  
**with**  $* m I$  **have** (dec-word  $?x$ , stream-dec  $n \{m\} ?x$ )  $\models \text{FIn } m M$   
**using** stream-enc-enc[OF -  $I(1)$ , symmetric]  
**by** (auto simp del: stream-enc.simps dest: stream-dec-not-Inr stream-dec-not-Inl stream-dec-enc-Inl stream-dec-enc-Inr split: sum.splits)  
**moreover from**  $m I(2)$   
**have** stream-enc-dec: stream-enc (dec-word (stream-enc (w, I)), stream-dec  $n \{m\}$  (stream-enc (w, I))) = stream-enc (w, I)  
**by** (intro stream-enc-dec)  
(auto simp: smap2-alt sdrop-snth shift-snth intro: stream-enc-unique, auto simp: smap2-szip stream.set-map)  
**moreover from**  $I$  **have** wf-word  $n x$  **unfolding** wf-word **by** (auto elim: enc-set- $\sigma$  simp del: enc.simps)  
**ultimately show**  $x \in \text{lang}_{WS1S} n (\text{FIn } m M)$  **unfolding** lang<sub>WS1S</sub>-def **using**  $m I(1,3)$   
**by** (auto simp del: enc.simps stream-enc.simps intro!: exI[of - enc (dec-word  $?x$ , stream-dec  $n \{m\} ?x$ )], fastforce simp del: enc.simps stream-enc.simps, auto simp del: stream-enc.simps simp: stream-enc[symmetric]  $I(2)$ )  
**qed**  
**next**  
**case** (FOr  $\varphi_1 \varphi_2$ )  
**from** FOr(3) **have** IH1: lang<sub>WS1S</sub>  $n \varphi_1 = \text{lang } n (\text{rexp-of } n \varphi_1)$   
**by** (intro FOr(1)) auto  
**from** FOr(3) **have** IH2: lang<sub>WS1S</sub>  $n \varphi_2 = \text{lang } n (\text{rexp-of } n \varphi_2)$   
**by** (intro FOr(2)) auto  
**show** ?case  
**proof** (intro equalityI subsetI)  
**fix**  $x$  **assume**  $x \in \text{lang}_{WS1S} n (\text{FOr } \varphi_1 \varphi_2)$  **thus**  $x \in \text{lang } n (\text{rexp-of } n (\text{FOr } \varphi_1 \varphi_2))$   
**using** lang<sub>WS1S</sub>-FOr[OF FOr(3)] **unfolding** lang-ENC-formula[OF FOr(3)]  
rexp-of.simps lang.simps IH1 IH2 **by** blast  
**next**  
**fix**  $x$  **assume**  $x \in \text{lang } n (\text{rexp-of } n (\text{FOr } \varphi_1 \varphi_2))$   
**then obtain**  $w I$  **where**  $or: x \in \text{lang}_{WS1S} n \varphi_1 \vee x \in \text{lang}_{WS1S} n \varphi_2$  **and**  
 $I: x \in \text{enc } (w, I) \text{ length } I = n$   
wf-interp-for-formula (w, I) (FOr  $\varphi_1 \varphi_2$ )  
**unfolding** lang-ENC-formula[OF FOr(3)] rexp-of.simps lang.simps IH1 IH2  
Int-Diff **by** auto

```

have (w, I) ⊨ φ1 ∨ (w, I) ⊨ φ2
proof (intro mp[OF disj-mono[OF impI impI] or])
  assume x ∈ langWS1S n φ1
  with I FOr(3) show (w, I) ⊨ φ1
    unfolding langWS1S-def I(1) wf-interp-for-formula-FOr
    by (auto dest!: enc-welldef[of x w I - - φ1] simp del: enc.simps)
next
  assume x ∈ langWS1S n φ2
  with I FOr(3) show (w, I) ⊨ φ2
    unfolding langWS1S-def I(1) wf-interp-for-formula-FOr
    by (auto dest!: enc-welldef[of x w I - - φ2] simp del: enc.simps)
qed
with I show x ∈ langWS1S n (FOr φ1 φ2) unfolding langWS1S-def by auto
qed
next
  case (FAnd φ1 φ2)
  from FAnd(3) have IH1: langWS1S n φ1 = lang n (rexp-of n φ1)
    by (intro FAnd(1)) auto
  from FAnd(3) have IH2: langWS1S n φ2 = lang n (rexp-of n φ2)
    by (intro FAnd(2)) auto
  show ?case
  proof (intro equalityI subsetI)
    fix x assume x ∈ langWS1S n (FAnd φ1 φ2) thus x ∈ lang n (rexp-of n (FAnd
φ1 φ2))
    using langWS1S-FAnd[OF FAnd(3)]
    unfolding lang-ENC-formula[OF FAnd(3)] rexp-of.simps rexp-of-list.simps
lang.simps IH1 IH2 Int-assoc
    by blast
  next
    fix x assume x ∈ lang n (rexp-of n (FAnd φ1 φ2))
    then obtain w I where and: x ∈ langWS1S n φ1 ∧ x ∈ langWS1S n φ2 and
I: x ∈ enc (w, I) length I = n
    wf-interp-for-formula (w, I) (FAnd φ1 φ2)
    unfolding lang-ENC-formula[OF FAnd(3)] rexp-of.simps rexp-of-list.simps
lang.simps IH1 IH2 Int-Diff by auto
    have (w, I) ⊨ φ1 ∧ (w, I) ⊨ φ2
    proof (intro mp[OF conj-mono[OF impI impI] and])
      assume x ∈ langWS1S n φ1
      with I FAnd(3) show (w, I) ⊨ φ1
        unfolding langWS1S-def I(1) wf-interp-for-formula-FAnd
        by (auto dest!: enc-welldef[of x w I - - φ1] simp del: enc.simps)
    next
      assume x ∈ langWS1S n φ2
      with I FAnd(3) show (w, I) ⊨ φ2
        unfolding langWS1S-def I(1) wf-interp-for-formula-FAnd
        by (auto dest!: enc-welldef[of x w I - - φ2] simp del: enc.simps)
    qed
    with I show x ∈ langWS1S n (FAnd φ1 φ2) unfolding langWS1S-def by
auto

```

```

qed
next
case (FNot  $\varphi$ )
hence IH:  $?L\ n\ \varphi = \text{lang}_{WS1S}\ n\ \varphi$  by simp
show ?case
proof (intro equalityI subsetI)
  fix x assume  $x \in \text{lang}_{WS1S}\ n\ (FNot\ \varphi)$ 
  then obtain w I where
    *:  $x \in \text{enc}\ (w, I)\ \text{wf-interp-for-formula}\ (w, I)\ \varphi\ \text{length}\ I = n$  and  $\text{unsat}: \neg$ 
     $(w, I) \models \varphi$ 
    unfolding langWS1S-def by auto
    { assume  $x \in ?L\ n\ \varphi$ 
      hence  $(w, I) \models \varphi$  using enc-welldef[of x w I - -  $\varphi$ , OF *(1) - - - *(2)]
    FNot(2)
    unfolding *(3) langWS1S-def IH by auto
    }
  with unsat have  $x \notin ?L\ n\ \varphi$  by blast
  with * show  $x \in ?L\ n\ (FNot\ \varphi)$  unfolding rexp-of.simps lang.simps using
  lang-ENC-formula[OF FNot(2)]
  by (auto simp del: enc.simps simp: comp-def intro: enc-set- $\sigma$ )
next
fix x assume  $x \in ?L\ n\ (FNot\ \varphi)$ 
with IH have  $x \in \text{lang}\ n\ (ENC\ n\ (FOV\ (FNot\ \varphi)))$  and  $x: x \notin \text{lang}_{WS1S}\ n$ 
 $\varphi$  by (auto simp del: o-apply)
then obtain w I where *:  $x \in \text{enc}\ (w, I)\ \text{wf-interp-for-formula}\ (w, I)\ (FNot\ \varphi)$ 
 $\text{length}\ I = n$ 
unfolding lang-ENC-formula[OF FNot(2)] by blast
{ assume  $\neg (w, I) \models FNot\ \varphi$ 
  with * have  $x \in \text{lang}_{WS1S}\ n\ \varphi$  unfolding langWS1S-def by auto
}
with x * show  $x \in \text{lang}_{WS1S}\ n\ (FNot\ \varphi)$  unfolding langWS1S-def by blast
qed
next
case (FExists  $\varphi$ )
have  $\sigma: (\text{any, replicate}\ n\ \text{False}) \in (\text{set}\ o\ \sigma\ \Sigma)\ n$  by (auto simp:  $\sigma$ -def set-n-lists
image-iff)
from FExists(2) have wf:  $\text{wf}\ n\ (\text{Pr}\ (\text{rexp-of}\ (\text{Suc}\ n)\ \varphi))$  by (fastforce intro:
wf-rexp-of)
note lang-quot = lang-samequot-exec[OF wf  $\sigma$ ]
show ?case
proof (intro equalityI subsetI)
  fix x assume  $x \in \text{lang}_{WS1S}\ n\ (FExists\ \varphi)$ 
  then obtain w I p where
    *:  $x \in \text{enc}\ (w, I)\ \text{wf-interp-for-formula}\ (w, I)\ (FExists\ \varphi)\ \text{length}\ I = n$  ( $w,$ 
 $\text{Inl}\ p\ \# I) \models \varphi$ 
    unfolding langWS1S-def by auto
  with FExists(2) have  $\text{enc}\ (w, \text{Inl}\ p\ \# I) \subseteq ?L\ (\text{Suc}\ n)\ \varphi$ 
  by (subst FExists(1)[of Suc n, symmetric])
  (fastforce simp del: enc.simps simp: langWS1S-def nth-Cons' intro!: exI[of

