MSO on Finite Words

Dmitriy Traytel and Tobias Nipkow

March 29, 2013

Contents

1 Regular Sets

theory *Regular-Set*
imports *Main*
begin

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

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

lemma [*code*]:
  A @@ *B* = (%(*xs*, *ys*). *xs* @ *ys*) ‘ (*A* × *B*)
  unfolding *conc-def* **by** *auto*

overloading *word-pow* == *compow* :: *nat* ⇒ *'a lang* ⇒ *'a lang*
begin
  primrec *word-pow* :: *nat* ⇒ *'a list* ⇒ *'a list* **where**
  word-pow 0 w = [] |
  word-pow (*Suc n*) *w* = *w* @ *word-pow n w*
end

overloading *lang-pow* == *compow* :: *nat* ⇒ *'a lang* ⇒ *'a lang*
begin

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

for code generation

definition *lang-pow* :: *nat* ⇒ *'a lang* ⇒ *'a lang* **where**
lang-pow-code-def [*code-abbrev*]: *lang-pow* = *compow*

lemma [*code*]:
lang-pow (*Suc n*) *A* = *A* @@ (*lang-pow n A*)
lang-pow 0 A = {[]}
by (*simp-all add*: *lang-pow-code-def*)

definition *word-pow* :: *nat* ⇒ *'a list* ⇒ *'a list* **where**
word-pow-code-def [*code-abbrev*]: *word-pow* = *compow*

lemma [*code*]:
word-pow 0 w = []
word-pow (*Suc n*) *w* = *w* @ *word-pow n w*
by (*simp-all add*: *word-pow-code-def*)

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

hide-const (**open**) *lang-pow word-pow*

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

1.1 Concatenation of Languages

lemma *concI*[*simp*,*intro*]: *u* : *A* ⟹ *v* : *B* ⟹ *u*@*v* : *A* @@ *B*
by (*auto simp add*: *conc-def*)

lemma *concE*[*elim*]:
assumes *w* ∈ *A* @@ *B*
obtains *u v* **where** *u* ∈ *A* *v* ∈ *B* *w* = *u*@*v*
using *assms* **by** (*auto simp*: *conc-def*)

lemma *conc-mono*: *A* ⊆ *C* ⟹ *B* ⊆ *D* ⟹ *A* @@ *B* ⊆ *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*: $(A\ @@\ B)\ @@\ C = A\ @@\ (B\ @@\ C)$
by (*auto elim*!: *concE*) (*simp only*: *append-assoc*[*symmetric*] *concI*)

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

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

lemma *hom-image-conc*: $[\![\bigwedge xs\ ys.\ f\ (xs\ @\ ys) = f\ xs\ @\ f\ ys]\!] \Longrightarrow f\ `\ (A\ @@\ B) = f\ `\ A\ @@\ f\ `\ B$
unfolding *conc-def* **by** (*auto simp*: *image-iff*) *metis*

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

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

1.2 Iteration of Languages

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

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

lemma *lang-pow-empty*: $\{\}\ \hat{}\hat{}\ n = (\mathit{if}\ n = 0\ \mathit{then}\ \{[]\}\ \mathit{else}\ \{\})$
by (*induct n*) *auto*

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

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

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

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

lemma *lang-pow-subset-lists*: $A \subseteq$ *lists* $S \Longrightarrow A$ ^^ $n \subseteq$ *lists* S
by(*induction* n)(*auto simp*: *conc-subset-lists*[*OF assms*])

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

lemma *star-if-lang-pow*[*simp*]: w : A ^^ $n \Longrightarrow w$: *star* A
by (*auto simp*: *star-def*)

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

lemma *star-if-lang*[*simp*]: **assumes** w : A **shows** w : *star* A
proof (*rule star-if-lang-pow*)
show w : A ^^ *1* **using** ‹w : A› **by** *simp*
qed

lemma *append-in-starI*[*simp*]:
assumes u : *star* A **and** v : *star* A **shows** u@v : *star* A
proof −
from ‹u : *star* A› **obtain** m **where** u : A ^^ m **by** (*auto simp*: *star-def*)
moreover
from ‹v : *star* A› **obtain** n **where** v : A ^^ n **by** (*auto simp*: *star-def*)
ultimately have u@v : A ^^ $(m+n)$ **by** (*simp add*: *lang-pow-add*)
thus *?thesis* **by** *simp*
qed

lemma *conc-star-star*: *star* A @@ *star* A = *star* A
by (*auto simp*: *conc-def*)

lemma *conc-star-comm*:
shows A @@ *star* A = *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 : *star* A
and P []
and *step*: !!u v. u : $A \Longrightarrow v$: *star* $A \Longrightarrow P$ $v \Longrightarrow P$ $(u@v)$
shows P w
proof −
{ fix n **have** w : A ^^ $n \Longrightarrow P$ w
by (*induct* n *arbitrary*: w) (*auto intro*: ‹P []› *step star-if-lang-pow*) **}**
with ‹w : *star* A› **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 us. ?R w us 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*: $x \in$ *star* A $x \neq []$
  shows $\exists\, a\ b.\ x = a\ @\ b \wedge a \neq [] \wedge a \in A \wedge b \in$ *star* A
  using *a* **by** (*induct rule*: *star-induct*) (*blast*)+

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

lemma *map-image-star*[*simp*]: *map f* ‘ *star A* = *star* (*map f* ‘ *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* ⇒ *'a lang* ⇒ *'a lang*
where *lQuot x A* = { *xs*. *x#xs* ∈ *A* }

definition *lQuots* :: *'a list* ⇒ *'a lang* ⇒ *'a lang*
where *lQuots xs A* = { *ys*. *xs* @ *ys* ∈ *A* }

abbreviation
  lQuotss :: *'a list* ⇒ *'a lang set* ⇒ *'a lang*
where
  lQuotss s As ≡ $\bigcup$ (*lQuots s*) ‘ *As*

lemma *lQuot-empty*[*simp*]: *lQuot a* {} = {}
  and *lQuot-epsilon*[*simp*]: *lQuot a* {[]} = {}
  and *lQuot-char*[*simp*]: *lQuot a* {[*b*]} = (*if a* = *b then* {[]} *else* {})
  and *lQuot-union*[*simp*]: *lQuot a* (*A* ∪ *B*) = *lQuot a A* ∪ *lQuot a B*
  and *lQuot-inter*[*simp*]: *lQuot a* (*A* ∩ *B*) = *lQuot a A* ∩ *lQuot a B*
  and *lQuot-compl*[*simp*]: *lQuot a* (−*A*) = − *lQuot a A*
  by (*auto simp*: *lQuot-def*)

lemma *lQuot-conc-subset*: *lQuot a A* @@ *B* ⊆ *lQuot a* (*A* @@ *B*) (**is** *?L* ⊆ *?R*)
proof
  fix *w* **assume** *w* ∈ *?L*
  then obtain *u v* **where** *w* = *u* @ *v a* # *u* ∈ *A v* ∈ *B*
    by (*auto simp*: *lQuot-def*)
  then have *a* # *w* ∈ *A* @@ *B*
    by (*auto intro*: *concI*[*of a* # *u*, *simplified*])
  thus *w* ∈ *?R* **by** (*auto simp*: *lQuot-def*)
qed

lemma *lQuot-conc* [*simp*]: *lQuot c* (*A* @@ *B*) = (*lQuot c A*) @@ *B* ∪ (*if* [] ∈ *A* *then lQuot c B else* {})

unfolding *lQuot-def conc-def*
by (*auto simp add*: *Cons-eq-append-conv*)

lemma *lQuot-star* [*simp*]: *lQuot c* (*star A*) = (*lQuot c A*) @@ *star A*
proof −
have *incl*: [] ∈ *A* ⟹ *lQuot c* (*star A*) ⊆ (*lQuot c A*) @@ *star A*
unfolding *lQuot-def conc-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* ∪ {[]})
by (*simp only*: *star-unfold-left*[*symmetric*])
also have ... = *lQuot c* (*A* @@ *star A*)
by (*simp only*: *lQuot-union*) (*simp*)
also have ... = (*lQuot c A*) @@ (*star A*) ∪ (*if* [] ∈ *A then lQuot c* (*star A*) *else* {})
by *simp*
also have ... = (*lQuot c A*) @@ *star A*
using *incl* **by** *auto*
finally show *lQuot c* (*star A*) = (*lQuot c A*) @@ *star A* **.**
qed

lemma *lQuot-diff*[*simp*]: *lQuot c* (*A* − *B*) = *lQuot c A* − *lQuot c B*
by(*auto simp add*: *lQuot-def*)

lemma *lQuot-lists*[*simp*]: *c* : *S* ⟹ *lQuot c* (*lists S*) = *lists S*
by(*auto simp add*: *lQuot-def*)

lemma *lQuots-simps* [*simp*]:
shows *lQuots* [] *A* = *A*
and *lQuots* (*c* # *s*) *A* = *lQuots s* (*lQuot c A*)
and *lQuots* (*s1* @ *s2*) *A* = *lQuots s2* (*lQuots s1 A*)
unfolding *lQuots-def lQuot-def* **by** *auto*

lemma *lQuots-append*[*iff*]: *v* ∈ *lQuots w A* ⟷ *w* @ *v* ∈ *A*
by (*induct w arbitrary*: *v A*) (*auto simp add*: *lQuot-def*)

1.4 Right-Quotients of Languages

definition *rQuot* :: *'a* ⇒ *'a lang* ⇒ *'a lang*
where *rQuot x A* = { *xs*. *xs* @ [*x*] ∈ *A* }

definition *rQuots* :: *'a list* ⇒ *'a lang* ⇒ *'a lang*
where *rQuots xs A* = { *ys*. *ys* @ *rev xs* ∈ *A* }

abbreviation
rQuotss :: *'a list* ⇒ *'a lang set* ⇒ *'a lang*

where
  $rQuotss\ s\ As \equiv \bigcup\ (rQuots\ s)\ `\ As$

lemma *rQuot-rev-lQuot*: $rQuot\ x\ A = rev\ `\ lQuot\ x\ (rev\ `\ A)$
  unfolding *rQuot-def lQuot-def* **by** (*auto simp*: *rev-swap*[*symmetric*])

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

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

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

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

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

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

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

lemma *image-rev-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-conc* [*simp*]: $rQuot\ c\ (A\ @@\ B) = A\ @@\ (rQuot\ c\ B) \cup (if\ [] \in B\ then\ rQuot\ c\ A\ else\ \{\})$
  unfolding *rQuot-rev-lQuot* **by** (*auto simp*: *image-image image-Un*)

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

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

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

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

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

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

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

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

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

and *biQuot-compl*[*simp*]: *biQuot a b* $(-A) = -$ *biQuot a b A*
by (*auto simp*: *biQuot-def*)

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

lemma *biQuots-simps* [*simp*]:
shows *biQuots* [] [] *A* = *A*
and *biQuots* (*a*#*as*) (*b*#*bs*) *A* = *biQuots as bs* (*biQuot a b A*)
and $[\![$*length s1* = *length t1*; *length s2* = *length t2*$]\!] \Longrightarrow$
biQuots (*s1* @ *s2*) (*t1* @ *t2*) *A* = *biQuots s2 t2* (*biQuots s1 t1 A*)
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* ^^ *Suc n*) @@ *X* $\cup$ ($\bigcup$*m*$\leq$*n*. (*A* ^^ *m*) @@ *B*)
proof (*induct n*)
case *0*
show *X* = (*A* ^^ *Suc 0*) @@ *X* $\cup$ ($\bigcup$*m*$\leq$*0*. (*A* ^^ *m*) @@ *B*)
using *eq* **by** *simp*
next
case (*Suc n*)
have *ih*: *X* = (*A* ^^ *Suc n*) @@ *X* $\cup$ ($\bigcup$*m*$\leq$*n*. (*A* ^^ *m*) @@ *B*) **by** *fact*
also have ... = (*A* ^^ *Suc n*) @@ (*A* @@ *X* $\cup$ *B*) $\cup$ ($\bigcup$*m*$\leq$*n*. (*A* ^^ *m*) @@ *B*)
using *eq* **by** *simp*
also have ... = (*A* ^^ *Suc* (*Suc n*)) @@ *X* $\cup$ ((*A* ^^ *Suc n*) @@ *B*) $\cup$ ($\bigcup$*m*$\leq$*n*. (*A* ^^ *m*) @@ *B*)
by (*simp add*: *conc-Un-distrib conc-assoc*[*symmetric*] *conc-pow-comm*)
also have ... = (*A* ^^ *Suc* (*Suc n*)) @@ *X* $\cup$ ($\bigcup$*m*$\leq$*Suc n*. (*A* ^^ *m*) @@ *B*)

by (*auto simp add*: *le-Suc-eq*)
finally show $X = (A\ \hat{}\hat{}\ Suc\ (Suc\ n))\ @@\ X \cup (\bigcup m{\le}Suc\ n.\ (A\ \hat{}\hat{}\ m)\ @@\ B)$
.
qed