```

-  $enc(w, Inl\ p\ \# I)]+$   
**thus**  $x \in ?L\ n\ (FExists\ \varphi)$  **using**  $*(1,2,3)$   
**by**  $(auto\ simp:\ lang\ quot\ simp\ del:\ o\ apply\ enc.\ simps\ elim:\ subsetD[OF\ SAMEQUOT\ mono[OF\ image\ mono]])$   
**next**  
**fix**  $x$  **assume**  $x \in ?L\ n\ (FExists\ \varphi)$   
**then obtain**  $x'\ m$  **where**  $x' \in ?L\ (Suc\ n)\ \varphi$  **and**  
 $x:\ x = fin\ cut\ same\ (any,\ replicate\ n\ False)\ (map\ \pi\ x')\ @\ replicate\ m\ (any,\ replicate\ n\ False)$   
**by**  $(auto\ simp:\ lang\ quot\ SAMEQUOT\ def\ simp\ del:\ o\ apply\ enc.\ simps)$   
**with**  $FExists(2)$  **have**  $x' \in lang_{WS1S}\ (Suc\ n)\ \varphi$   
**by**  $(intro\ subsetD[OF\ equalityD2[OF\ FExists(1)],\ of\ Suc\ n\ x'])$   
 $(auto\ split:\ if\ split\ asm\ sum.\ splits)$   
**then obtain**  $w\ I'$  **where**  
 $*:\ x' \in enc(w, I')\ wf\ interp\ for\ formula\ (w, I')\ \varphi\ length\ I' = Suc\ n\ (w, I')$   
 $\models \varphi$   
**unfolding**  $lang_{WS1S}\ def$  **by**  $blast$   
**moreover then obtain**  $I_0\ I$  **where**  $I' = I_0\ \# I$  **by**  $(cases\ I')\ auto$   
**moreover with**  $FExists(2)\ *(2)$  **obtain**  $p$  **where**  $I_0 = Inl\ p$   
**by**  $(auto\ simp:\ nth\ Cons'\ split:\ sum.\ splits\ if\ split\ asm)$   
**ultimately have**  $x \in enc(w, I)\ wf\ interp\ for\ formula\ (w, I)\ (FExists\ \varphi)$   
 $length\ I = n$   
 $(w, I) \models FExists\ \varphi$  **using**  $FExists(2)\ fin\ cut\ same\ tl[OF\ ex\ Loop\ stream\ enc,\ of\ Inl\ p\ \# I\ w]$   
**unfolding**  $x$  **by**  $(auto\ simp\ add:\ \pi\ def\ nth\ Cons'\ split:\ if\ split\ asm)$   
**thus**  $x \in lang_{WS1S}\ n\ (FExists\ \varphi)$  **unfolding**  $lang_{WS1S}\ def$  **by**  $(auto\ intro!:\ exI[of\ -\ I])$   
**qed**  
**next**  
**case**  $(FEXISTS\ \varphi)$   
**have**  $\sigma:\ (any,\ replicate\ n\ False) \in (set\ o\ \sigma\ \Sigma)\ n$  **by**  $(auto\ simp:\ \sigma\ def\ set\ n\ lists\ image\ iff)$   
**from**  $FEXISTS(2)$  **have**  $wf:\ wf\ n\ (Pr\ (rexp\ of\ (Suc\ n)\ \varphi))$  **by**  $(fastforce\ intro:\ wf\ rexp\ of)$   
**note**  $lang\ quot = lang\ samequot\ exec[OF\ wf\ \sigma]$   
**show**  $?case$   
**proof**  $(intro\ equalityI\ subsetI)$   
**fix**  $x$  **assume**  $x \in lang_{WS1S}\ n\ (FEXISTS\ \varphi)$   
**then obtain**  $w\ I\ P$  **where**  
 $*:\ x \in enc(w, I)\ wf\ interp\ for\ formula\ (w, I)\ (FEXISTS\ \varphi)\ length\ I = n$   
 $finite\ P\ (w, Inr\ P\ \# I) \models \varphi$   
**unfolding**  $lang_{WS1S}\ def$  **by**  $auto$   
**with**  $FEXISTS(2)$  **have**  $enc(w, Inr\ P\ \# I) \subseteq ?L\ (Suc\ n)\ \varphi$   
**by**  $(subst\ FEXISTS(1)[of\ Suc\ n,\ symmetric])$   
 $(fastforce\ simp\ del:\ enc.\ simps\ simp:\ lang_{WS1S}\ def\ nth\ Cons'\ intro!:\ exI[of\ -\ enc(w, Inr\ P\ \# I)]+$   
**thus**  $x \in ?L\ n\ (FEXISTS\ \varphi)$  **using**  $*(1,2,3,4)$   
**by**  $(auto\ simp:\ lang\ quot\ simp\ del:\ o\ apply\ enc.\ simps\ elim:\ subsetD[OF\ SAMEQUOT\ mono[OF\ image\ mono]])$

**next**  
**fix**  $x$  **assume**  $x \in ?L\ n$  ( $FEXISTS\ \varphi$ )  
**then obtain**  $x' m$  **where**  $x' \in ?L\ (Suc\ n)\ \varphi$  **and**  
 $x: x = \text{fin-cut-same}$  (*any, replicate n False*) ( $\text{map } \pi\ x'$ )  $\text{@}$  *replicate m* (*any, replicate n False*)  
**by** (*auto simp: lang-quot SAMEQUOT-def simp del: o-apply enc.simps*)  
**with**  $FEXISTS(2)$  **have**  $x' \in \text{lang}_{WS1S}\ (Suc\ n)\ \varphi$   
**by** (*intro subsetD[OF equalityD2[OF FEXISTS(1)], of Suc n x']*)  
(*auto split: if-split-asm sum.splits*)  
**then obtain**  $w\ I'$  **where**  
 $*: x' \in \text{enc}\ (w, I')\ \text{wf-interp-for-formula}\ (w, I')\ \varphi\ \text{length } I' = Suc\ n\ (w, I')$   
 $\models \varphi$   
**unfolding**  $\text{lang}_{WS1S}\text{-def}$  **by** *blast*  
**moreover then obtain**  $I_0\ I$  **where**  $I' = I_0 \# I$  **by** (*cases I'*) *auto*  
**moreover with**  $FEXISTS(2)\ *(2)$  **obtain**  $P$  **where**  $I_0 = \text{Inr } P$  *finite P*  
**by** (*auto simp: nth-Cons' split: sum.splits if-split-asm*)  
**ultimately have**  $x \in \text{enc}\ (w, I)\ \text{wf-interp-for-formula}\ (w, I)\ (FEXISTS\ \varphi)$   
 $\text{length } I = n$   
 $(w, I) \models FEXISTS\ \varphi$  **using**  $FEXISTS(2)\ \text{fin-cut-same-tl}$  [*OF ex-Loop-stream-enc, of Inr P # I*]  
**unfolding**  $x$  **by** (*auto simp: nth-Cons'  $\pi$ -def split: if-split-asm*)  
**thus**  $x \in \text{lang}_{WS1S}\ n\ (FEXISTS\ \varphi)$  **unfolding**  $\text{lang}_{WS1S}\text{-def}$  **by** (*auto intro!: exI[of - I]*)  
**qed**  
**qed**

**lemma**  $\text{wf-rexp-of-alt}: \text{wf-formula } n\ \varphi \implies \text{wf } n\ (\text{rexp-of-alt } n\ \varphi)$   
**by** (*induct  $\varphi$  arbitrary: n*)  
(*auto intro!: wf-samequot-exec wf-rexp-ENC,*  
*auto simp: max-idx-vars finite-FOV*)

**lemma**  $\text{wf-rexp-of}' : \text{wf-formula } n\ \varphi \implies \text{wf } n\ (\text{rexp-of}'\ n\ \varphi)$   
**unfolding**  $\text{rexp-of}'\text{-def}$  **by** (*auto simp: max-idx-vars intro: wf-rexp-of-alt wf-rexp-ENC[OF finite-FOV]*)

**lemma**  $\text{wf-rexp-of-alt}' : \text{wf-formula } n\ \varphi \implies \text{wf } n\ (\text{rexp-of-alt}'\ n\ \varphi)$   
**by** (*induct  $\varphi$  arbitrary: n*)  
(*auto intro!: wf-samequot-exec wf-rexp-ENC,*  
*auto simp: max-idx-vars finite-FOV*)

**lemma**  $\text{wf-rexp-of}'' : \text{wf-formula } n\ \varphi \implies \text{wf } n\ (\text{rexp-of}''\ n\ \varphi)$   
**unfolding**  $\text{rexp-of}''\text{-def}$  **by** (*auto simp: wf-rexp-ENC wf-rexp-of-alt' finite-FOV max-idx-vars*)

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

**lemma**  $\text{ENC-FAnd}$ :

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

**proof**

**fix**  $x$  **assume**  $wf: wf\text{-formula } n (FAnd \varphi \psi)$  **and**  $x: x \in lang\ n (ENC\ n (FOV (FAnd \varphi \psi)))$

**hence**  $wf1: wf\text{-formula } n \varphi$  **and**  $wf2: wf\text{-formula } n \psi$  **by** *auto*

**from**  $x$  **obtain**  $w\ I$  **where**  $I: x \in enc\ (w, I)\ wf\text{-interp-for-formula}\ (w, I)\ (FAnd\ \varphi\ \psi)\ length\ I = n$

**using**  $lang\text{-ENC-formula}[OF\ wf]$  **by** *auto*

**hence**  $wf\text{-interp-for-formula}\ (w, I)\ \varphi\ wf\text{-interp-for-formula}\ (w, I)\ \psi$

**using**  $wf\text{-interp-for-formula-FAnd}$  **by** *auto*

**thus**  $x \in lang\ n (ENC\ n (FOV \varphi)) \cap lang\ n (ENC\ n (FOV \psi))$

**unfolding**  $lang\text{-ENC-formula}[OF\ wf1]\ lang\text{-ENC-formula}[OF\ wf2]$  **using**  $I$  **by**

*blast*

**qed**

**lemma** *ENC-FOr*:

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

**proof**

**fix**  $x$  **assume**  $wf: wf\text{-formula } n (FOr \varphi \psi)$  **and**  $x: x \in lang\ n (ENC\ n (FOV (FOr \varphi \psi)))$

**hence**  $wf1: wf\text{-formula } n \varphi$  **and**  $wf2: wf\text{-formula } n \psi$  **by** *auto*

**from**  $x$  **obtain**  $w\ I$  **where**  $I: x \in enc\ (w, I)\ wf\text{-interp-for-formula}\ (w, I)\ (FOr\ \varphi\ \psi)\ length\ I = n$

**using**  $lang\text{-ENC-formula}[OF\ wf]$  **by** *auto*

**hence**  $wf\text{-interp-for-formula}\ (w, I)\ \varphi\ wf\text{-interp-for-formula}\ (w, I)\ \psi$

**using**  $wf\text{-interp-for-formula-FOr}$  **by** *auto*

**thus**  $x \in lang\ n (ENC\ n (FOV \varphi)) \cap lang\ n (ENC\ n (FOV \psi))$

**unfolding**  $lang\text{-ENC-formula}[OF\ wf1]\ lang\text{-ENC-formula}[OF\ wf2]$  **using**  $I$  **by**

*blast*

**qed**

**lemma** *ENC-FExists*:

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

**proof** (*intro equalityI subsetI*)

**fix**  $x$  **assume**  $wf: wf\text{-formula } n (FExists \varphi)$  **and**  $x: x \in ?L$

**hence**  $wf1: wf\text{-formula}\ (Suc\ n)\ \varphi$  **by** *auto*

**from**  $x$  **obtain**  $w\ I$  **where**  $I: x \in enc\ (w, I)\ wf\text{-interp-for-formula}\ (w, I)\ (FExists\ \varphi)\ length\ I = n$

**using**  $lang\text{-ENC-formula}[OF\ wf]$  **by** *auto*

**from**  $I(2)$  **obtain**  $p$  **where**  $wf\text{-interp-for-formula}\ (w, Inl\ p\ \# I)\ \varphi$

**using**  $wf\text{-interp-for-formula-FExists}[OF\ wf[folded\ I(3)]]$  **by** *blast*

**with**  $I(3)$  **show**  $x \in ?R$

**unfolding**  $lang\text{-ENC-formula}[OF\ wf1]$  **using**  $I(1)\ tl\text{-enc}[of\ Inl\ p\ I, symmetric]$

**by** (*simp del: enc.simps*)

(*fastforce simp del: enc.simps elim!: rev-subsetD[OF - SAMEQUOT-mono[OF*

*image-mono*]]  
 intro: exI[of - enc (w, Inl p # I)])  
**next**  
**fix** x **assume** wf: wf-formula n (FExists  $\varphi$ ) **and** x: x  $\in$  ?R  
**hence** wf1: wf-formula (Suc n)  $\varphi$  **and** 0  $\in$  FOV  $\varphi$  **by** auto  
**from** x **obtain** w I **where** I: x  $\in$  SAMEQUOT (any, replicate n False) (map  
 $\pi$  ' enc (w, I))  
 wf-interp-for-formula (w, I)  $\varphi$  length I = Suc n  
**using** lang-ENC-formula[OF wf1] **unfolding** SAMEQUOT-def **by** fast  
**with** ⟨0  $\in$  FOV  $\varphi$ ⟩ **obtain** p I' **where** I': I = Inl p # I' **by** (cases I) (fastforce  
 split: sum.splits)+  
**with** I **have** wtlI: x  $\in$  enc (w, I') length I' = n **using** tl-enc[of Inl p I' w] **by**  
 auto  
**have** wf-interp-for-formula (w, I') (FExists  $\varphi$ )  
**using** wf-interp-for-formula-FExists[OF wf[folded wtlI(2)]]  
 wf-interp-for-formula-any-Inl[OF I(2)][unfolded I'] ..  
**with** wtlI **show** x  $\in$  ?L **unfolding** lang-ENC-formula[OF wf] **by** blast  
**qed**

**lemma** ENC-FEXISTS:

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

**proof** (intro equalityI subsetI)

**fix** x **assume** wf: wf-formula n (FEXISTS  $\varphi$ ) **and** x: x  $\in$  ?L  
**hence** wf1: wf-formula (Suc n)  $\varphi$  **by** auto  
**from** x **obtain** w I **where** I: x  $\in$  enc (w, I) wf-interp-for-formula (w, I)  
 (FEXISTS  $\varphi$ ) length I = n  
**using** lang-ENC-formula[OF wf] **by** auto  
**from** I(2) **obtain** P **where** wf-interp-for-formula (w, Inr P # I)  $\varphi$   
**using** wf-interp-for-formula-FEXISTS[OF wf[folded I(3)]] **by** blast  
**with** I(3) **show** x  $\in$  ?R  
**unfolding** lang-ENC-formula[OF wf1] **using** I(1) tl-enc[of Inr P I, symmetric]  
**by** (simp del: enc.simps)  
 (fastforce simp del: enc.simps elim!: rev-subsetD[OF - SAMEQUOT-mono[OF  
*image-mono*]]  
 intro: exI[of - enc (w, Inr P # I)])

**next**

**fix** x **assume** wf: wf-formula n (FEXISTS  $\varphi$ ) **and** x: x  $\in$  ?R  
**hence** wf1: wf-formula (Suc n)  $\varphi$  **and** 0  $\in$  SOV  $\varphi$  **by** auto  
**from** x **obtain** w I **where** I: x  $\in$  SAMEQUOT (any, replicate n False) (map  
 $\pi$  ' enc (w, I))  
 wf-interp-for-formula (w, I)  $\varphi$  length I = Suc n  
**using** lang-ENC-formula[OF wf1] **unfolding** SAMEQUOT-def **by** fast  
**with** ⟨0  $\in$  SOV  $\varphi$ ⟩ **obtain** P I' **where** I': I = Inr P # I' **by** (cases I) (fastforce  
 split: sum.splits)+  
**with** I **have** wtlI: x  $\in$  enc (w, I') length I' = n **using** tl-enc[of Inr P I' w] **by**  
 auto  
**have** wf-interp-for-formula (w, I') (FEXISTS  $\varphi$ )