lemma *Arden*:
assumes $[] \notin A$
shows $X = A\ @@\ X \cup B \longleftrightarrow X = star\ A\ @@\ B$
proof
assume *eq*: $X = A\ @@\ X \cup B$
{ **fix** w **assume** $w : X$
let $?n = size\ w$
from ‹$[] \notin A$› **have** $ALL\ u : A.\ length\ u \ge 1$
by (*metis Suc-eq-plus1 add-leD2 le-0-eq length-0-conv not-less-eq-eq*)
hence $ALL\ u : A\hat{}\hat{}(?n{+}1).\ length\ u \ge ?n{+}1$
by (*metis length-lang-pow-lb nat-mult-1*)
hence $ALL\ u : A\hat{}\hat{}(?n{+}1)@@X.\ length\ u \ge ?n{+}1$
by(*auto simp only*: *conc-def length-append*)
hence $w \notin A\hat{}\hat{}(?n{+}1)@@X$ **by** *auto*
hence $w : star\ A\ @@\ B$ **using** ‹$w : X$› **using** *arden-helper*[*OF eq*, **where** $n{=}?n$]
by (*auto simp add*: *star-def conc-UNION-distrib*)
} **moreover**
{ **fix** w **assume** $w : star\ A\ @@\ B$
hence $EX\ n.\ w : A\hat{}\hat{}n\ @@\ B$ **by**(*auto simp*: *conc-def star-def*)
hence $w : X$ **using** *arden-helper*[*OF eq*] **by** *blast*
} **ultimately show** $X = star\ A\ @@\ B$ **by** *blast*
next
assume *eq*: $X = star\ A\ @@\ B$
have $star\ A = A\ @@\ star\ A \cup \{[]\}$
by (*rule star-unfold-left*)
then have $star\ A\ @@\ B = (A\ @@\ star\ A \cup \{[]\})\ @@\ B$
by *metis*
also have $\ldots = (A\ @@\ star\ A)\ @@\ B \cup B$
unfolding *conc-Un-distrib* **by** *simp*
also have $\ldots = A\ @@\ (star\ A\ @@\ B) \cup B$
by (*simp only*: *conc-assoc*)
finally show $X = A\ @@\ X \cup B$
using *eq* **by** *blast*
qed

lemma *reversed-arden-helper*:
assumes *eq*: $X = X\ @@\ A \cup B$
shows $X = X\ @@\ (A\ \hat{}\hat{}\ Suc\ n) \cup (\bigcup m{\le}n.\ B\ @@\ (A\ \hat{}\hat{}\ m))$
proof (*induct n*)
case *0*
show $X = X\ @@\ (A\ \hat{}\hat{}\ Suc\ 0) \cup (\bigcup m{\le}0.\ B\ @@\ (A\ \hat{}\hat{}\ m))$
using *eq* **by** *simp*

```
next
  case (Suc n)
  have ih: X = X @@ (A ^^ Suc n) ∪ (⋃m≤n. B @@ (A ^^ m)) by fact
  also have ... = (X @@ A ∪ B) @@ (A ^^ Suc n) ∪ (⋃m≤n. B @@ (A ^^ m))
using eq by simp
  also have ... = X @@ (A ^^ Suc (Suc n)) ∪ (B @@ (A ^^ Suc n)) ∪ (⋃m≤n.
B @@ (A ^^ m))
    by (simp add: conc-Un-distrib conc-assoc)
  also have ... = X @@ (A ^^ Suc (Suc n)) ∪ (⋃m≤Suc n. B @@ (A ^^ m))
    by (auto simp add: le-Suc-eq)
  finally show X = X @@ (A ^^ Suc (Suc n)) ∪ (⋃m≤Suc n. B @@ (A ^^ m))
.
qed

theorem reversed-Arden:
  assumes nemp: [] ∉ A
  shows X = X @@ A ∪ B ⟷ X = B @@ star A
proof
 assume eq: X = X @@ A ∪ B
  { fix w assume w : X
    let ?n = size w
    from ‹[] ∉ A› have ALL u : A. length u ≥ 1
      by (metis Suc-eq-plus1 add-leD2 le-0-eq length-0-conv not-less-eq-eq)
    hence ALL u : A^^(?n+1). length u ≥ ?n+1
      by (metis length-lang-pow-lb nat-mult-1)
    hence ALL u : X @@ A^^(?n+1). length u ≥ ?n+1
      by(auto simp only: conc-def length-append)
    hence w ∉ X @@ A^^(?n+1) by auto
     hence w : B @@ star A using ‹w : X› using reversed-arden-helper[OF eq,
where n=?n]
      by (auto simp add: star-def conc-UNION-distrib)
  } moreover
  { fix w assume w : B @@ star A
    hence EX n. w : B @@ A^^n by (auto simp: conc-def star-def)
    hence w : X using reversed-arden-helper[OF eq] by blast
  } ultimately show X = B @@ star A by blast
next
  assume eq: X = B @@ star A
  have star A = {[]} ∪ star A @@ A
    unfolding conc-star-comm[symmetric]
    by(metis Un-commute star-unfold-left)
  then have B @@ star A = B @@ ({[]} ∪ star A @@ A)
    by metis
  also have ... = B ∪ B @@ (star A @@ A)
    unfolding conc-Un-distrib by simp
  also have ... = B ∪ (B @@ star A) @@ A
    by (simp only: conc-assoc)
  finally show X = X @@ A ∪ 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*: $A \subseteq listsN\ (n + 1)\ S \Longrightarrow tl\ `\ A \subseteq listsN\ n\ S$
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*: $A \subseteq lists\ (listsN\ (n + 1)\ S) \Longrightarrow map\ tl\ `\ A \subseteq lists\ (listsN\ n\ S)$
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

end

2 Π-Extended Regular Expressions

2.1 Syntax of regular expressions

datatype *'a rexp* =
  Zero |
  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)

Lifting constructors to lists

fun *rexp-of-list* **where**
  rexp-of-list OP N [] = *N*
| *rexp-of-list OP N* [*x*] = *x*
| *rexp-of-list OP N* (*x* # *xs*) = *OP x* (*rexp-of-list OP N xs*)

abbreviation *PLUS* ≡ *rexp-of-list Plus Zero*
abbreviation *TIMES* ≡ *rexp-of-list Times One*
abbreviation *INTERSECT* ≡ *rexp-of-list Inter* (*Not Zero*)

lemma *list-singleton-induct* [*case-names nil single cons*]:
  assumes *nil*: P []
  assumes *single*: $\bigwedge x.\ P\ [x]$
  assumes *cons*: $\bigwedge x\ y\ xs.\ P\ (y\ \#\ xs) \Longrightarrow P\ (x\ \#\ (y\ \#\ xs))$
  shows $P\ xs$
  using *assms*
proof (*induct xs*)
  case (*Cons x xs*) **thus** *?case* **by** (*cases xs*) *auto*
qed *simp*

Term ordering

instantiation *rexp* :: (*order*) {*order*}
begin

fun *le-rexp* :: ('*a*::*order*) *rexp* ⇒ ('*a*::*order*) *rexp* ⇒ *bool*
where
  le-rexp Zero - = True
| *le-rexp - Zero = False*
| *le-rexp One - = True*
| *le-rexp - One = False*
| *le-rexp* (*Atom a*) (*Atom b*) = (*a* <= *b*)
| *le-rexp* (*Atom -*) - = *True*
| *le-rexp* - (*Atom -*) = *False*
| *le-rexp* (*Star r*) (*Star s*) = *le-rexp r s*
| *le-rexp* (*Star -*) - = *True*
| *le-rexp* - (*Star -*) = *False*
| *le-rexp* (*Not r*) (*Not s*) = *le-rexp r s*
| *le-rexp* (*Not -*) - = *True*
| *le-rexp* - (*Not -*) = *False*
| *le-rexp* (*Plus r r'*) (*Plus s s'*) =
    (*if r = s then le-rexp r' s' else le-rexp r s*)
| *le-rexp* (*Plus - -*) - = *True*
| *le-rexp* - (*Plus - -*) = *False*
| *le-rexp* (*Times r r'*) (*Times s s'*) =
    (*if r = s then le-rexp r' s' else le-rexp r s*)
| *le-rexp* (*Times - -*) - = *True*
| *le-rexp* - (*Times - -*) = *False*
| *le-rexp* (*Inter r r'*) (*Inter s s'*) =
    (*if r = s then le-rexp r' s' else le-rexp r s*)
| *le-rexp* (*Inter - -*) - = *True*
| *le-rexp* - (*Inter - -*) = *False*
| *le-rexp* (*Pr r*) (*Pr s*) = *le-rexp r s*

definition *less-eq-rexp* **where** $r \le s \equiv$ *le-rexp r s*
definition *less-rexp* **where** $r < s \equiv$ *le-rexp r s* $\wedge\ r \neq s$

lemma *le-rexp-Zero*:
  le-rexp r Zero $\Longrightarrow r = $ *Zero*
  by (*induct r*) *auto*

lemma *le-rexp-refl*[*intro*]:
le-rexp r r
by (*induct r*) *auto*

lemma *le-rexp-antisym*:
$[\![$*le-rexp r s*; *le-rexp s r*$]\!] \Longrightarrow r = s$
by (*induct r s rule*: *le-rexp.induct*) (*auto dest*: *le-rexp-Zero*)

lemma *le-rexp-trans*:
$[\![$*le-rexp r s*; *le-rexp s t*$]\!] \Longrightarrow$ *le-rexp r t*
proof (*induct r s arbitrary*: *t rule*: *le-rexp.induct*)
fix *v t* **assume** *le-rexp* (*Atom v*) *t* **thus** *le-rexp One t* **by** (*cases t*) *auto*
next
fix *s1 s2 t* **assume** *le-rexp* (*Plus s1 s2*) *t* **thus** *le-rexp One t* **by** (*cases t*) *auto*
next
fix *s1 s2 t* **assume** *le-rexp* (*Times s1 s2*) *t* **thus** *le-rexp One t* **by** (*cases t*) *auto*
next
fix *s t* **assume** *le-rexp* (*Star s*) *t* **thus** *le-rexp One t* **by** (*cases t*) *auto*
next
fix *s t* **assume** *le-rexp* (*Not s*) *t* **thus** *le-rexp One t* **by** (*cases t*) *auto*
next
fix *s1 s2 t* **assume** *le-rexp* (*Inter s1 s2*) *t* **thus** *le-rexp One t* **by** (*cases t*) *auto*
next
fix *s t* **assume** *le-rexp* (*Pr s*) *t* **thus** *le-rexp One t* **by** (*cases t*) *auto*
next
fix *v u t* **assume** *le-rexp* (*Atom v*) (*Atom u*) *le-rexp* (*Atom u*) *t*
thus *le-rexp* (*Atom v*) *t* **by** (*cases t*) *auto*
next
fix *v s1 s2 t* **assume** *le-rexp* (*Plus s1 s2*) *t* **thus** *le-rexp* (*Atom v*) *t* **by** (*cases t*) *auto*
next
fix *v s1 s2 t* **assume** *le-rexp* (*Times s1 s2*) *t* **thus** *le-rexp* (*Atom v*) *t* **by** (*cases t*) *auto*
next
fix *v s t* **assume** *le-rexp* (*Star s*) *t* **thus** *le-rexp* (*Atom v*) *t* **by** (*cases t*) *auto*
next
fix *v s t* **assume** *le-rexp* (*Not s*) *t* **thus** *le-rexp* (*Atom v*) *t* **by** (*cases t*) *auto*
next
fix *v s1 s2 t* **assume** *le-rexp* (*Inter s1 s2*) *t* **thus** *le-rexp* (*Atom v*) *t* **by** (*cases t*) *auto*
next
fix *v s t* **assume** *le-rexp* (*Pr s*) *t* **thus** *le-rexp* (*Atom v*) *t* **by** (*cases t*) *auto*
next
fix *r s t*
assume *IH*: $\bigwedge t$. *local.le-rexp r s* $\Longrightarrow$ *local.le-rexp s t* $\Longrightarrow$ *local.le-rexp r t*
and *le-rexp* (*Star r*) (*Star s*) *le-rexp* (*Star s*) *t*
thus *local.le-rexp* (*Star r*) *t* **by** (*cases t*) *auto*
next

fix *r s1 s2 t* **assume** *le-rexp* (*Plus s1 s2*) *t* **thus** *le-rexp* (*Star r*) *t* **by** (*cases t*) *auto*
next
fix *r s1 s2 t* **assume** *le-rexp* (*Times s1 s2*) *t* **thus** *le-rexp* (*Star r*) *t* **by** (*cases t*) *auto*
next
fix *r s t* **assume** *le-rexp* (*Not s*) *t* **thus** *le-rexp* (*Star r*) *t* **by** (*cases t*) *auto*
next
fix *r s1 s2 t* **assume** *le-rexp* (*Inter s1 s2*) *t* **thus** *le-rexp* (*Star r*) *t* **by** (*cases t*) *auto*
next
fix *r s t* **assume** *le-rexp* (*Pr s*) *t* **thus** *le-rexp* (*Star r*) *t* **by** (*cases t*) *auto*
next
fix *r s t*
assume *IH*: $\bigwedge t.$ *local.le-rexp r s* $\Longrightarrow$ *local.le-rexp s t* $\Longrightarrow$ *local.le-rexp r t*
and *le-rexp* (*Not r*) (*Not s*) *le-rexp* (*Not s*) *t*
thus *local.le-rexp* (*Not r*) *t* **by** (*cases t*) *auto*
next
fix *r s1 s2 t* **assume** *le-rexp* (*Plus s1 s2*) *t* **thus** *le-rexp* (*Not r*) *t* **by** (*cases t*) *auto*
next
fix *r s1 s2 t* **assume** *le-rexp* (*Times s1 s2*) *t* **thus** *le-rexp* (*Not r*) *t* **by** (*cases t*) *auto*
next
fix *r s1 s2 t* **assume** *le-rexp* (*Inter s1 s2*) *t* **thus** *le-rexp* (*Not r*) *t* **by** (*cases t*) *auto*
next
fix *r s t* **assume** *le-rexp* (*Pr s*) *t* **thus** *le-rexp* (*Not r*) *t* **by** (*cases t*) *auto*
next
fix *r1 r2 s1 s2 t*
assume $\bigwedge t.$ *r1* = *s1* $\Longrightarrow$ *local.le-rexp r2 s2* $\Longrightarrow$ *local.le-rexp s2 t* $\Longrightarrow$ *local.le-rexp r2 t*
$\bigwedge t.$ *r1* $\neq$ *s1* $\Longrightarrow$ *local.le-rexp r1 s1* $\Longrightarrow$ *local.le-rexp s1 t* $\Longrightarrow$ *local.le-rexp r1 t*
le-rexp (*Plus r1 r2*) (*Plus s1 s2*) *le-rexp* (*Plus s1 s2*) *t*
thus *le-rexp* (*Plus r1 r2*) *t* **by** (*cases t*) (*auto split*: *split-if-asm intro*: *le-rexp-antisym*)
next
fix *r1 r2 s1 s2 t* **assume** *le-rexp* (*Times s1 s2*) *t* **thus** *le-rexp* (*Plus r1 r2*) *t* **by** (*cases t*) *auto*
next
fix *r1 r2 s1 s2 t* **assume** *le-rexp* (*Inter s1 s2*) *t* **thus** *le-rexp* (*Plus r1 r2*) *t* **by** (*cases t*) *auto*
next
fix *r1 r2 s t* **assume** *le-rexp* (*Pr s*) *t* **thus** *le-rexp* (*Plus r1 r2*) *t* **by** (*cases t*) *auto*
next
fix *r1 r2 s1 s2 t*
assume $\bigwedge t.$ *r1* = *s1* $\Longrightarrow$ *local.le-rexp r2 s2* $\Longrightarrow$ *local.le-rexp s2 t* $\Longrightarrow$ *local.le-rexp r2 t*

$\bigwedge t.\ r1 \neq s1 \Longrightarrow local.le\text{-}rexp\ r1\ s1 \Longrightarrow local.le\text{-}rexp\ s1\ t \Longrightarrow local.le\text{-}rexp$
$r1\ t$
$le\text{-}rexp\ (Times\ r1\ r2)\ (Times\ s1\ s2)\ le\text{-}rexp\ (Times\ s1\ s2)\ t$
thus *le-rexp* (*Times r1 r2*) *t* **by** (*cases t*) (*auto split*: *split-if-asm intro*: *le-rexp-antisym*)
next
fix *r1 r2 s1 s2 t* **assume** *le-rexp* (*Inter s1 s2*) *t* **thus** *le-rexp* (*Times r1 r2*) *t* **by**
(*cases t*) *auto*
next
fix *r1 r2 s t* **assume** *le-rexp* (*Pr s*) *t* **thus** *le-rexp* (*Times r1 r2*) *t* **by** (*cases t*)
auto
next
fix *r1 r2 s1 s2 t*
assume $\bigwedge t.\ r1 = s1 \Longrightarrow local.le\text{-}rexp\ r2\ s2 \Longrightarrow local.le\text{-}rexp\ s2\ t \Longrightarrow local.le\text{-}rexp$
$r2\ t$
$\bigwedge t.\ r1 \neq s1 \Longrightarrow local.le\text{-}rexp\ r1\ s1 \Longrightarrow local.le\text{-}rexp\ s1\ t \Longrightarrow local.le\text{-}rexp$
$r1\ t$
$le\text{-}rexp\ (Inter\ r1\ r2)\ (Inter\ s1\ s2)\ le\text{-}rexp\ (Inter\ s1\ s2)\ t$
thus *le-rexp* (*Inter r1 r2*) *t* **by** (*cases t*) (*auto split*: *split-if-asm intro*: *le-rexp-antisym*)
next
fix *r1 r2 s t* **assume** *le-rexp* (*Pr s*) *t* **thus** *le-rexp* (*Inter r1 r2*) *t* **by** (*cases t*)
auto
next
fix *r s t*
assume *IH*: $\bigwedge t.\ local.le\text{-}rexp\ r\ s \Longrightarrow local.le\text{-}rexp\ s\ t \Longrightarrow local.le\text{-}rexp\ r\ t$
and *le-rexp* (*Pr r*) (*Pr s*) *le-rexp* (*Pr s*) *t*
thus *local.le-rexp* (*Pr r*) *t* **by** (*cases t*) *auto*
qed *auto*

instance proof
qed (*auto simp add*: *less-eq-rexp-def less-rexp-def intro*: *le-rexp-antisym le-rexp-trans*)

end

instantiation *rexp* :: (*linorder*) {*linorder*}
begin

lemma *le-rexp-total*:
$le\text{-}rexp\ (r :: 'a :: linorder\ rexp)\ s \lor le\text{-}rexp\ s\ r$
by (*induct r s rule*: *le-rexp.induct*) *auto*

instance proof
qed (*unfold less-eq-rexp-def less-rexp-def*, *rule le-rexp-total*)

end

2.2 ACI normalization

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

| *toplevel-summands r* = $\{r\}$

abbreviation *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* ($\ll$-$\gg$) **where**
$\ll$*Zero*$\gg$ = *Zero*
| $\ll$*One*$\gg$ = *One*
| $\ll$*Atom a*$\gg$ = *Atom a*
| $\ll$*Plus r s*$\gg$ = *flatten PLUS* (*toplevel-summands* (*Plus* $\ll$*r*$\gg$ $\ll$*s*$\gg$))
| $\ll$*Times r s*$\gg$ = *Times* $\ll$*r*$\gg$ $\ll$*s*$\gg$
| $\ll$*Star r*$\gg$ = *Star* $\ll$*r*$\gg$
| $\ll$*Not r*$\gg$ = *Not* $\ll$*r*$\gg$
| $\ll$*Inter r s*$\gg$ = *Inter* $\ll$*r*$\gg$ $\ll$*s*$\gg$
| $\ll$*Pr r*$\gg$ = *Pr* $\ll$*r*$\gg$

lemma *Plus-toplevel-summands*:
Plus r s $\in$ *toplevel-summands t* $\Longrightarrow$ *False*
by (*induct t*) *auto*

lemma *toplevel-summands-not-Plus*[*simp*]:
($\forall$ *r s. x* $\neq$ *Plus r s*) $\Longrightarrow$ *toplevel-summands x* = $\{x\}$
by (*induct x*) *auto*

lemma *toplevel-summands-PLUS-strong*:
$[\![$*xs* $\neq$ []; *list-all* ($\lambda x.$ $\neg$($\exists$ *r s. x* = *Plus r s*)) *xs*$]\!]$ $\Longrightarrow$ *toplevel-summands* (*PLUS xs*) = *set xs*
by (*induct xs rule*: *list-singleton-induct*) *auto*

lemma *toplevel-summands-flatten*:
$[\![$*X* $\neq \{\}$; *finite X*; $\forall$ *x* $\in$ *X*. $\neg$($\exists$ *r s. x* = *Plus r s*)$]\!]$ $\Longrightarrow$ *toplevel-summands* (*flatten PLUS X*) = *X*
using *toplevel-summands-PLUS-strong*[*of sorted-list-of-set X*] *sorted-list-of-set*[*of X*]
unfolding *list-all-iff* **by** *fastforce*

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

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

‘ *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*:
toplevel-summands (*flatten PLUS* (*ACI-norm* ‘ *toplevel-summands r* ∪ *ACI-norm* ‘ *toplevel-summands s*)) =
ACI-norm ‘ *toplevel-summands r* ∪ *ACI-norm* ‘ *toplevel-summands s*
by (*intro toplevel-summands-flatten*) (*auto dest*!: *ACI-norm-Plus*[*OF sym*] *intro*: *Plus-toplevel-summands*)

lemma *toplevel-summands-ACI-norm*:
toplevel-summands ≪*r*≫ = *ACI-norm* ‘ *toplevel-summands r*
by (*induct r*) (*auto simp*: *toplevel-summands-flatten-ACI-norm-image-Union*)

lemma *ACI-norm-flatten*:
≪*r*≫ = *flatten PLUS* (*ACI-norm* ‘ *toplevel-summands r*)
by (*induct r*) (*auto simp*: *image-Un toplevel-summands-flatten-ACI-norm-image*)

theorem *ACI-norm-idem*[*simp*]:
≪≪*r*≫≫ = ≪*r*≫
proof (*induct r*)
case (*Plus r s*)
have ≪≪*Plus r s*≫≫ = ≪*flatten PLUS* (*toplevel-summands* ≪*r*≫ ∪ *toplevel-summands* ≪*s*≫)≫
(**is** - = ≪*flatten PLUS ?U*≫) **by** *simp*
also have . . . = *flatten PLUS* (*ACI-norm* ‘ *toplevel-summands* (*flatten PLUS ?U*))
unfolding *ACI-norm-flatten* **..**
also have *toplevel-summands* (*flatten PLUS ?U*) = *?U*
by (*intro toplevel-summands-flatten*) (*auto intro*: *Plus-toplevel-summands*)
also have *flatten PLUS* (*ACI-norm* ‘ *?U*) = *flatten PLUS* (*toplevel-summands* ≪*r*≫ ∪ *toplevel-summands* ≪*s*≫)
unfolding *image-Un toplevel-summands-ACI-norm*[*symmetric*] *Plus* **..**
finally show *?case* **by** *simp*
qed *auto*

fun *ACI-nPlus* :: *'a*::*linorder rexp* ⇒ *'a rexp* ⇒ *'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**

$ACI\text{-}norm\text{-}alt\ Zero = Zero$
$|\ ACI\text{-}norm\text{-}alt\ One = One$
$|\ ACI\text{-}norm\text{-}alt\ (Atom\ a) = Atom\ a$
$|\ ACI\text{-}norm\text{-}alt\ (Plus\ r\ s) = ACI\text{-}nPlus\ (ACI\text{-}norm\text{-}alt\ r)\ (ACI\text{-}norm\text{-}alt\ s)$