**using** *wf-interp-for-formula-FEXISTS*[*OF wf*[*folded wtl*(2)]]  
*wf-interp-for-formula-any-Inr*[*OF I*(2)[*unfolded I*']] ..  
**with** *wtl* **show**  $x \in ?L$  **unfolding** *lang-ENC-formula*[*OF wf*] **by** *blast*  
**qed**

**lemma** *lang<sub>WS1S</sub>-rexp-of-rexp-of'*:  
*wf-formula*  $n \varphi \implies \text{lang } n (\text{rexp-of } n \varphi) = \text{lang } n (\text{rexp-of}' n \varphi)$   
**unfolding** *rexp-of'-def* **proof** (*induction*  $\varphi$  *arbitrary*:  $n$ )  
**case** (*FNot*  $\varphi$ )  
**hence** *wf-formula*  $n \varphi$  **by** *simp*  
**with** *FNot.IH* **show** ?*case* **unfolding** *rexp-of.simps rexp-of-alt.simps lang.simps ENC-FNot* **by** *blast*  
**next**  
**case** (*FAnd*  $\varphi_1 \varphi_2$ )  
**hence** *wf1*: *wf-formula*  $n \varphi_1$  **and** *wf2*: *wf-formula*  $n \varphi_2$  **by** *force+*  
**from** *FAnd.IH*(1)[*OF wf1*] *FAnd.IH*(2)[*OF wf2*] **show** ?*case* **using** *ENC-FAnd*[*OF FAnd.prem*s]  
**unfolding** *rexp-of.simps rexp-of-alt.simps lang.simps rexp-of-list.simps* **by** *blast*  
**next**  
**case** (*FOR*  $\varphi_1 \varphi_2$ )  
**hence** *wf1*: *wf-formula*  $n \varphi_1$  **and** *wf2*: *wf-formula*  $n \varphi_2$  **by** *force+*  
**from** *FOR.IH*(1)[*OF wf1*] *FOR.IH*(2)[*OF wf2*] **show** ?*case* **using** *ENC-FOR*[*OF FOR.prem*s]  
**unfolding** *rexp-of.simps rexp-of-alt.simps lang.simps* **by** *blast*  
**next**  
**case** (*FExists*  $\varphi$ )  
**from** *FExists*(2) **have** *IH*: *lang*  $(n + 1) (\text{rexp-of } (n + 1) \varphi) =$   
*lang*  $(n + 1) (\text{Inter } (\text{rexp-of-alt } (n + 1) \varphi) (\text{ENC } (n + 1) (\text{FOV } \varphi)))$  **by**  
(*intro FExists.IH*) *auto*  
**have**  $\sigma$ : (*any, replicate*  $n \text{False}$ )  $\in (\text{set o } \sigma \Sigma) n$  **by** (*auto simp:*  $\sigma$ -*def set-n-lists image-iff*)  
**from** *FExists*(2) **have** *wf*: *wf*  $n (\text{Pr } (\text{rexp.Inter } (\text{rexp-of-alt } (n + 1) \varphi) (\text{ENC } (n + 1) (\text{FOV } \varphi))))$   
*wf*  $n (\text{Pr } (\text{rexp-of } (n + 1) \varphi))$  **by** (*fastforce simp: max-idx-vars intro!: wf-rexp-of wf-rexp-of-alt wf-rexp-ENC*[*OF finite-FOV*])+  
**note** *lang-quot* = *lang-samequot-exec*[*OF wf*(1)  $\sigma$ ] *lang-samequot-exec*[*OF wf*(2)  $\sigma$ ]  
**show** ?*case* **unfolding** *rexp-of.simps rexp-of-alt.simps lang.simps IH lang-quot Suc-eq-plus1*  
*ENC-FExists*[*OF FExists.prem*s, *unfolded Suc-eq-plus1*] **by** (*auto simp add: SAMEQUOT-def*)  
**next**  
**case** (*FEXISTS*  $\varphi$ )  
**from** *FEXISTS*(2) **have** *IH*: *lang*  $(n + 1) (\text{rexp-of } (n + 1) \varphi) =$   
*lang*  $(n + 1) (\text{Inter } (\text{rexp-of-alt } (n + 1) \varphi) (\text{ENC } (n + 1) (\text{FOV } \varphi)))$  **by**  
(*intro FEXISTS.IH*) *auto*  
**have**  $\sigma$ : (*any, replicate*  $n \text{False}$ )  $\in (\text{set o } \sigma \Sigma) n$  **by** (*auto simp:*  $\sigma$ -*def set-n-lists image-iff*)  
**from** *FEXISTS*(2) **have** *wf*: *wf*  $n (\text{Pr } (\text{rexp.Inter } (\text{rexp-of-alt } (n + 1) \varphi) (\text{ENC } (n + 1) (\text{FOV } \varphi))))$

$(n + 1) (FOV \varphi)))$   
*wf*  $n$  (*Pr* (*rexp-of*  $(n + 1) \varphi$ )) **by** (*fastforce simp: max-idx-vars intro: wf-rexp-of wf-rexp-of-alt wf-rexp-ENC[OF finite-FOV]*)+  
**note** *lang-quot* = *lang-samequot-exec*[*OF wf*(1)  $\sigma$ ] *lang-samequot-exec*[*OF wf*(2)  $\sigma$ ]  
**show** ?*case unfolding* *rexp-of.simps rexp-of-alt.simps lang.simps IH lang-quot Suc-eq-plus1*  
*ENC-FEXISTS*[*OF FEXISTS.prem*s, *unfolded Suc-eq-plus1*] **by** (*auto simp add: SAMEQUOT-def*)  
**qed** *auto*

**lemma** *SAMEQUTO-UN*[*simp*]: *SAMEQUOT*  $x (\bigcup y \in A. B y) = (\bigcup y \in A. \textit{SAMEQUOT } x (B y))$   
**unfolding** *SAMEQUOT-def* **by** *auto*

**lemma** *finite-positions-in-row*[*simp*]:  
 $n > 0 \implies \textit{finite} (\textit{positions-in-row} (x @- \textit{sconst} (any, \textit{replicate } n \textit{False})) 0)$   
**unfolding** *positions-in-row shift-snth* **by** *auto*

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

**lemma** *fin-cut-same-idem*: *fin-cut-same*  $x (\textit{fin-cut-same } x xs) = \textit{fin-cut-same } x xs$   
**by** (*induct xs*) *auto*

**lemma** *cut-same-sconst*: *cut-same*  $x (xs @- \textit{sconst } x) = \textit{fin-cut-same } x xs$   
**proof** (*induct xs rule: rev-induct*)  
**case** (*snoc y ys*)  
**then show** ?*case* **by** (*auto simp del: id-apply simp add: fin-cut-same-snoc sconst-collapse*)  
**qed** (*simp del: id-apply*)

**lemma** *length-cut-same*: *length* (*cut-same*  $x s$ ) = (*LEAST*  $n. \textit{sdrop } n s = \textit{sconst } x$ )  
**unfolding** *cut-same-def* **by** *simp*

**lemma** *enc-alt*: *wf-interp*  $w I \implies$   
 $x \in \textit{enc} (w, I) \iff x @- \textit{sconst} ((any, \textit{replicate} (\textit{length } I) \textit{False})) = \textit{stream-enc} (w, I)$   
**unfolding** *enc.simps*  
**by** (*force simp only: shift-append shift-replicate-sconst stream-enc-cut-same[symmetric] length-append length-replicate length-cut-same sdrop-shift drop-all diff-self-eq-0 shift.simps sdrop.simps*  
*dest: sym[of - stream-enc (w, I)]*  
*intro: shift-sconst-inj[rotated, of - (any, replicate (length I) False)] Least-le*  
*exI[of - length x - length (cut-same (any, replicate (length I) False) (stream-enc (w, I)))]*  
*le-add-diff-inverse[symmetric] )*