$|\ ACI\text{-}norm\text{-}alt\ (Times\ r\ s) = Times\ (ACI\text{-}norm\text{-}alt\ r)\ (ACI\text{-}norm\text{-}alt\ s)$
$|\ ACI\text{-}norm\text{-}alt\ (Star\ r) = Star\ (ACI\text{-}norm\text{-}alt\ r)$
$|\ ACI\text{-}norm\text{-}alt\ (Not\ r) = Not\ (ACI\text{-}norm\text{-}alt\ r)$
$|\ ACI\text{-}norm\text{-}alt\ (Inter\ r\ s) = Inter\ (ACI\text{-}norm\text{-}alt\ r)\ (ACI\text{-}norm\text{-}alt\ s)$
$|\ ACI\text{-}norm\text{-}alt\ (Pr\ r) = Pr\ (ACI\text{-}norm\text{-}alt\ r)$

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*: $\ll ACI\text{-}norm\text{-}alt\ r\gg\ = \ll r\gg$
proof (*induction r*)
case (*Plus r s*) **show** *?case*
by (*auto simp*: *ACI-norm-flatten image-Un toplevel-summands-ACI-nPlus*)
(*metis Plus.IH toplevel-summands-ACI-norm*)
qed *auto*

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

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

proof (*induct xs1 arbitrary*: *xs2 rule*: *list-singleton-induct*)
  case (*single x1*)
  thus *?case*
   apply (*auto intro*!: *trans*[*OF ACI-nPlus-singleton-PLUS*] *simp del*: *sorted-list-of-set-insert*)
    apply *fastforce*
    apply (*simp only*: *insert-absorb*)
    apply (*metis List.finite-set finite-sorted-distinct-unique sorted-list-of-set*)
    apply (*rule arg-cong*[*of - - PLUS*])
   apply (*metis List.set.simps*(*2*) *distinct.simps*(*2*) *remdups-id-iff-distinct sort-key-simps*(*2*) *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 *fastforce*
  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*:
  $[\![X1 \neq \{\}; X2 \neq \{\}; \mathit{finite}\ X1; \mathit{finite}\ X2; \forall x \in X1 \cup X2.\ \neg(\exists r\ s.\ x = \mathit{Plus}\ r\ s)]\!] \Longrightarrow$
  $\mathit{ACI\text{-}nPlus}\ (\mathit{flatten}\ \mathit{PLUS}\ X1)\ (\mathit{flatten}\ \mathit{PLUS}\ X2) = \mathit{flatten}\ \mathit{PLUS}\ (X1 \cup X2)$
  by (*rule trans*[*OF ACI-nPlus-PLUS*]) (*auto*, (*metis List.set.simps*(*1*) *all-not-in-conv sorted-list-of-set*)+)

lemma *ACI-nPlus-ACI-norm*[*simp*]: $\mathit{ACI\text{-}nPlus} \ll r\gg \ll s\gg = \ll \mathit{Plus}\ r\ s\gg$
  by (*auto simp*: *image-Un Un-assoc ACI-norm-flatten intro*!: *trans*[*OF ACI-nPlus-flatten-PLUS*])
   (*metis ACI-norm-Plus Plus-toplevel-summands ACI-norm-flatten*)+

lemma *ACI-norm-alt*:
  $\mathit{ACI\text{-}norm\text{-}alt}\ r = \ll r\gg$
  by (*induct r*) *auto*

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

2.3 Finality

primrec *final* :: *'a rexp* ⇒ *bool*
where
  final Zero = *False*
| *final One* = *True*
| *final* (*Atom* -) = *False*
| *final* (*Plus r s*) = (*final r* ∨ *final s*)
| *final* (*Times r s*) = (*final r* ∧ *final s*)

| *final* (*Star* -) = *True*
| *final* (*Not r*) = (~ *final r*)
| *final* (*Inter r1 r2*) = (*final r1* ∧ *final r2*)
| *final* (*Pr r*) = *final r*

lemma *toplevel-summands-final*:
final s = (∃ *r*∈*toplevel-summands s*. *final r*)
by (*induct s*) *auto*

lemma *final-PLUS*:
final (*PLUS xs*) = (∃ *r* ∈ *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 Σ :: *nat* ⇒ *'a* :: *linorder set* (Σ -)
begin

primrec *wf* :: *nat* ⇒ *'a rexp* ⇒ *bool*
where
wf n Zero = *True* |
wf n One = *True* |
wf n (*Atom a*) = (*a* ∈ Σ *n*) |
wf n (*Plus r s*) = (*wf n r* ∧ *wf n s*) |
wf n (*Times r s*) = (*wf n r* ∧ *wf n s*) |
wf n (*Star r*) = *wf n r* |
wf n (*Not r*) = *wf n r* |
wf n (*Inter r s*) = (*wf n r* ∧ *wf n s*) |
wf n (*Pr r*) = *wf* (*n* + *1*) *r*

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

lemma *wf-word-snoc*[*simp*]: *wf-word n* (*ws* @ [*w*]) = ((*w* ∈ Σ *n*) ∧ *wf-word n ws*)
by (*induct ws*) *auto*

lemma *wf-word-append*[*simp*]: *wf-word n* (*ws* @ *vs*) = (*wf-word n ws* ∧ *wf-word n vs*)
by (*induct ws arbitrary*: *vs*) *auto*

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

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

theorem *ACI-norm-wf*[*simp*]:
  wf n «*r*» = *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*:
  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 $\Longrightarrow$ *wf n* (*flatten INTERSECT X*) = $(\forall\, r \in X.$ *wf n r*)
 using *wf-INTERSECT*[*of n sorted-list-of-set X*] *sorted-list-of-set*[*of X*] **by** *fastforce*

end

2.5 Language

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

primrec *lang* :: *nat* $\Rightarrow$ *'a rexp* => *'a lang* **where**
lang n Zero = {} |
lang n One = {[]} |
lang n (*Atom a*) = {[*a*]} |
lang n (*Plus r s*) = (*lang n r*) $\cup$ (*lang n s*) |
lang n (*Times r s*) = *conc* (*lang n r*) (*lang n s*) |
lang n (*Star r*) = *star* (*lang n r*) |
lang n (*Not r*) = *lists* $(\Sigma\ n)$ − *lang n r* |

lang n (*Inter r s*) = (*lang n r* ∩ *lang n s*) |
lang n (*Pr r*) = *map project* ‘ *lang* (*n* + *1*) *r*

lemma *wf-word-map-project*[*simp*]: *wf-word* (*Suc n*) *ws* ⟹ *wf-word n* (*map project ws*)
by (*induct ws arbitrary*: *n*) (*auto intro*: *project*)

lemma *wf-lang-wf-word*: *wf n r* ⟹ ∀ *w* ∈ *lang n r*. *wf-word n w*
by (*induct r arbitrary*: *n*) (*auto elim*: *set-rev-mp*[*OF - conc-mono*] *star-induct intro*: *iffD2*[*OF wf-word*])

lemma *lang-subset-lists*: *wf n r* ⟹ *lang n r* ⊆ *lists* (Σ *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 ∈ *toplevel-summands s* ⟹ *lang n r* ⊆ *lang n s*
by (*induct s*) *auto*

lemma *toplevel-summands-lang-UN*:
lang n s = (⋃ *r*∈*toplevel-summands s*. *lang n r*)
by (*induct s*) *auto*

lemma *toplevel-summands-in-lang*:
w ∈ *lang n s* = (∃ *r*∈*toplevel-summands s*. *w* ∈ *lang n r*)
by (*induct s*) *auto*

lemma *lang-PLUS*:
lang n (*PLUS xs*) = (⋃ *r* ∈ *set xs*. *lang n r*)
by (*induct xs rule*: *list-singleton-induct*) *auto*

lemma *lang-PLUS-map*[*simp*]:
lang n (*PLUS* (*map f xs*)) = (⋃ *a*∈*set xs*. *lang n* (*f a*))
by (*induct xs rule*: *list-singleton-induct*) *auto*

lemma *lang-flatten-PLUS*[*simp*]:
finite X ⟹ *lang n* (*flatten PLUS X*) = (⋃ *r* ∈ *X*. *lang n r*)
using *lang-PLUS*[*of n sorted-list-of-set X*] *sorted-list-of-set*[*of X*] **by** *fastforce*

theorem *ACI-norm-lang*[*simp*]:
lang n «*r*» = *lang n r*
proof (*induct r arbitrary*: *n*)
case (*Plus r1 r2*)
moreover
from *Plus*[*symmetric*] **have** *lang n* (*Plus r1 r2*) ⊆ *lang n* «*Plus r1 r2*»
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*: *final r* = ([] ∈ *lang n r*)
using *concI*[*of* [] - []] **by** (*induct r arbitrary*: *n*) *auto*

lemma *in-lang-INTERSECT*:
wf-word n w ⟹ *w* ∈ *lang n* (*INTERSECT xs*) = (∀ *r* ∈ *set xs*. *w* ∈ *lang n r*)
by (*induct xs rule*: *list-singleton-induct*) (*auto simp*: *wf-word*)

lemma *lang-flatten-INTERSECT*[*simp*]:
assumes *finite X X* ≠ {} ∀ *r*∈*X*. *wf n r*
shows *w* ∈ *lang n* (*flatten INTERSECT X*) = (∀ *r* ∈ *X*. *w* ∈ *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

end

3 Derivatives of Π-Extended Regular Expressions

locale *embed* = *project* Σ *project*
for Σ :: *nat* ⇒ *'a* :: *linorder set*
and *project* :: *'a* ⇒ *'a* +
fixes *embed* :: *'a* ⇒ *'a list*
assumes *embed*: ⋀*a*. *a* ∈ Σ *n* ⟹ *b* ∈ *set* (*embed a*) = (*b* ∈ Σ (*Suc n*) ∧ *project b* = *a*)
begin

3.1 Syntactic Derivatives

primrec *lderiv* :: *'a* ⇒ *'a rexp* ⇒ *'a rexp* **where**
lderiv - Zero = *Zero*
| *lderiv - One* = *Zero*
| *lderiv as* (*Atom bs*) = (*if as* = *bs then One else Zero*)
| *lderiv as* (*Plus r s*) = *Plus* (*lderiv as r*) (*lderiv as s*)

| *lderiv as* (*Times r s*) =
  (*let r's* = *Times* (*lderiv as r*) *s*
   in if final r then Plus r's (*lderiv as s*) *else r's*)
| *lderiv as* (*Star r*) = *Times* (*lderiv as r*) (*Star r*)
| *lderiv as* (*Not r*) = *Not* (*lderiv as r*)
| *lderiv as* (*Inter r s*) = *Inter* (*lderiv as r*) (*lderiv as s*)
| *lderiv as* (*Pr r*) = *Pr* (*PLUS* (*map* (λ*a*. *lderiv a r*) (*embed as*)))

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

3.2 Finiteness of ACI-Equivalent Derivatives

lemma *toplevel-summands-lderiv*:
 toplevel-summands (*lderiv as r*) = (⋃*s*∈*toplevel-summands r*. *toplevel-summands* (*lderiv as s*))
 by (*induct r*) (*auto simp*: *Let-def*)

lemma *lderivs-Zero*[*simp*]: *lderivs xs Zero* = *Zero*
 by (*induct xs*) *auto*

lemma *lderivs-One*: *lderivs xs One* ∈ {*Zero*, *One*}
 by (*induct xs*) *auto*

lemma *lderivs-Atom*: *lderivs xs* (*Atom as*) ∈ {*Zero*, *One*, *Atom as*}
proof (*induct xs*)
 case (*Cons x xs*) **thus** *?case* **by** (*auto intro*: *insertE*[*OF lderivs-One*])
qed *simp*

lemma *lderivs-Plus*: *lderivs xs* (*Plus r s*) = *Plus* (*lderivs xs r*) (*lderivs xs s*)
 by (*induct xs arbitrary*: *r s*) *auto*

lemma *lderivs-PLUS*: *lderivs xs* (*PLUS ys*) = *PLUS* (*map* (*lderivs xs*) *ys*)
 by (*induct ys rule*: *list-singleton-induct*) (*auto simp*: *lderivs-Plus*)

lemma *toplevel-summands-lderivs-Times*: *toplevel-summands* (*lderivs xs* (*Times r s*)) ⊆
 {*Times* (*lderivs xs r*) *s*} ∪
 {*r'*. ∃*ys zs*. *r'* ∈ *toplevel-summands* (*lderivs ys s*) ∧ *ys* ≠ [] ∧ *zs* @ *ys* = *xs*}
proof (*induct xs arbitrary*: *r s*)
 case (*Cons x xs*)
 thus *?case* **by** (*auto simp*: *Let-def lderivs-Plus*) (*fastforce intro*: *exI*[*of - x*#*xs*])+
qed *simp*

lemma *toplevel-summands-lderivs-Star-nonempty*:
 xs ≠ [] ⟹ *toplevel-summands* (*lderivs xs* (*Star r*)) ⊆
  {*Times* (*lderivs ys r*) (*Star r*) | *ys*. ∃*zs*. *ys* ≠ [] ∧ *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-lderivs-Times*] *intro*: *exI*[*of* - *y#ys*])
(*auto elim*!: *impE dest*!: *meta-spec subsetD*)
qed

lemma *toplevel-summands-lderivs-Star*:
toplevel-summands (*lderivs xs* (*Star r*)) $\subseteq$
$\{Star\ r\} \cup \{Times\ (lderivs\ ys\ r)\ (Star\ r) \mid ys.\ \exists zs.\ ys \neq [] \wedge zs\ @\ ys = xs\}$
by (*cases* $xs = []$) (*auto dest*!: *toplevel-summands-lderivs-Star-nonempty*)

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

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

lemma *lderiv-toplevel-summands-Zero*:
[[*lderivs xs* (*Pr r*) = *Pr s*; *toplevel-summands r* = {*Zero*}]] $\Longrightarrow$ *toplevel-summands s* = {*Zero*}
proof (*induct xs arbitrary*: *r s*)
case (*Cons y ys*)
from *Cons.prems*(*1*) **have** *toplevel-summands* (*PLUS* (*map* ($\lambda a.$ *lderiv a r*) (*embed y*))) = {*Zero*}
proof (*cases embed y* = [])
case *False*
show *?thesis* **using** *Cons.prems*(*2*) **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*:
[[$xs \neq []$; *lderivs xs* (*Pr r*) = *Pr s*]] $\Longrightarrow$
toplevel-summands s $\subseteq$ {*Zero*} $\vee$ *toplevel-summands s* $\subseteq$ ($\bigcup$ *xs*. *toplevel-summands* (*lderivs xs r*))
proof (*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 lderivs.simps*)
next
  case (*Cons z zs*)
  have *toplevel-summands s* $\subseteq$ {*Zero*} $\vee$ *toplevel-summands s* $\subseteq$
    ($\bigcup$*xs. toplevel-summands* (*lderivs xs* (*PLUS* (*map* (λ*a. lderiv a r*) (*embed y*))))) (**is** - $\vee$ *?B*)
    by (*rule* $*$(*1*)) (*auto simp*: *Cons* $*$(*3*)[*symmetric*])
  thus *?thesis*
  proof
    assume *?B*
    also have ... $\subseteq$ ($\bigcup$*xs. toplevel-summands* (*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*:
{*lderivs xs* (*Pr r*) | *xs. True*} $\subseteq$
  {*Pr s* | *s. toplevel-summands s* $\subseteq$ {*Zero*} $\vee$
    toplevel-summands s $\subseteq$ ($\bigcup$*xs. toplevel-summands* (*lderivs xs r*))}
(**is** *?L* $\subseteq$*?R*)
proof (*rule subsetI*)
fix *s* **assume** *s* $\in$ *?L*
then obtain *xs* **where** *s* = *lderivs xs* (*Pr r*) **by** *blast*
moreover obtain *t* **where** *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*: *toplevel-summands r* $\subseteq$ {*Zero*} $\Longrightarrow$ $\ll$*r*$\gg$ = *Zero*
by (*subst ACI-norm-flatten*) (*auto dest*: *subset-singletonD*)

lemma *ACI-norm-lderivs-Pr*:
ACI-norm ‘ {*lderivs xs* (*Pr r*) | *xs. True*} $\subseteq$
  {*Pr Zero*} $\cup$ {*Pr* $\ll$*s*$\gg$ | *s. toplevel-summands s* $\subseteq$ ($\bigcup$*xs. toplevel-summands* $\ll$*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* :: *′a rexp*
assume $*$: *x* = $\ll$*x′*$\gg$ *x′* = *Pr s* **and** *toplevel-summands s* $\subseteq$ {*Zero*}
hence $\ll$*Pr s*$\gg$ = *Pr Zero* **using** *ACI-norm-toplevel-summands-Zero* **by** *simp*
thus *x* $\in$ {*Pr Zero*} $\cup$
  {*Pr* $\ll$*s*$\gg$ |*s. toplevel-summands s* $\subseteq$ ($\bigcup$*xs. toplevel-summands* $\ll$*lderivs xs r*$\gg$)}

unfolding $*$ **by** *blast*
next
fix x x' s :: $'a$ *rexp*
assume $*$: $x = \ll x' \gg$ $x' = Pr\ s$ **and** *toplevel-summands* $s \subseteq (\bigcup xs.$ *toplevel-summands* (*lderivs xs r*))
hence *toplevel-summands* $\ll s \gg \subseteq (\bigcup xs.$ *toplevel-summands* $\ll$*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*(*9*)
..
ultimately show $x \in \{Pr\ Zero\} \cup$
$\{Pr \ll s \gg |s.$ *toplevel-summands* $s \subseteq (\bigcup xs.$ *toplevel-summands* $\ll$*lderivs xs r*$\gg$)}
by *blast*
qed

lemma *finite-ACI-norm-toplevel-summands*: *finite B* $\Longrightarrow$ *finite* $\{f \ll s \gg |s.$ *toplevel-summands* $s \subseteq B\}$
by (*elim finite-surj*[*OF iffD2*[*OF finite-Pow-iff*], *of - - f o flatten PLUS o image ACI-norm*])
(*auto simp*: *Pow-def image-Collect ACI-norm-flatten*)

lemma *lderivs-Not*: *lderivs xs* (*Not r*) = *Not* (*lderivs xs r*)
by (*induct xs arbitrary*: *r*) *auto*

lemma *lderivs-Inter*: *lderivs xs* (*Inter r s*) = *Inter* (*lderivs xs r*) (*lderivs xs s*)
by (*induct xs arbitrary*: *r s*) *auto*

theorem *finite-lderivs*: *finite* $\{\ll$*lderivs xs r*$\gg$ | *xs* . *True*}
proof (*induct r*)
case *Zero* **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* ($\bigcup$ *toplevel-summands* ' $\{\ll$*lderivs xs s*$\gg$ | *xs* . *True*}) **by** *auto*
moreover have $\{\ll r' \gg |r'. \exists ys.\ r' \in$ *toplevel-summands* (*local.lderivs ys s*)} =
$\{r'. \exists ys.\ r' \in$ *toplevel-summands* $\ll$*local.lderivs ys s*$\gg\}$
unfolding *toplevel-summands-ACI-norm* **by** *auto*
ultimately have *fin*: *finite* $\{\ll r' \gg |r'. \exists ys.\ r' \in$ *toplevel-summands* (*local.lderivs ys s*)}
by (*fastforce intro*: *finite-subset*[*of* - $\bigcup$ *toplevel-summands* ' $\{\ll$*lderivs xs s*$\gg$ | *xs* . *True*}])

let $?X = \lambda xs.\ \{Times\ (lderivs\ ys\ r)\ s \mid ys.\ True\} \cup \{r'.\ r' \in (\bigcup ys.\ toplevel\text{-}summands$ $(lderivs\ ys\ s))\}$
show *?case* **unfolding** *ACI-norm-flatten*
proof (*rule finite-surj*[*of* $\{X.\ \exists xs.\ X \subseteq ACI\text{-}norm\ `\ ?X\ xs\}$ - *flatten PLUS*])
show *finite* $\{X.\ \exists xs.\ X \subseteq ACI\text{-}norm\ `\ ?X\ xs\}$
using *fin* **by** (*fastforce simp*: *image-Un elim*: *finite-subset*[*rotated*] *intro*: *finite-surj*[*OF Times*(*1*), *of* - $\lambda r.\ Times\ r \ll s \gg$])
qed (*fastforce dest*!: *subsetD*[*OF toplevel-summands-lderivs-Times*] *intro*!: *imageI*)
next
case (*Star r*)
let $?f = \lambda f\ r'.\ Times\ r'\ (Star\ (f\ r))$
let $?X = \{Star\ r\} \cup\ ?f\ id\ `\ \{r'.\ r' \in \{lderivs\ ys\ r | ys.\ True\}\}$
show *?case* **unfolding** *ACI-norm-flatten*
proof (*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* $(\bigcup toplevel\text{-}summands\ `\ \{\ll lderivs\ xs\ r \gg \mid xs\ .\ True\})$ **by** *auto*
have *finite* $(\bigcup xs.\ toplevel\text{-}summands \ll lderivs\ xs\ r \gg)$ **by** (*rule finite-subset*[*OF* - $*$]) *auto*
hence *fin*: *finite* $\{Pr \ll s \gg | s.\ toplevel\text{-}summands\ s \subseteq (\bigcup xs.\ toplevel\text{-}summands \ll lderivs\ xs\ r \gg)\}$
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*]: $wf\ n\ r \Longrightarrow wf\ n\ (lderiv\ w\ r)$
by (*induct r arbitrary*: *n w*) (*auto simp add*: *Let-def*)

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

lemma *lQuot-map-project*:
assumes $as \in \Sigma\ n\ A \subseteq lists\ (\Sigma\ (Suc\ n))$
shows $lQuot\ as\ (map\ project\ `\ A) = map\ project\ `\ (\bigcup a \in set\ (embed\ as).\ lQuot\ a\ A)$ (**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*: $[\![wf\ n\ r;\ w \in \Sigma\ n]\!] \Longrightarrow lang\ n\ (lderiv\ w\ r) = lQuot\ w\ (lang\ n\ r)$
proof (*induct r arbitrary*: *n w*)
  case (*Pr r*)
  hence $*$: $wf\ (Suc\ n)\ r\ \bigwedge w'.\ w' \in set\ (embed\ w) \Longrightarrow\ w' \in \Sigma\ (Suc\ n)$ **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*: $[\![wf\ n\ r;\ wf\text{-}word\ n\ ws]\!] \Longrightarrow lang\ n\ (lderivs\ ws\ r) = lQuots\ ws\ (lang\ n\ r)$
  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 $lderivs\text{-}set\ n\ r\ s \equiv \{(\ll lderivs\ w\ r\gg,\ \ll lderivs\ w\ s\gg) \mid w.\ wf\text{-}word\ n\ w\}$

3.4 Deriving preserves ACI-equivalence

lemma *ACI-PLUS*:
  $list\text{-}all2\ (\lambda r\ s.\ \ll r\gg = \ll s\gg)\ xs\ ys \Longrightarrow\ \ll PLUS\ xs\gg\ =\ \ll PLUS\ ys\gg$
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* ≪*lderiv as* ≪*a*≫≫) = *toplevel-summands* ≪*lderiv as* ≪*r*≫≫
proof (*induct r*)
case (*Plus r1 r2*) **thus** *?case*
unfolding *toplevel-summands.simps toplevel-summands-ACI-norm*
toplevel-summands-lderiv[*of as* ≪*Plus r1 r2*≫] *SUP-def image-Un Union-Un-distrib*
by (*simp add*: *image-UN*)
qed (*auto simp*: *Let-def*)

theorem *ACI-norm-lderiv*:
≪*lderiv as* ≪*r*≫≫ = ≪*lderiv 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-lderiv[*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-lderiv*)
next
case (*Pr r*)
hence *list-all2* (λ*r s*. ≪*r*≫ = ≪*s*≫)
(*map* (λ*a*. *local.lderiv a* ≪*r*≫) (*embed as*)) (*map* (λ*a*. *local.lderiv a r*) (*embed as*))
unfolding *list-all2-map1 list-all2-map2* **by** (*blast intro*: *list-all2-refl*)
thus *?case* **unfolding** *lderiv.simps ACI-norm.simps* **by** (*blast intro*: *ACI-PLUS*)
qed (*simp-all add*: *Let-def*)

corollary *lderiv-preserves*: ≪*r*≫ = ≪*s*≫ ⟹ ≪*lderiv as r*≫ = ≪*lderiv as s*≫
by (*rule box-equals*[*OF - ACI-norm-lderiv ACI-norm-lderiv*]) (*erule arg-cong*)

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

theorem *ACI-norm-lderivs*:
≪*lderivs ws* ≪*r*≫≫ = ≪*lderivs ws r*≫
proof (*induct ws arbitrary*: *r rule*: *rev-induct*)
case (*snoc w ws*) **thus** *?case*
using *ACI-norm-lderiv*[*of w lderivs ws r*] *ACI-norm-lderiv*[*of w lderivs ws* ≪*r*≫]
by *auto*
qed *simp*

end

end

4 Deciding Equivalence of Π-Extended Regular Expressions

4.1 Bisimulation between languages and regular expressions

type-synonym $'a$ *rexp-pair* $=$ $'a$ *list list* $*$ $('a$ *list rexp* $*$ $'a$ *list rexp*$)$
type-synonym $'a$ *rexp-pairs* $=$ $'a$ *rexp-pair list*

context *alphabet*
begin

context
fixes n :: *nat*
begin

coinductive *bisimilar* :: $'a$ *lang* $\Rightarrow$ $'a$ *lang* $\Rightarrow$ *bool* **where**
$K \subseteq$ *lists* $(\Sigma\ n) \Longrightarrow L \subseteq$ *lists* $(\Sigma\ n)$
$\Longrightarrow ([] \in K \longleftrightarrow [] \in L)$
$\Longrightarrow (\bigwedge x.\ x \in \Sigma\ n \Longrightarrow$ *bisimilar* $(lQuot\ x\ K)\ (lQuot\ x\ L))$
$\Longrightarrow$ *bisimilar* $K\ L$

lemma *equal-if-bisimilar*:
assumes $K \subseteq$ *lists* $(\Sigma\ n)$ $L \subseteq$ *lists* $(\Sigma\ n)$ *bisimilar* $K\ L$ **shows** $K = L$
proof (*rule set-eqI*)
  fix w
  from *assms* **show** $w \in K \longleftrightarrow w \in L$
  proof (*induction* w *arbitrary*: $K\ L$)
    case *Nil* **thus** *?case* **by** (*auto elim*: *bisimilar.cases*)
  next
    case (*Cons* $a\ w\ K\ L$)
    show *?case*
    proof *cases*
      assume $a \in \Sigma\ n$
      with ‹*bisimilar* $K\ L$› **have** *bisimilar* $(lQuot\ a\ K)\ (lQuot\ a\ L)$
        by (*auto elim*!: *bisimilar.cases*)
      then have $w \in lQuot\ a\ K \longleftrightarrow w \in lQuot\ a\ L$
        by (*intro Cons.IH*) (*auto elim*!: *bisimilar.cases*)
      thus *?case* **by** (*auto simp*: *lQuot-def*)
    next
      assume $a \notin \Sigma\ n$
      thus *?case* **using** *Cons.prems* **by** *fastforce*
    qed
  qed
qed

lemma *language-coinduct*:
fixes R (**infixl** $\sim$ *50*)

assumes $\bigwedge K\ L.\ K \sim L \Longrightarrow K \subseteq \mathit{lists}\ (\Sigma\ n) \wedge L \subseteq \mathit{lists}\ (\Sigma\ n)$
assumes $K \sim L$
assumes $\bigwedge K\ L.\ K \sim L \Longrightarrow ([] \in K \longleftrightarrow [] \in L)$
assumes $\bigwedge K\ L\ x.\ K \sim L \Longrightarrow x : \Sigma\ n \Longrightarrow \mathit{lQuot}\ x\ K \sim \mathit{lQuot}\ x\ L$
shows $K = L$
apply (*rule equal-if-bisimilar*)
apply (*rule conjunct1*[*OF assms*(*1*)[*OF assms*(*2*)]])
apply (*rule conjunct2*[*OF assms*(*1*)[*OF assms*(*2*)]])
apply (*rule bisimilar.coinduct*[*of R*, *OF* ‹$K \sim L$›])
apply (*auto simp*: *assms*)
done

end

end

context *embed*
begin

definition *is-bisimulation* **where**
is-bisimulation n X =
  $(\forall (r,s) \in X.\ \mathit{wf}\ n\ r \wedge \mathit{wf}\ n\ s \wedge (\mathit{final}\ r \longleftrightarrow \mathit{final}\ s) \wedge$
  $(\forall a{\in}\Sigma\ n.\ (\ll \mathit{lderiv}\ a\ r \gg, \ll \mathit{lderiv}\ a\ s \gg) \in X))$

lemma *bisim-lang-eq*:
fixes *r s* :: *'a rexp*
assumes *bisim*: *is-bisimulation n X*
assumes $(r, s) \in X$
shows *lang n r* = *lang n s*
proof −
  let $?R = \lambda K\ L.\ (\exists (r,s) \in X.\ K = \mathit{lang}\ n\ r \wedge L = \mathit{lang}\ n\ s)$
  show *?thesis*
  proof (*rule language-coinduct*[**where** *R*=*?R*])
    from ‹$(r, s) \in X$› *bisim* **show** *?R* (*lang n r*) (*lang n s*)
      by (*auto split*: *prod.splits simp*: *is-bisimulation-def*)
  next
    fix *K L* **assume** *?R K L*
    then obtain *r s* **where** *rs*: $(r, s) \in X$
      and *KL*: *K* = *lang n r L* = *lang n s* **by** *auto*
    with *bisim* **have** *final r* ⟷ *final s* **and** *wfr*: *wf n r* **and** *wfs*: *wf n s*
      by (*auto simp*: *is-bisimulation-def*)
    thus $[] \in K \longleftrightarrow [] \in L$
      by (*auto simp*: *lang-final*[*of r n*] *lang-final*[*of s n*] *KL*)

next case, but shared context

    from *bisim rs KL lang-subset-lists*
    show $K \subseteq \mathit{lists}\ (\Sigma\ n) \wedge L \subseteq \mathit{lists}\ (\Sigma\ n)$
      unfolding *is-bisimulation-def* **by** *fastforce*

next case, but shared context

fix *a* **assume** ∗: $a \in \Sigma\ n$
with *rs bisim* **have** *witness*: (≪*lderiv a r*≫, ≪*lderiv a s*≫) ∈ *X*
by (*fastforce simp*: *is-bisimulation-def*)
show *?R* (*lQuot a K*) (*lQuot a L*)
using *KL ACI-norm-lang lang-lderiv*[*OF wfr* ∗] *lang-lderiv*[*OF wfs* ∗]
by (*blast intro*!: *bexI*[*OF - witness*])
qed
qed

lemma *lderivs-lang-eq*:
fixes *r s* :: *'a rexp*
assumes *wf n r wf n s*
shows (∀(*r*, *s*) ∈ *lderivs-set n* ≪*r*≫ ≪*s*≫. *final r* = *final s*) = (*lang n r* = *lang n s*) (**is** *?L* = *?R*)
proof
assume *?L*
hence ∀(*r*, *s*) ∈ *lderivs-set n* ≪*r*≫ ≪*s*≫. *wf n r* ∧ *wf n s* ∧ (*final r* ⟷ *final s*)
using *assms* **by** (*auto simp add*: *ACI-norm-lderivs*)
moreover
{ **fix** *r' s' w* **assume** (*r'*, *s'*) ∈ *lderivs-set n r s* **and** ∗: $w \in \Sigma\ n$
then obtain *ws* **where** *ws*: *wf-word n ws r'* = ≪*lderivs ws r*≫ *s'* = ≪*lderivs ws s*≫ **by** *auto*
with ∗ **have** (≪*lderiv w r'*≫, ≪*lderiv w s'*≫) = (≪*lderivs* (*ws* @ [*w*]) *r*≫, ≪*lderivs* (*ws* @ [*w*]) *s*≫)
by (*auto simp*: *ACI-norm-lderiv*)
hence (≪*lderiv w r'*≫, ≪*lderiv w s'*≫) ∈ *lderivs-set n r s*
using ∗ *ws*(*1*) **by** (*auto intro*!: *imageI exI*[*of - ws* @ [*w*]])
}
ultimately have *is-bisimulation n* (*lderivs-set n* ≪*r*≫ ≪*s*≫)
unfolding *is-bisimulation-def* **by** (*auto simp*: *ACI-norm-lderivs*)
hence *lang n* ≪*r*≫ = *lang n* ≪*s*≫ **by** (*intro bisim-lang-eq*) (*auto intro*: *exI*[*of* - []])
thus *?R* **by** (*rule box-equals*[*OF - ACI-norm-lang ACI-norm-lang*])
next
assume *?R* **thus** *?L* **using** *assms lang-lderivs lang-final* **by** (*auto simp*: *ACI-norm-lderivs*) *metis*+
qed

end

4.2 Different normalization function

locale *normalizer* = *embed* Σ *project embed*
for Σ :: *nat* ⇒ *'a* :: *linorder set*
and *project* :: *'a* ⇒ *'a*
and *embed* :: *'a* ⇒ *'a list* +
fixes *norm* :: *'a* :: *linorder rexp* ⇒ *'c*
and *nlang* :: *nat* ⇒ *'c* ⇒ *'a list set*
assumes *lang-norm*: *wf n r* ⟹ *nlang n* (*norm r*) = *lang n r*

begin

abbreviation *nfinal n r* ≡ ([] ∈ *nlang n r*)

lemma *nfinal-final*: *wf n r* ⟹ *nfinal n* (*norm r*) = *final r*
  using *lang-final lang-norm* **by** *blast*

definition *norms* ≡ (%(*r*,*s*). (*norm r*, *norm s*))

lemma *finite-norm*: *finite* {*norm* «*lderivs xs r*» | *xs* . *True*}
  by (*rule finite-surj*[*OF finite-lderivs*, *of - norm*]) *auto*

lemma *finite-norm-lderivs*: *finite* (*norms* ‘ (*lderivs-set n r s*))
  by (*intro finite-subset*[*OF - finite-cartesian-product*[*OF finite-norm finite-norm*]])
(*auto simp*: *norms-def*)

definition *is-nbisimulation* **where**
is-nbisimulation n X =
  (∀ (*r*,*s*) ∈ *X*. *wf n r* ∧ *wf n s* ∧ (*final r* ⟷ *final s*) ∧
  (∀ *a*∈Σ *n*. (*norm* «*lderiv a r*», *norm* «*lderiv a s*») ∈ *norms* ‘ *X*))

lemma *nbisim-lang-eq*:
  fixes *r s* :: *'a rexp*
  assumes *nbisim*: *is-nbisimulation n X*
  assumes (*r*, *s*) ∈ *X*
  shows *lang n r* = *lang n s*
proof −
  let *?R* = λ*K L*. (∃ (*r*,*s*) ∈ *norms* ‘ *X*. *K* = *nlang n r* ∧ *L* = *nlang n s*)
  show *?thesis*
  proof (*rule language-coinduct*[**where** *R*=*?R*])
    from ‹(*r*, *s*) ∈ *X*› *nbisim* **show** *?R* (*lang n r*) (*lang n s*)
      by (*auto split*: *prod.splits simp*: *lang-norm norms-def is-nbisimulation-def*)
  next
    fix *K L* **assume** *?R K L*
    then obtain *r s* **where** *rs*: (*r*, *s*) ∈ *X*
      and *KL*: *K* = *nlang n* (*norm r*) *L* = *nlang n* (*norm s*) **by** (*auto simp*:
norms-def)
    with *nbisim* **have** *final r* ⟷ *final s* **and** *wfr*: *wf n r* **and** *wfs*: *wf n s*
      by (*auto simp*: *is-nbisimulation-def*)
    thus [] ∈ *K* ⟷ [] ∈ *L*
      by (*auto simp*: *lang-norm*[*OF wfr*] *lang-norm*[*OF wfs*] *lang-final*[*of r n*]
lang-final[*of s n*] *KL*)

next case, but shared context

    from *nbisim rs KL lang-subset-lists*
    show *K* ⊆ *lists* (Σ *n*) ∧ *L* ⊆ *lists* (Σ *n*)
      unfolding *is-nbisimulation-def lang-norm*[*OF wfr*] *lang-norm*[*OF wfs*] **by**
fastforce

next case, but shared context

fix *a* **assume** ∗: $a \in \Sigma\ n$
with *rs nbisim* **have** *witness*: (*norm* ≪*lderiv a r*≫, *norm* ≪*lderiv a s*≫) ∈ *norms* ‘ *X*
by (*fastforce simp*: *is-nbisimulation-def*)
show *?R* (*lQuot a K*) (*lQuot a L*)
using *KL*[*unfolded lang-norm*[*OF wfr*] *lang-norm*[*OF wfs*]]
trans[*OF lang-norm*[*OF iffD2*[*OF ACI-norm-wf*, *OF wf-lderiv*[*OF wfr*]]] *ACI-norm-lang*]
trans[*OF lang-norm*[*OF iffD2*[*OF ACI-norm-wf*, *OF wf-lderiv*[*OF wfs*]]] *ACI-norm-lang*]
lang-lderiv[*OF wfr* ∗] *lang-lderiv*[*OF wfs* ∗]
by (*blast intro*!: *bexI*[*OF - witness*])
qed
qed

lemma *norm-lderivs-lang-eq*:
fixes *r s* :: *'a rexp*
assumes *wf n r wf n s*
shows (∀ (*r*, *s*) ∈ *norms* ‘ *lderivs-set n* ≪*r*≫ ≪*s*≫. *nfinal n r* = *nfinal n s*) = (*lang n r* = *lang n s*)
by (*rule trans*[*OF - lderivs-lang-eq*[*OF assms*]]) (*fastforce simp*: *norms-def assms nfinal-final*)

end

Closure computation

primrec *remdups'* **where**
remdups' f [] = []
| *remdups' f* (*x* # *xs*) =
(*case List.find* (λ*y*. *f x* = *f y*) *xs of None* ⇒ *x* # *remdups' f xs* | - ⇒ *remdups' f xs*)

lemma *map-remdups'*[*simp*]: *map f* (*remdups' f xs*) = *remdups* (*map f xs*)
by (*induct xs*) (*auto split*: *option.splits simp add*: *find-Some-iff find-None-iff*)

lemma *remdups'-map*[*simp*]: *remdups' f* (*map g xs*) = *map g* (*remdups'* (*f o g*) *xs*)
by (*induct xs*) (*auto split*: *option.splits simp add*: *find-None-iff*,
auto simp: *find-Some-iff elim*: *imageI*[*OF nth-mem*])

lemma *map-apfst-remdups'*:
map (*f o fst*) (*remdups' snd xs*) = *map fst* (*remdups' snd* (*map* (*apfst f*) *xs*))
by (*auto simp*: *comp-def*)

lemma *set-remdups'*[*simp*]: *f* ‘ *set* (*remdups' f xs*) = *f* ‘ *set xs*
by (*induct xs*) (*auto split*: *option.splits simp add*: *find-Some-iff*)

lemma *subset-remdups'*: *set* (*remdups' f xs*) ⊆ *set xs*
by (*induct xs*) (*auto split*: *option.splits*)

lemma *find-append*[*simp*]:
List.find P (*xs* @ *ys*) = *None* = (*List.find P xs* = *None* ∧ *List.find P ys* = *None*)
by (*induct xs*) *auto*

lemma *subset-remdups′-append*: *set* (*remdups′ f* (*xs* @ *ys*)) ⊆ *set* (*remdups′ f xs*) ∪ *set* (*remdups′ f ys*)
by (*induct xs arbitrary*: *ys*) (*auto split*: *option.splits*)

lemmas *mp-remdups′* = *set-mp*[*OF subset-remdups′*]
lemmas *mp-remdups′-append* = *set-mp*[*OF subset-remdups′-append*]

lemma *inj-on-set-remdups′*[*simp*]: *inj-on f* (*set* (*remdups′ f xs*))
by (*induct xs*) (*auto split*: *option.splits simp add*: *find-None-iff dest*!: *mp-remdups′*)

lemma *distinct-remdups′*[*simp*]: *distinct* (*map f* (*remdups′ f xs*))
by (*induct xs*) (*auto split*: *option.splits simp*: *find-None-iff*)

lemma *distinct-remdups′-strong*: (∀ *x*∈*set xs*. ∀ *y*∈*set xs*. *g x* = *g y* ⟶ *f x* = *f y*) ⟹
distinct (*map g* (*remdups′ f xs*))
proof (*induct xs*)
case (*Cons x xs*) **thus** *?case*
by (*auto split*: *option.splits*) (*fastforce simp*: *find-None-iff dest*!: *mp-remdups′*)
qed *simp*

lemma *set-remdups′-strong*: (∀ *x*∈*set xs*. ∀ *y*∈*set xs*. *g x* = *g y* ⟶ *f x* = *f y*) ⟹
f ‘ *set* (*remdups′ g xs*) = *f* ‘ *set xs*
proof (*induct xs*)
case (*Cons x xs*) **thus** *?case*
by (*clarsimp split*: *option.splits simp add*: *find-Some-iff*)
(*intro insert-absorb*[*symmetric*] *image-eqI*[*OF - nth-mem*, *of - f xs*], *auto*)
qed *simp*

fun *test* **where** *test* (*ws*, -, -) = (*case ws of* [] ⇒ *False* | (*w*,*p*,*q*)#- ⇒ *final p* = *final q*)
fun *test′* **where** *test′* (*ws*, -) = (*case ws of* [] ⇒ *False* | (*p*,*q*)#- ⇒ *final p* = *final q*)

locale *equivalence-checker* =
fixes σ :: *nat* ⇒ ′*a* :: *linorder list*
and π :: ′*a* ⇒ ′*a*
and ε :: ′*a* ⇒ ′*a list*