**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** (*coinduction arbitrary: xs ys*)

**case** *Eq-stream*

**then show** *?case*

**proof** (*cases xs ys rule: stream.exhaust[case-product stream.exhaust]*)

**case** (*SCons-SCons h1 t1 h2 t2*)

**with** *Eq-stream show ?thesis*

**by** (*cases snd h1 snd h2 rule: list.exhaust[case-product list.exhaust]*)

        (*auto simp:  $\pi$ -def split: prod.splits*)

**qed**

**qed**

**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) (map  $\pi$  x) @ replicate m (z n)  $\in$  enc (w, I)*

**assumes** *nonempty:  $\forall (-, x) \in \text{set } x. x \neq []$*

**shows**  $x \in \text{enc } (w, I')$

**proof** –

**have** [*simp*]:  $\pi (z (\text{Suc } n)) = z n$

**and** *z-def:  $\bigwedge n. z n = (\text{any}, \text{replicate } n \text{ False})$  unfolding  $\pi$ -def z-def by auto*

**have** *wf': wf-interp w I'* **by** (*simp add: wf I'-def z-def del: replicate-Suc*)

**note** *simps[ $\text{simp del}$ ] = stream-enc.simps*

**show** *?thesis unfolding enc-alt[OF wf']*

**proof** (*rule stream-stream-eqI*)

**from** *nonempty stream-smap-nats[of map ( $\lambda(-, y). \text{hd } y) x @- \text{sconst False}$ ]*

*smap-szip-fst*

**show**  $\text{smap } (\lambda(-, x). \text{hd } x) (x @- \text{sconst } (\text{any}, \text{replicate } (\text{length } I') \text{ False})) =$

$\text{smap } (\lambda(-, x). \text{hd } x) (\text{stream-enc } (w, I'))$

**by** (*auto simp add: stream-enc.simps I'-def z-def smap2-szip stream.map-comp*

*o-def split-def*

*positions-in-row shift-snth hd-conv-nth intro: smap-szip-fst[symmetric]*

*cong: stream.map-cong*)

**next**

**from** *wf have fin-cut-same (z n) (map  $\pi$  x) = cut-same (z n) (stream-enc (w, I))*

**using** *stream-enc-enc[OF - enc] by (auto simp add: cut-same-sconst z-def*

*n-def fin-cut-same-idem)*

**then obtain** *m' where  $\pi x: \text{map } \pi x = \text{cut-same } (z n) (\text{stream-enc } (w, I)) @$*

*replicate m' (z n)*

**by** (*auto dest!: fin-cut-sameE*)

**with** *wf show smap  $\pi (x @- \text{sconst } (\text{any}, \text{replicate } (\text{length } I') \text{ False})) =$*

$\text{smap } \pi (\text{stream-enc } (w, I'))$

**by** (*simp del: replicate-Suc add: n-def[symmetric] z-def[symmetric] I'-def  
stream-enc-cut-same[of I, symmetric, folded n-def z-def]*)  
**qed** (*insert nonempty, simp-all add: stream-enc.simps I'-def split-beta smap2-szip  
stream.set-map*)  
**qed**

**lemma** *pred-case-conv*:  $x - \text{Suc } 0 = (\text{case } x \text{ of } 0 \Rightarrow 0 \mid \text{Suc } m \Rightarrow m)$   
**by** (*cases x*) *auto*

**lemma** *in-pred-image-iff*:  $0 \notin X \implies (x \in (\lambda x. x - \text{Suc } 0) ' X) = (\text{Suc } x \in X)$   
**by** (*auto simp: pred-case-conv split: nat.splits*)

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

**fixes**  $X Z n$   
**defines**  $z \equiv (\text{any, replicate } n \text{ False})$   
**assumes**  $0 \notin X \ X \subseteq \{0 ..< n + 1\} \ Z \subseteq \text{lists } ((\text{set } o \ \sigma \ \Sigma) (n + 1))$   
**shows**  $\text{SAMEQUOT } z (\text{map } \pi ' (Z \cap \text{lang } (n + 1) (\text{ENC } (n + 1) X))) =$   
 $\text{SAMEQUOT } z (\text{map } \pi ' Z) \cap \text{lang } n (\text{ENC } n ((\lambda x. x - 1) ' X))$   
**proof** –  
**let**  $?Y = \{0 ..< n + 1\} - X$   
**let**  $?fX = (\lambda x. x - 1) ' X$   
**let**  $?fY = \{0 ..< n\} - (\lambda x. x - 1) ' X$   
**from** *assms* **have**  $*$ :  $(\lambda x. x - 1) ' X \subseteq \{0 ..< n\}$  **by** (*cases n*) *auto*  
**show** *?thesis*  
**proof** (*safe elim!: subsetD[OF SAMEQUOT-mono[OF subset-trans[OF image-Int-subset  
Int-lower1]]]*)  
**fix**  $w$  **assume**  $w \in \text{SAMEQUOT } z (\text{map } \pi ' (Z \cap \text{lang } (n + 1) (\text{ENC } (n + 1) X)))$   
**then** **have**  $w \in \text{SAMEQUOT } z (\text{map } \pi ' \text{lang } (n + 1) (\text{ENC } (n + 1) X))$   
**by** (*rule rev-subsetD[OF - SAMEQUOT-mono]*) *auto*  
**with** *assms*(2) **show**  $w \in \text{lang } n (\text{ENC } n ((\lambda x. x - 1) ' X))$   
**unfolding** *lang-ENC*[*OF assms*(3)] *subset-refl* *lang-ENC*[*OF \* subset-refl*]  
**by** (*auto simp: image-Union z-def length-Suc-conv simp del: enc.simps  
intro!: exI[of - enc (w, I) for w I, OF conjI[of - x ∈ A for x A]]*)  
(*fastforce simp: nth-Cons image-iff split: nat.splits sum.splits*)  
**next**  
**fix**  $w$  **assume**  $w \in \text{SAMEQUOT } z (\text{map } \pi ' Z) \ w \in \text{lang } n (\text{ENC } n ((\lambda x. x - 1) ' X))$   
**then** **show**  $w \in \text{SAMEQUOT } z (\text{map } \pi ' (Z \cap \text{lang } (n + 1) (\text{ENC } (n + 1) X)))$   
**unfolding** *z-def SAMEQUOT-def* **proof** (*safe, intro exI conjI*)  
**fix**  $m x$   
**assume**  $\pi x$ : *fin-cut-same* (*any, replicate n False*) (*map*  $\pi$   $x$ ) @  
 $\text{replicate } m (\text{any, replicate } n \text{ False}) \in \text{lang } n (\text{ENC } n ((\lambda x. x - 1) ' X))$   
**and**  $x \in Z$   
**show**  $\text{map } \pi \ x \in \text{map } \pi ' (Z \cap \text{lang } (n + 1) (\text{ENC } (n + 1) X))$   
**proof** (*intro imageI IntI*)  
**from** ( $x \in Z$ ) *assms*(4) **have**  $\forall (-, x) \in \text{set } x. x \neq []$  **by** (*auto simp:  $\sigma$ -def*)  
**with**  $\pi x$  *assms*(2) **show**  $x \in \text{lang } (n + 1) (\text{ENC } (n + 1) X)$

```

unfolding lang-ENC[OF assms( $\beta$ ) subset-refl] lang-ENC[OF * subset-refl]
proof (safe, intro UnionI[OF - project-enc-extend[rotated]] CollectI exI
conjI)
  fix w and I :: (nat + nat set) list
  assume Ball (set I) (case-sum ( $\lambda a$ . True) finite)
  then show Ball (set
    (Inr (positions-in-row (x @- sconst (any, replicate (Suc (length I))
False)) 0) #I))
    (case-sum ( $\lambda a$ . True) finite) by (auto simp del: replicate-Suc)
  qed (auto simp add: nth-Cons' Ball-def in-pred-image-iff)
  qed (rule (x  $\in$  Z))
  qed (rule refl)
qed
qed

```

**lemma** lang-ENC-split:

```

assumes finite X X = Y1  $\cup$  Y2 n = 0  $\vee$  ( $\forall p \in X$ . p < n)
shows lang n (ENC n X) = lang n (ENC n Y1)  $\cap$  lang n (ENC n Y2)
unfolding ENC-def lang-INTERSECT using assms lang-subset-lists[OF wf-rexp-valid-ENC,
of n] by auto

```

**lemma** map-project-ENC:

```

fixes n
assumes X  $\subseteq$  {0 ..< n + 1} Z  $\subseteq$  lists ((set o  $\sigma$   $\Sigma$ ) (n + 1))
defines z  $\equiv$  (any, replicate n False)
shows SAMEQUOT z (map  $\pi$  ' (Z  $\cap$  lang (n + 1) (ENC (n + 1) X))) =
  (if 0  $\in$  X
  then SAMEQUOT z (map  $\pi$  ' (Z  $\cap$  lang (n + 1) (ENC (n + 1) {0})))  $\cap$  lang
n (ENC n (( $\lambda x$ . x - 1) ' (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 (split if-splits, intro conjI impI)
  assume 0: 0  $\notin$  X
  from assms have fin: finite X finite (( $\lambda x$ . x - 1) ' X)
  by (auto elim: finite-subset intro!: finite-imageI[of X])
  from 0 show ?L = ?R2 using map-project-Int-ENC[OF 0 assms(1,2)]
  unfolding lists-image[symmetric]  $\pi$ - $\sigma$ 
    Int-absorb1[OF lang-subset-lists[OF wf-rexp-ENC[OF fin(1)], of n + 1]
    Int-absorb1[OF lang-subset-lists[OF wf-rexp-ENC[OF fin(2)], of n] unfolding
z-def
  by auto
next
  assume 0  $\in$  X
  hence 0: 0  $\notin$  X - {0} and X: X = {0}  $\cup$  (X - {0}) by auto
  from assms have fin: finite X
  by (auto elim: finite-subset intro!: finite-imageI[of X])
  have ?L = SAMEQUOT z (map  $\pi$  ' ((Z  $\cap$  lang (n + 1) (ENC (n + 1) {0})))
 $\cap$  lang (n + 1) (ENC (n + 1) (X - {0})))
  unfolding Int-assoc z-def using assms by (subst lang-ENC-split[OF fin X, of

```

$n + 1$ ]) *auto*  
**also have**  $\dots = ?R1$  **unfolding** *z-def*  
**using** *assms(1,2)* **by** (*intro map-project-Int-ENC*) *auto*  
**finally show**  $?L = ?R1$  .  
**qed**

**lemma** *lang<sub>M2L</sub>-rexp-of'-rexp-of''*:  
 $wf\text{-formula } n \ \varphi \implies lang \ n \ (rexp\text{-of}' \ n \ \varphi) = lang \ n \ (rexp\text{-of}'' \ n \ \varphi)$   
**unfolding** *rexp-of'-def rexp-of''-def*  
**proof** (*induction*  $\varphi$  *arbitrary: n*)  
**case** (*FNot*  $\varphi$ )  
**hence** *wf-formula*  $n \ \varphi$  **by** *simp*  
**with** *FNot.IH* **show**  $?case$  **unfolding** *rexp-of-alt.simps rexp-of-alt'.simps lang.simps*  
*ENC-FNot* **by** *blast*  
**next**  
**case** (*FAnd*  $\varphi_1 \ \varphi_2$ )  
**hence** *wf1: wf-formula*  $n \ \varphi_1$  **and** *wf2: wf-formula*  $n \ \varphi_2$  **by** *force+*  
**from** *FAnd.IH(1)[OF wf1] FAnd.IH(2)[OF wf2]* **show**  $?case$  **using** *ENC-FAnd[OF*  
*FAnd.premis]*  
**unfolding** *rexp-of-alt.simps rexp-of-alt'.simps lang.simps rexp-of-list.simps* **by**  
*blast*  
**next**  
**case** (*FOr*  $\varphi_1 \ \varphi_2$ )  
**hence** *wf1: wf-formula*  $n \ \varphi_1$  **and** *wf2: wf-formula*  $n \ \varphi_2$  **by** *force+*  
**from** *FOr.IH(1)[OF wf1] FOr.IH(2)[OF wf2]* **show**  $?case$  **using** *ENC-FOr[OF*  
*FOr.premis]*  
**unfolding** *rexp-of-alt.simps rexp-of-alt'.simps lang.simps rexp-of-list.simps* **by**  
*blast*  
**next**  
**case** (*FExists*  $\varphi$ )  
**hence** *wf: wf-formula*  $(n + 1) \ \varphi$  **and**  $0: 0 \in FOV \ \varphi$  **by** *auto*  
**then show**  $?case$   
**using** *max-idx-vars[of n + 1] \varphi wf-rexp-of-alt'[OF wf]*  
**unfolding** *rexp-of-alt.simps rexp-of-alt'.simps lang.simps Suc-eq-plus1*  
**proof** (*subst (1 2) lang-samequot-exec*)  
**show** *SAMEQUOT (any, replicate n False)*  
 $(lang \ n \ (Pr \ (Inter \ (rexp\text{-of}\text{-alt} \ (n + 1) \ \varphi) \ (ENC \ (n + 1) \ (FOV \ \varphi)))) \cap$   
 $lang \ n \ (ENC \ n \ (FOV \ (FExists \ \varphi))) =$   
 $SAMEQUOT \ (any, \ replicate \ n \ False)$   
 $(lang \ n \ (Pr \ (Inter \ (rexp\text{-of}\text{-alt}' \ (n + 1) \ \varphi) \ (ENC \ (n + 1) \ \{0\})))) \cap$   
 $lang \ n \ (ENC \ n \ (FOV \ (FExists \ \varphi)))$   
**using** *wf 0 max-idx-vars[of n + 1] \varphi wf-rexp-of-alt'[OF wf]*  
**unfolding** *lang.simps FExists.IH[OF wf, unfolded lang.simps]*  
**by** (*subst (1) map-project-ENC*) (*auto dest: subsetD[OF lang-subset-lists]*)  
**qed** (*auto simp add: wf-rexp-of-alt finite-FOV wf-rexp-ENC*)  
**next**  
**case** (*FEXISTS*  $\varphi$ )  
**hence** *wf: wf-formula*  $(n + 1) \ \varphi$  **and**  $0: 0 \notin FOV \ \varphi$  **by** *auto*  
**then show**  $?case$

```

using max-idx-vars[of n + 1  $\varphi$ ] wf-rexp-of-alt'[OF wf]
unfolding rexp-of-alt.simps rexp-of-alt'.simps lang.simps Suc-eq-plus1
proof (subst (1 2) lang-samequot-exec)
  show SAMEQUOT (any, replicate n False)
    (lang n (Pr (Inter (rexp-of-alt (n + 1)  $\varphi$ ) (ENC (n + 1) (FOV  $\varphi$ ))))  $\cap$ 
    lang n (ENC n (FOV (FEXISTS  $\varphi$ ))) =
  SAMEQUOT (any, replicate n False)
    (lang n (Pr (rexp-of-alt' (n + 1)  $\varphi$ )))  $\cap$ 
    lang n (ENC n (FOV (FEXISTS  $\varphi$ )))
  using wf 0 max-idx-vars[of n + 1  $\varphi$ ] wf-rexp-of-alt'[OF wf]
  unfolding lang.simps FEXISTS.IH[OF wf, unfolded lang.simps]
  by (subst (1) map-project-ENC) (auto dest: subsetD[OF lang-subset-lists])
  qed (auto simp add: wf-rexp-of-alt finite-FOV wf-rexp-ENC)
qed simp-all

```

```

theorem langWS1S-rexp-of': wf-formula n  $\varphi \implies$  langWS1S n  $\varphi =$  lang n (rexp-of'
n  $\varphi$ )
  unfolding langWS1S-rexp-of-rexp-of'[symmetric] by (rule langWS1S-rexp-of)

```

```

theorem langWS1S-rexp-of'': wf-formula n  $\varphi \implies$  langWS1S n  $\varphi =$  lang n (rexp-of''
n  $\varphi$ )
  unfolding langM2L-rexp-of'-rexp-of''[symmetric] by (rule langWS1S-rexp-of')

```

end

## 14 Normalization of WS1S Formulas

**fun** nNot where

```

  nNot (FNot  $\varphi$ ) =  $\varphi$ 
| nNot (FAnd  $\varphi_1$   $\varphi_2$ ) = FOr (nNot  $\varphi_1$ ) (nNot  $\varphi_2$ )
| nNot (FOr  $\varphi_1$   $\varphi_2$ ) = FAnd (nNot  $\varphi_1$ ) (nNot  $\varphi_2$ )
| nNot  $\varphi =$  FNot  $\varphi$ 

```

**primrec** norm where

```

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

```

**context** formula

**begin**

**lemma** satisfies-nNot[simp]:  $(w, I) \models$  nNot  $\varphi \iff (w, I) \models$  FNot  $\varphi$

**by** (*induct*  $\varphi$  *rule: nNot.induct*) *auto*

**lemma** *FOV-nNot[simp]*:  $FOV (nNot \varphi) = FOV (FNot \varphi)$   
**by** (*induct*  $\varphi$  *rule: nNot.induct*) *auto*