and *norm* :: ′*a rexp* ⇒ ′*c*
and *nlang* :: *nat* ⇒ ′*c* ⇒ ′*a list set*
assumes *norm*: *normalizer* (*set* ∘ σ) π ε *norm nlang*

sublocale *equivalence-checker* ⊆ *normalizer set* ∘ σ π ε

by (*rule norm*)

context *equivalence-checker*
begin

fun *step* **where** *step n* (*ws*, *ps*, *N*) =
  (*let*
   (*w*, *r*, *s*) = *hd ws*;
   ps' = (*r*, *s*) # *ps*;
   succs = *map* (λ*a*.
    let
     r' = ≪*lderiv a r*≫;
     s' = ≪*lderiv a s*≫
    in ((*a* # *w*, *r'*, *s'*), (*norm r'*, *norm s'*))) (σ *n*);
   new = *remdups' snd* (*filter* (λ(-, *rs*). *rs* $\notin$ *N*) *succs*);
   ws' = *tl ws* @ *map fst new*;
   N' = *set* (*map snd new*) $\cup$ *N*
  in (*ws'*, *ps'*, *N'*))

fun *step'* **where** *step' n* (*ws*, *N*) =
  (*let*
   (*r*, *s*) = *hd ws*;
   succs = *map* (λ*a*.
    let
     r' = ≪*lderiv a r*≫;
     s' = ≪*lderiv a s*≫
    in ((*r'*, *s'*), (*norm r'*, *norm s'*))) (σ *n*);
   new = *remdups' snd* (*filter* (λ(-, *rs*). *rs* $\notin$ *N*) *succs*)
  in (*tl ws* @ *map fst new*, *set* (*map snd new*) $\cup$ *N*))

lemma *step-unfold*: *step n* (*w* # *ws*, *ps*, *N*) = (*ws'*, *ps'*, *N'*) $\Longrightarrow$ ($\exists$ *xs r s*.
 w = (*xs*, *r*, *s*) $\wedge$ *ps'* = (*r*, *s*) # *ps* $\wedge$
 ws' = *ws* @ *remdups'* (*norms o snd*) (*filter* (λ(-, *p*). *norms p* $\notin$ *N*)
  (*map* (λ*a*. (*a*#*xs*, ≪*lderiv a r*≫, ≪*lderiv a s*≫)) (σ *n*))) $\wedge$
 N' = *set* (*map* (λ*a*. (*norm* ≪*lderiv a r*≫, *norm* ≪*lderiv a s*≫)) (σ *n*)) $\cup$ *N*)
 by (*auto split*: *prod.splits dest*!: *mp-remdups'*
 simp: *Let-def norms-def filter-map set-n-lists image-Collect image-image comp-def*)

definition *closure* **where** *closure n* = *while-option test* (*step n*)
definition *closure'* **where** *closure' n* = *while-option test'* (*step' n*)

definition *pre-bisim* **where**
 pre-bisim n r s = (λ(*ws*, *ps*, *N*).
  (≪*r*≫, ≪*s*≫) $\in$ *snd* ‘ *set ws* $\cup$ *set ps* $\wedge$
  distinct (*map snd ws* @ *ps*) $\wedge$
  bij-betw norms (*set* (*map snd ws* @ *ps*)) *N* $\wedge$
  ($\forall$(*w*, *r'*, *s'*) $\in$ *set ws*. ≪*lderivs* (*rev w*) *r*≫ = *r'* $\wedge$ ≪*lderivs* (*rev w*) *s*≫ = *s'* $\wedge$
   wf-word n (*rev w*) $\wedge$ *wf n r'* $\wedge$ *wf n s'*) $\wedge$
  ($\forall$(*r'*, *s'*) $\in$ *set ps*. ($\exists$ *w*. ≪*lderivs w r*≫ = *r'* $\wedge$ ≪*lderivs w s*≫ = *s'*) $\wedge$

$wf\ n\ r' \wedge wf\ n\ s' \wedge (final\ r' \longleftrightarrow final\ s') \wedge$
$(\forall\, a{\in}set\ (\sigma\ n).\ (norm\ \ll lderiv\ a\ r'\gg,\ norm\ \ll lderiv\ a\ s'\gg) \in N)))$

lemma *pre-bisim-start*:
$[\![wf\ n\ r;\ wf\ n\ s]\!] \Longrightarrow pre\text{-}bisim\ n\ r\ s\ ([([], \ll r\gg, \ll s\gg)], [], \{(norm\ \ll r\gg, norm\ \ll s\gg)\})$
by (*auto simp add*: *pre-bisim-def bij-betw-def norms-def*)

lemma *step-mono*:
assumes $step\ n\ (ws,\ ps,\ N) = (ws',\ ps',\ N')$
shows $snd\ \text{`}\ set\ ws \cup set\ ps \subseteq snd\ \text{`}\ set\ ws' \cup set\ ps'$
using *assms* **proof** (*intro subsetI*, *elim UnE*)
fix x **assume** $x \in snd\ \text{`}set\ ws$
with *assms* **show** $x \in snd\ \text{`}\ set\ ws' \cup set\ ps'$
proof ($cases\ x = snd\ (hd\ ws)$)
case *False* **with** $\langle x \in image\ snd\ (set\ ws)\rangle$ **have** $x \in snd\ \text{`}\ set\ (tl\ ws)$ **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 *pre-bisim-step*: $pre\text{-}bisim\ n\ r\ s\ st \Longrightarrow test\ st \Longrightarrow pre\text{-}bisim\ n\ r\ s\ (step\ n\ st)$
proof (*unfold pre-bisim-def*, (*split prod.splits*)+, *elim prod-caseE conjE*, *clarify*, *intro allI impI conjI*)
fix $ws\ ps\ N\ ws'\ ps'\ N'$
assume *test*: $test\ (ws,\ ps,\ N)$
and *step*: $step\ n\ (ws,\ ps,\ N) = (ws',\ ps',\ N')$
and *rs*: $(\ll r\gg, \ll s\gg) \in snd\ \text{`}\ set\ ws \cup set\ ps$
and *distinct*: $distinct\ (map\ snd\ ws\ @\ ps)$
and *bij*: $bij\text{-}betw\ norms\ (set\ (map\ snd\ ws\ @\ ps))\ N$
and *ws*: $\forall (w,\ r',\ s') \in set\ ws.\ \ll lderivs\ (rev\ w)\ r\gg = r' \wedge \ll lderivs\ (rev\ w)\ s\gg = s' \wedge$
$wf\text{-}word\ n\ (rev\ w) \wedge wf\ n\ r' \wedge wf\ n\ s'$
(**is** $\forall (w,\ r',\ s') \in set\ ws.\ ?ws\ w\ r'\ s'$)
and *ps*: $\forall (r',\ s') \in set\ ps.\ (\exists\, w.\ \ll lderivs\ w\ r\gg = r' \wedge \ll lderivs\ w\ s\gg = s') \wedge$
$wf\ n\ r' \wedge wf\ n\ s' \wedge (final\ r' \longleftrightarrow final\ s') \wedge$
$(\forall\, a{\in}set\ (\sigma\ n).\ (norm\ \ll lderiv\ a\ r'\gg,\ norm\ \ll lderiv\ a\ s'\gg) \in N)$
(**is** $\forall (r,\ s) \in set\ ps.\ ?ps\ r\ s\ N$)
from *test* **obtain** $x\ xs$ **where** *ws-Cons*: $ws = x\#xs$ **by** (*cases ws*) *auto*
obtain $w\ r'\ s'$ **where** x: $x = (w,\ r',\ s')$ **and** ps': $ps' = (r',\ s')\ \#\ ps$
and ws': $ws' = xs\ @\ remdups'\ (norms\ o\ snd)\ (filter\ (\lambda(\text{-},\ p).\ norms\ p \notin N)$
$(map\ (\lambda a.\ (a\ \#\ w,\ \ll lderiv\ a\ r'\gg,\ \ll lderiv\ a\ s'\gg))\ (\sigma\ n)))$
and N': $N' = (set\ (map\ (\lambda a.\ (norm\ \ll lderiv\ a\ r'\gg,\ norm\ \ll lderiv\ a\ s'\gg))\ (\sigma\ n)) - N) \cup N$
using *step-unfold*[*OF step*[*unfolded ws-Cons*]] **by** *blast*
hence $ws'ps'$: $set\ (map\ snd\ ws'\ @\ ps') = set\ (remdups'\ norms\ (filter\ (\lambda p.\ norms\ p \notin N)$
$(map\ (\lambda a.\ (\ll lderiv\ a\ r'\gg,\ \ll lderiv\ a\ s'\gg))\ (\sigma\ n)))) \cup (set\ (map\ snd\ ws\ @\ ps))$

unfolding *ws′ ps′ ws-Cons x* **by** (*auto dest*!: *mp-remdups′ simp*: *filter-map image-image image-Un o-def*)
from *rs step* **show** (≪*r*≫, ≪*s*≫) ∈ *snd* ‘ *set ws′* ∪ *set ps′* **by** (*blast dest*: *step-mono*)

from *distinct ps′ ws′ ws-Cons x bij* **show** *distinct* (*map snd ws′* @ *ps′*)
by (*auto simp*: *bij-betw-def*
intro!: *imageI*[*of* - - *norms*] *distinct-remdups′-strong*
dest!: *mp-remdups′*
elim: *image-eqI*[*of* - *snd*, *OF sym*[*OF snd-conv*]])

from *ps′ ws′ N′ ws x bij* **show** *bij-betw norms* (*set* (*map snd ws′* @ *ps′*)) *N′*
unfolding *ws′ps′ N′* **by** (*intro bij-betw-combine*[*OF* - *bij*]) (*auto simp*: *bij-betw-def norms-def*)

from *ws x ws-Cons* **have** *wr′s′*: *?ws w r′ s′* **by** *auto*
with *ws ws-Cons* **show** ∀ (*w*, *r′*, *s′*) ∈ *set ws′*. *?ws w r′ s′* **unfolding** *ws′*
by (*auto dest*!: *mp-remdups′ simp*: *ACI-norm-lderiv elim*!: *set-mp*)

from *ps wr′s′ test*[*unfolded ws-Cons x*] **show** ∀ (*r′*, *s′*) ∈ *set ps′*. *?ps r′ s′ N′*
unfolding *ps′ N′*
by (*fastforce simp*: *image-Collect*)
qed

lemma *step-commute*: *ws* ≠ [] ⟹ (*case step n* (*ws*, *ps*, *N*) *of* (*ws′*, *ps′*, *N′*) ⇒ (*map snd ws′*, *N′*)) = *step′ n* (*map snd ws*, *N*)
apply (*auto split*: *prod.splits*)
apply (*auto simp only*: *step.simps step′.simps Let-def map-apfst-remdups′ filter-map List.map.compositionality*[*unfolded comp-def*] *apfst-def map-pair-def snd-conv id-def*)
apply (*auto simp*: *filter-map comp-def map-tl hd-map*)
apply (*intro image-eqI*, *auto*)+
done

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

theorem *closure-sound*:
assumes *result*: *closure n* ([([], ≪*r*≫, ≪*s*≫)], [], {(*norm* ≪*r*≫, *norm* ≪*s*≫)}) = *Some*([], *ps*, *N*)
and *wf*: *wf n r wf n s*
shows *lang n r* = *lang n s*
proof −
from *pre-bisim-step pre-bisim-start*[*OF wf*] **have** *pre-bisim-ps*: *pre-bisim n r s* ([], *ps*, *N*)
by (*rule while-option-rule*[*OF* - *result*[*unfolded closure-def*]])

then have *is-nbisimulation n* (*set ps*) (≪*r*≫, ≪*s*≫) ∈ *set ps*
by (*auto simp*: *bij-betw-def pre-bisim-def is-nbisimulation-def in-lists-conv-set norms-def*)
hence *lang n* ≪*r*≫ = *lang n* ≪*s*≫
by (*intro nbisim-lang-eq image-eqI*) *auto*
thus *lang n r* = *lang n s* **unfolding** *ACI-norm-lang* **.**
qed

theorem *closure'-sound*:
assumes *result*: *closure' n* ([(≪*r*≫, ≪*s*≫)], {(*norm* ≪*r*≫, *norm* ≪*s*≫)}) = *Some*([], *N*)
and *wf*: *wf n r wf n s*
shows *lang n r* = *lang n s*
using *wf trans*[*OF closure-closure'*[*of n* [([], ≪*r*≫, ≪*s*≫)] [] {(*norm* ≪*r*≫, *norm* ≪*s*≫)}, *simplified*]
result, *unfolded option-map-eq-Some*]
by (*auto dest*: *closure-sound*)

theorem *closure-termination*:
assumes *wf*: *wf n r wf n s*
and *cl*: *closure n* ([([], ≪*r*≫, ≪*s*≫)], [], {(*norm* ≪*r*≫, *norm* ≪*s*≫)}) = *None* (**is** *?cl* = *None*)
shows *False*
proof −
let *?D* = {*norm* ≪*lderivs xs r*≫ | *xs* . *True*} × {*norm* ≪*lderivs xs s*≫ | *xs* . *True*}
let *?X* = λ*ps*. *?D* − *norms* ' *set ps*
let *?f* = λ(*ws*, *ps*, *N*). *card* (*?X ps*)
have ∃ *st*. *?cl* = *Some st* **unfolding** *closure-def*
proof (*rule measure-while-option-Some*[*of pre-bisim n r s - - ?f*], *intro conjI*)
fix *st* **assume** *pre-bisim*: *pre-bisim n r s st* **and** *test st*
hence *pre-bisim-step*: *pre-bisim n r s* (*step n st*) **by** (*rule pre-bisim-step*)
obtain *ws ps N* **where** *st*: *st* = (*ws*, *ps*, *N*) **by** (*cases st*) *blast*
hence *finite* (*?X ps*) **by** (*blast intro*: *finite-cartesian-product finite-norm*)
moreover obtain *ws' ps' N'* **where** *step*: *step n* (*ws*, *ps*, *N*) = (*ws'*, *ps'*, *N'*)
by (*cases step n* (*ws*, *ps*, *N*)) *blast*
moreover
{ have *norms* ' *set ps* ⊆ *?D* **using** *pre-bisim*[*unfolded st pre-bisim-def*]
by (*auto simp*: *norms-def ACI-norm-lderivs*)
moreover
have *norms* ' *set ps'* ⊆ *?D* **using** *pre-bisim-step*[*unfolded st step pre-bisim-def*]
by (*auto simp*: *norms-def ACI-norm-lderivs*)
moreover
{ have *distinct* (*map snd ws* @ *ps*) *inj-on norms* (*set* (*map snd ws* @ *ps*))
using *pre-bisim*[*unfolded st pre-bisim-def*] **by** (*auto simp*: *bij-betw-def*)
hence *distinct* (*map norms* (*map snd ws* @ *ps*)) **unfolding** *distinct-map* **..**
hence *norms* ' *set ps* ⊂ *norms* ' *set* (*snd* (*hd ws*) # *ps*) **using** ‹*test st*› *st*
by (*cases ws*) *auto*
moreover have *norms* ' *set ps'* = *norms* ' *set* (*snd* (*hd ws*) # *ps*)

using *step* **by** (*auto split*: *prod.splits*)
ultimately have *norms* ‘ *set ps* ⊂ *norms* ‘ *set ps′* **by** *simp*
}
ultimately have *?X ps′* ⊂ *?X ps* **by** (*auto simp add*: *image-set simp del*: *set-map*)
}
ultimately show *?f* (*step n st*) < *?f st* **unfolding** *st step*
using *psubset-card-mono*[*of ?X ps ?X ps′*] **by** *simp*
qed (*auto simp add*: *pre-bisim-start*[*OF wf*] *pre-bisim-step*)
thus *False* **using** *cl* **by** *auto*
qed

theorem *closure′-termination*:
assumes *wf*: *wf n r wf n s*
and *cl*: *closure′ n* ([(≪*r*≫, ≪*s*≫)], {(*norm* ≪*r*≫, *norm* ≪*s*≫)}) = *None*
shows *False*
using *wf trans*[*OF closure-closure′*[*of n* [([], ≪*r*≫, ≪*s*≫)] [] {(*norm* ≪*r*≫, *norm* ≪*s*≫)}, *simplified*]
cl, *unfolded option-map-is-None*]
by (*auto intro*: *closure-termination*)

theorem *closure-complete*:
assumes *eq*: *lang n r* = *lang n s*
and *wf*: *wf n r wf n s*
shows ∃ *ps N*. *closure n* ([([], ≪*r*≫, ≪*s*≫)], [], {(*norm* ≪*r*≫, *norm* ≪*s*≫)}) = *Some*([], *ps*, *N*)
(**is** ∃ - -. *?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*)
then obtain *w r′ s′* **where** *wrs*: *wrs* = (*w*, *r′*, *s′*) **by** (*cases wrs*) *blast*
with *ws-ps-N Cons* **have** *final r′* ≠ *final s′*
using *while-option-stop2*[*OF Some*[*unfolded closure-def*]] **by** *simp*
moreover
from *pre-bisim-step pre-bisim-start*[*OF wf*] **have** *pre-bisim-ps*: *pre-bisim n r s st*
by (*rule while-option-rule*[*OF - Some*[*unfolded closure-def*]])
hence ≪*lderivs* (*rev w*) *r*≫ = *r′* ≪*lderivs* (*rev w*) *s*≫ = *s′* *wf-word n* (*rev w*)
unfolding *ws-ps-N Cons wrs pre-bisim-def ACI-norm-lderivs* **by** *auto*
ultimately show *?thesis* **using** *eq wf lderivs-final* **by** *auto*
qed *blast*
qed (*auto intro*: *closure-termination*[*OF wf*])

theorem *closure′-complete*:
assumes *eq*: *lang n r* = *lang n s*
and *wf*: *wf n r wf n s*

shows $\exists N.$ *closure′ n* ([(≪*r*≫, ≪*s*≫)], {(*norm* ≪*r*≫, *norm* ≪*s*≫)}) = *Some*([], *N*)
using *assms closure-closure′*[*of n* [([], ≪*r*≫, ≪*s*≫)] [] {(*norm* ≪*r*≫, *norm* ≪*s*≫)}, *symmetric*]
by (*auto dest*!: *closure-complete*)

The overall procedure

definition *check-eqv* **where**
check-eqv n r s ⟷ *wf n r* ∧ *wf n s* ∧
(*let r′* = ≪*r*≫; *s′* = ≪*s*≫ *in* (*case closure n* ([([], *r′*, *s′*)], [], {(*norm r′*, *norm s′*)}) *of*
Some([],-) ⇒ *True* | - ⇒ *False*))

definition *check-eqv-counterexample* **where**
check-eqv-counterexample n r s =
(*let r′* = ≪*r*≫; *s′* = ≪*s*≫ *in* (*case closure n* ([([], *r′*, *s′*)], [], {(*norm r′*, *norm s′*)}) *of*
Some([],-) ⇒ *None* | *Some*((*w*,-,-) # -, -) ⇒ *Some w*))

definition *check-eqv′* **where**
check-eqv′ n r s ⟷ *wf n r* ∧ *wf n s* ∧
(*let r′* = ≪*r*≫; *s′* = ≪*s*≫ *in* (*case closure′ n* ([(*r′*, *s′*)], {(*norm r′*, *norm s′*)}) *of*
Some([],-) ⇒ *True* | - ⇒ *False*))

lemma *check-eqv-check-eqv′*: *check-eqv n r s* = *check-eqv′ n r s*
unfolding *check-eqv-def check-eqv′-def Let-def*
using *closure-closure′*[*of n* [([], ≪*r*≫, ≪*s*≫)] [] {(*norm* ≪*r*≫, *norm* ≪*s*≫)}, *symmetric*]
by (*auto split*: *option.splits list.splits*)

lemma *soundness*:
assumes *check-eqv n r 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*)

lemma *soundness′*:
assumes *check-eqv′ n r s*
shows *lang n r* = *lang n s*
using *soundness check-eqv-check-eqv′ assms* **by** *auto*

lemma *completeness*:
assumes *lang n r* = *lang n s wf n r wf n s*
shows *check-eqv n r s*
using *closure-complete*[*OF assms*] *assms*(*2*,*3*) **by** (*auto simp*: *check-eqv-def*)

lemma *completeness′*:
assumes *lang n r* = *lang n s wf n r wf n s*
shows *check-eqv′ n r s*
using *completeness check-eqv-check-eqv′ assms* **by** *auto*

end

end

5 Normalization of Π-Extended Regular Expressions

5.1 Normalizing Constructors

lemma *not-less-Zero*[*elim*!]: $r < Zero \Longrightarrow P$
  by (*induct r*) (*auto simp*: *less-rexp-def*)

fun *nPlus* :: *'a*::*linorder rexp* ⇒ *'a rexp* ⇒ *'a rexp*
where
  nPlus Zero r = *r*
| *nPlus r Zero* = *r*
| *nPlus* (*Plus r1 r2*) (*Plus s1 s2*) =
  (*if r1* < *s1 then Plus r1* (*nPlus r2* (*Plus s1 s2*))
  else if s1 < *r1 then Plus s1* (*nPlus* (*Plus r1 r2*) *s2*)
  else nPlus (*Plus r1 r2*) *s2*)
| *nPlus* (*Plus r1 r2*) *s* =
  (*if s* = *Not Zero then Not Zero*
  else if r1 < *s then Plus r1* (*nPlus r2 s*)
  else if s < *r1 then Plus s* (*Plus r1 r2*)
  else Plus r1 r2)
| *nPlus r* (*Plus s1 s2*) =
  (*if r* = *Not Zero then Not Zero*
  else if r < *s1 then Plus r* (*Plus s1 s2*)
  else if s1 < *r then Plus s1* (*nPlus r s2*)
  else Plus s1 s2)
| *nPlus r s* =
  (*if r* = *Not Zero* ∨ *s* = *Not Zero then Not Zero*
  else if r < *s then Plus r s*
  else if s < *r then Plus s r*
  else r)

fun *nTimes* :: *'a rexp* ⇒ *'a rexp* ⇒ *'a rexp*
where
  nTimes Zero - = *Zero*
| *nTimes* - *Zero* = *Zero*
| *nTimes One r* = *r*
| *nTimes r One* = *r*
| *nTimes* (*Times r s*) *t* = *Times r* (*nTimes s t*)
| *nTimes r s* = *Times r s*

fun *nStar* :: *'a rexp* ⇒ *'a rexp*
where

```
  nStar Zero = One
| nStar One = One
| nStar (Star r) = nStar r
| nStar r = Star r

fun nInter :: 'a::linorder rexp ⇒ 'a rexp ⇒ 'a rexp
where
  nInter Zero - = Zero
| nInter - Zero = Zero
| nInter (Inter r1 r2) (Inter s1 s2) =
  (if r1 < s1 then Inter r1 (nInter r2 (Inter s1 s2))
  else if s1 < r1 then Inter s1 (nInter (Inter r1 r2) s2)
  else nInter (Inter r1 r2) s2)
| nInter (Inter r1 r2) s =
  (if s = Not Zero then Inter r1 r2
  else if r1 < s then Inter r1 (nInter r2 s)
  else if s < r1 then Inter s (Inter r1 r2)
  else Inter r1 r2)
| nInter r (Inter s1 s2) =
  (if r = Not Zero then Inter s1 s2
  else if r < s1 then Inter r (Inter s1 s2)
  else if s1 < r then Inter s1 (nInter r s2)
  else Inter s1 s2)
| nInter r s =
  (if r = Not Zero then s
  else if s = Not Zero then r
  else if r < s then Inter r s
  else if s < r then Inter s r
  else r)

fun nNot :: 'a::linorder rexp ⇒ 'a rexp
where
  nNot (Not r) = r
| nNot (Plus r s) = nInter (nNot r) (nNot s)
| nNot (Inter r s) = nPlus (nNot r) (nNot s)
| nNot r = Not r