**lemma** *SOV-nNot[simp]*:  $SOV (nNot \varphi) = SOV (FNot \varphi)$   
**by** (*induct*  $\varphi$  *rule: nNot.induct*) *auto*

**lemma** *pre-wf-formula-nNot[simp]*:  $pre-wf-formula\ n (nNot \varphi) = pre-wf-formula\ n (FNot \varphi)$   
**by** (*induct*  $\varphi$  *rule: nNot.induct*) *auto*

**lemma** *FOV-norm[simp]*:  $FOV (norm \varphi) = FOV \varphi$   
**by** (*induct*  $\varphi$ ) *auto*

**lemma** *SOV-norm[simp]*:  $SOV (norm \varphi) = SOV \varphi$   
**by** (*induct*  $\varphi$ ) *auto*

**lemma** *pre-wf-formula-norm[simp]*:  $pre-wf-formula\ n (norm \varphi) = pre-wf-formula\ n \varphi$   
**by** (*induct*  $\varphi$  *arbitrary: n*) *auto*

**lemma** *satisfies-norm[simp]*:  $wI \models norm \varphi \longleftrightarrow wI \models \varphi$   
**by** (*induct*  $\varphi$  *arbitrary: wI*) *auto*

**lemma** *lang<sub>WS1S</sub>-norm[simp]*:  $lang_{WS1S}\ n (norm \varphi) = lang_{WS1S}\ n \varphi$   
**unfolding** *lang<sub>WS1S</sub>-def* **by** *auto*

**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*)  
**by** *unfold-locales (auto simp:  $\sigma$ -def  $\pi$ -def  $\varepsilon$ -def set-n-lists)*

**lemma** *enum-not-empty[simp]*:  $Enum.enum \neq []$  (**is**  $?enum \neq []$ )  
**proof** (*rule notI*)  
**assume**  $?enum = []$   
**hence** *set ?enum = {}* **by** *simp*



thus *False* **unfolding** *UNIV-enum*[*symmetric*] **by** *simp*  
**qed**

**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*) =  $\Omega$  *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-of-alt'* =  $\Phi$ .*rexp-of-alt'*  
**and** *rexp-of''* =  $\Phi$ .*rexp-of''*  
**and** *rexp-of'''* =  $\Phi$ .*rexp-of'''*  
**and** *valid-ENC* =  $\Phi$ .*valid-ENC*  
**and** *ENC* =  $\Phi$ .*ENC*  
**and** *dec-interp* =  $\Phi$ .*stream-dec*  
**and** *any* =  $\Phi$ .*any*  
**by** *unfold-locales* (*auto simp:  $\sigma$ -def  $\pi$ -def  $\Omega$ -def*)

**lemmas** *lang<sub>W<sub>S1S</sub></sub>*-rexp-of-norm = *trans*[*OF sym*[*OF  $\Phi$ .lang<sub>W<sub>S1S</sub></sub>*-norm]  $\Phi$ .*lang<sub>W<sub>S1S</sub></sub>*-rexp-of]  
**lemmas** *lang<sub>W<sub>S1S</sub></sub>*-rexp-of'-norm = *trans*[*OF sym*[*OF  $\Phi$ .lang<sub>W<sub>S1S</sub></sub>*-norm]  $\Phi$ .*lang<sub>W<sub>S1S</sub></sub>*-rexp-of']  
**lemmas** *lang<sub>W<sub>S1S</sub></sub>*-rexp-of''-norm = *trans*[*OF sym*[*OF  $\Phi$ .lang<sub>W<sub>S1S</sub></sub>*-norm]  $\Phi$ .*lang<sub>W<sub>S1S</sub></sub>*-rexp-of'']

**setup** (*Sign.map-naming* (*Name-Space.mandatory-path slow*))

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

$\lambda a \ r. \llbracket \mathfrak{D} \ \Sigma \ a \ r \rrbracket$  *final alphabet.wf* (*wf-atom*  $\Sigma$ ) *n* *pnorm lang*  $\Sigma$  *n* *n*

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

**defines**

*test* = *rexp-DA.test* (*final* :: 'a *atom rexp*  $\Rightarrow$  *bool*)

**and** *step* = *rexp-DA.step* ( $\sigma$   $\Sigma$ ) ( $\lambda a \ r. \llbracket \mathfrak{D} \ \Sigma \ a \ r \rrbracket$ ) *pnorm n*

**and** *closure* = *rexp-DA.closure* ( $\sigma$   $\Sigma$ ) ( $\lambda a \ r. \llbracket \mathfrak{D} \ \Sigma \ a \ r \rrbracket$ ) *final pnorm n*

**and** *check-equivRE* = *rexp-DA.check-equiv* ( $\sigma$   $\Sigma$ ) ( $\lambda x. \llbracket \text{pnorm } (inorm \ x) \rrbracket$ ) ( $\lambda a \ r. \llbracket \mathfrak{D} \ \Sigma \ a \ r \rrbracket$ ) *final pnorm n*

**and** *test-invariant* = *rexp-DA.test-invariant* (*final* :: 'a *atom rexp*  $\Rightarrow$  *bool*) ::

(('a  $\times$  *bool list*) *list*  $\times$  -) *list*  $\times$  -  $\Rightarrow$  *bool*

**and** *step-invariant* = *rexp-DA.step-invariant* ( $\sigma$   $\Sigma$ ) ( $\lambda a \ r. \llbracket \mathfrak{D} \ \Sigma \ a \ r \rrbracket$ ) *pnorm n*

**and** *closure-invariant* = *rexp-DA.closure-invariant* ( $\sigma$   $\Sigma$ ) ( $\lambda a \ r. \llbracket \mathfrak{D} \ \Sigma \ a \ r \rrbracket$ )

*final pnorm n*

**and** *counterexampleRE* = *rexp-DA.counterexample* ( $\sigma$   $\Sigma$ ) ( $\lambda x. \llbracket \text{pnorm } (inorm \ x) \rrbracket$ ) ( $\lambda a \ r. \llbracket \mathfrak{D} \ \Sigma \ a \ r \rrbracket$ ) *final pnorm n*

**and** *reachable* = *rexp-DA.reachable* ( $\sigma$   $\Sigma$ ) ( $\lambda x. \llbracket \text{pnorm } (inorm \ x) \rrbracket$ ) ( $\lambda a \ r. \llbracket \mathfrak{D} \ \Sigma \ a \ r \rrbracket$ ) *pnorm n*

**and** *automaton* = *rexp-DA.automaton* ( $\sigma$   $\Sigma$ ) ( $\lambda x. \llbracket \text{pnorm } (inorm \ x) \rrbracket$ ) ( $\lambda a \ r. \llbracket \mathfrak{D} \ \Sigma \ a \ r \rrbracket$ ) *pnorm n*

**by** *unfold-locales* (*auto simp only: comp-apply*)

*ACI-norm-wf ACI-norm-lang wf-inorm lang-inorm wf-pnorm lang-pnorm wf-lderi*

*lang-lderiv*  
*lang-final finite-fold-lderiv dest!: lang-subset-lists)*

**definition** *check-equiv where*

*check-equiv n φ ψ*  $\longleftrightarrow$  *wf-formula n (FOR φ ψ) ∧*  
*slow.check-equivRE Enum.enum n (rexp-of'' n (norm φ)) (rexp-of'' n (norm ψ))*

**definition** *counterexample where*

*counterexample n φ ψ =*  
*map-option (λw. dec-interp n (FOV (FOR φ ψ)) (w @- sconst (any, replicate*  
*n False)))*  
*(slow.counterexampleRE Enum.enum n (rexp-of'' n (norm φ)) (rexp-of'' n*  
*(norm ψ)))*

**lemma** *soundness: slow.check-equiv n φ ψ*  $\implies$   $\Phi$ .*lang<sub>WS1S</sub> n φ = Φ.lang<sub>WS1S</sub> n*  
*ψ*

**by** (*rule box-equals[OF slow.D.check-equiv-sound*  
*sym[OF trans[OF lang<sub>WS1S</sub>-rexp-of''-norm]] sym[OF trans[OF lang<sub>WS1S</sub>-rexp-of''-norm]]]*)  
*(auto simp: slow.check-equiv-def intro!: Φ.wf-rexp-of'')*

**lemma** *completeness:*

**assumes**  $\Phi$ .*lang<sub>WS1S</sub> n φ = Φ.lang<sub>WS1S</sub> n ψ wf-formula n (FOR φ ψ)*  
**shows** *slow.check-equiv n φ ψ*  
**using** *assms(2) unfolding slow.check-equiv-def*  
**by** (*intro conjI[OF assms(2) slow.D.check-equiv-complete,*  
*OF box-equals[OF assms(1) lang<sub>WS1S</sub>-rexp-of''-norm lang<sub>WS1S</sub>-rexp-of''-norm]]]*)  
*(auto intro!: Φ.wf-rexp-of'')*

**setup** *(Sign.map-naming Name-Space.parent-path)*

**setup** *(Sign.map-naming (Name-Space.mandatory-path fast))*

**global-interpretation** *D: rexp-DA-no-post σ Σ wf-atom Σ π lookup λx. pnorm*  
*(inorm x)*

*λa r. pnorm (⊔ Σ a r) final alphabet.wf (wf-atom Σ) n lang Σ n n*  
**for**  $\Sigma :: 'a :: \text{linorder list}$  **and**  $n :: \text{nat}$

**defines**

*test = rexp-DA.test (final :: 'a atom rexp  $\implies$  bool)*  
**and** *step = rexp-DA.step (σ Σ) (λa r. pnorm (⊔ Σ a r)) id n*  
**and** *closure = rexp-DA.closure (σ Σ) (λa r. pnorm (⊔ Σ a r)) final id n*  
**and** *check-equivRE = rexp-DA.check-equiv (σ Σ) (λx. pnorm (inorm x)) (λa r.*  
*pnorm (⊔ Σ a r)) final id n*  
**and** *test-invariant = rexp-DA.test-invariant (final :: 'a atom rexp  $\implies$  bool) ::*  
*(('a × bool list) list × -) list × -  $\implies$  bool*  
**and** *step-invariant = rexp-DA.step-invariant (σ Σ) (λa r. pnorm (⊔ Σ a r)) id*  
*n*  
**and** *closure-invariant = rexp-DA.closure-invariant (σ Σ) (λa r. pnorm (⊔ Σ a*  
*r)) final id n*  
**and** *counterexampleRE = rexp-DA.counterexample (σ Σ) (λx. pnorm (inorm x))*

$(\lambda a r. \text{pnorm } (\mathfrak{D} \Sigma a r)) \text{ final id } n$   
**and**  $\text{reachable} = \text{rexp-DA.reachable } (\sigma \Sigma) (\lambda x. \text{pnorm } (\text{inorm } x)) (\lambda a r. \text{pnorm } (\mathfrak{D} \Sigma a r)) \text{ id } n$   
**and**  $\text{automaton} = \text{rexp-DA.automaton } (\sigma \Sigma) (\lambda x. \text{pnorm } (\text{inorm } x)) (\lambda a r. \text{pnorm } (\mathfrak{D} \Sigma a r)) \text{ id } n$   
**by**  $\text{unfold-locales } (\text{auto simp only: comp-apply } \text{ACI-norm-wf ACI-norm-lang wf-inorm lang-inorm wf-pnorm lang-pnorm wf-ldderiv lang-ldderiv id-apply } \text{lang-final dest!: lang-subset-lists})$

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

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

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

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

**lemma** *soundness*:  $\text{fast.check-equiv } n \varphi \psi \implies \Phi.\text{lang}_{WS1S} n \varphi = \Phi.\text{lang}_{WS1S} n \psi$

**by**  $(\text{rule } \text{box-equals}[\text{OF fast.D.check-equiv-sound } \text{sym}[\text{OF trans}[\text{OF lang}_{WS1S}\text{-rexp-of''-norm}] \text{sym}[\text{OF trans}[\text{OF lang}_{WS1S}\text{-rexp-of''-norm}]]]])$   
 $(\text{auto simp: fast.check-equiv-def intro!: } \Phi.\text{wf-rexp-of''})$

**setup**  $\langle \text{Sign.map-naming Name-Space.parent-path} \rangle$

**setup**  $\langle \text{Sign.map-naming (Name-Space.mandatory-path dual)} \rangle$

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

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

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

**defines**

$\text{test} = \text{rexp-DA.test } (\text{final-dual } :: 'a \text{ atom rexp-dual } \Rightarrow \text{bool})$

**and**  $\text{step} = \text{rexp-DA.step } (\sigma \Sigma) (\lambda a r. \text{pnorm-dual } (\text{Co}\mathfrak{D} \Sigma a r)) \text{ id } n$

**and**  $\text{closure} = \text{rexp-DA.closure } (\sigma \Sigma) (\lambda a r. \text{pnorm-dual } (\text{Co}\mathfrak{D} \Sigma a r)) \text{ final-dual id } n$

**and**  $\text{check-equivRE} = \text{rexp-DA.check-equiv } (\sigma \Sigma) (\lambda x. \text{pnorm-dual } (\text{rexp-dual-of } (\text{inorm } x))) (\lambda a r. \text{pnorm-dual } (\text{Co}\mathfrak{D} \Sigma a r)) \text{ final-dual id } n$

**and**  $\text{test-invariant} = \text{rexp-DA.test-invariant } (\text{final-dual } :: 'a \text{ atom rexp-dual } \Rightarrow \text{bool}) ::$

$(('a \times \text{bool list}) \text{ list } \times -) \text{ list } \times - \Rightarrow \text{bool}$

**and**  $\text{step-invariant} = \text{rexp-DA.step-invariant } (\sigma \Sigma) (\lambda a r. \text{pnorm-dual } (\text{Co}\mathfrak{D} \Sigma a r)) \text{ id } n$

**and**  $\text{closure-invariant} = \text{rexp-DA.closure-invariant } (\sigma \Sigma) (\lambda a r. \text{pnorm-dual } (\text{Co}\mathfrak{D} \Sigma a r)) \text{ final-dual id } n$

**and**  $\text{counterexampleRE} = \text{rexp-DA.counterexample } (\sigma \Sigma) (\lambda x. \text{pnorm-dual } (\text{rexp-dual-of } (\text{inorm } x)))$

$(inorm\ x))$   $(\lambda a\ r.\ pnorm\text{-}dual\ (Co\mathfrak{D}\ \Sigma\ a\ r))\ final\text{-}dual\ id\ n$   
**and**  $reachable = rexp\text{-}DA.reachable\ (\sigma\ \Sigma)\ (\lambda x.\ pnorm\text{-}dual\ (rexp\text{-}dual\text{-}of\ (inorm\ x)))\ (\lambda a\ r.\ pnorm\text{-}dual\ (Co\mathfrak{D}\ \Sigma\ a\ r))\ id\ n$   
**and**  $automaton = rexp\text{-}DA.automaton\ (\sigma\ \Sigma)\ (\lambda x.\ pnorm\text{-}dual\ (rexp\text{-}dual\text{-}of\ (inorm\ x)))\ (\lambda a\ r.\ pnorm\text{-}dual\ (Co\mathfrak{D}\ \Sigma\ a\ r))\ id\ n$   
**by**  $unfold\text{-}locales\ (auto\ simp\ only:\ comp\text{-}apply\ id\text{-}apply$   
 $wf\text{-}inorm\ lang\text{-}inorm$   
 $wf\text{-}dual\text{-}pnorm\text{-}dual\ lang\text{-}dual\text{-}pnorm\text{-}dual$   
 $wf\text{-}dual\text{-}rexp\text{-}dual\text{-}of\ lang\text{-}dual\text{-}rexp\text{-}dual\text{-}of$   
 $wf\text{-}dual\text{-}lderiv\text{-}dual\ lang\text{-}dual\text{-}lderiv\text{-}dual$   
 $lang\text{-}dual\text{-}final\text{-}dual\ dest!\!: lang\text{-}dual\text{-}subset\text{-}lists)$

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

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

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

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

**lemma** *soundness*:  $dual.check\text{-}equiv\ n\ \varphi\ \psi \implies \Phi.lang_{WS1S}\ n\ \varphi = \Phi.lang_{WS1S}\ n\ \psi$

**by**  $(rule\ box\ equals[OF\ dual.D.check\text{-}equiv\text{-}sound$   
 $sym[OF\ trans[OF\ lang_{WS1S}\text{-}rexp\text{-}of''\text{-}norm]]\ sym[OF\ trans[OF\ lang_{WS1S}\text{-}rexp\text{-}of''\text{-}norm]]])$   
 $(auto\ simp:\ dual.check\text{-}equiv\text{-}def\ intro!\!: \Phi.wf\text{-}rexp\text{-}of'')$

**setup**  $\langle Sign.map\text{-}naming\ Name\text{-}Space.parent\text{-}path \rangle$

## 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.