fun nPr :: 'a rexp ⇒ 'a rexp
where
  nPr Zero = Zero
| nPr One = One
| nPr (Plus r s) = Plus (nPr r) (nPr s)
| nPr (Times r s) = Times (nPr r) (nPr s)
| nPr (Star r) = Star (nPr r)
| nPr r = Pr r

fun norm :: ('a::linorder) rexp ⇒ 'a rexp where
  norm Zero = Zero
| norm One = One
```

| *norm* (*Atom a*) = *Atom a*
| *norm* (*Plus r s*) = *nPlus* (*norm r*) (*norm s*)
| *norm* (*Times r s*) = *nTimes* (*norm r*) (*norm s*)
| *norm* (*Star r*) = *nStar* (*norm r*)
| *norm* (*Not r*) = *nNot* (*norm r*)
| *norm* (*Inter r s*) = *nInter* (*norm r*) (*norm s*)
| *norm* (*Pr r*) = *nPr* (*norm r*)

context *alphabet*
begin

lemma *wf-nPlus*[*simp*]: ⟦*wf n r*; *wf n s*⟧ ⟹ *wf n* (*nPlus r s*)
  by (*induct r s rule*: *nPlus.induct*) *auto*

lemma *wf-nTimes*[*simp*]: ⟦*wf n r*; *wf n s*⟧ ⟹ *wf n* (*nTimes r s*)
  by (*induct r s rule*: *nTimes.induct*) *auto*

lemma *wf-nStar*[*simp*]: *wf n r* ⟹ *wf n* (*nStar r*)
  by (*induct r rule*: *nStar.induct*) *auto*

lemma *wf-nInter*[*simp*]: ⟦*wf n r*; *wf n s*⟧ ⟹ *wf n* (*nInter r s*)
  by (*induct r s rule*: *nInter.induct*) *auto*

lemma *wf-nNot*[*simp*]: *wf n r* ⟹ *wf n* (*nNot r*)
  by (*induct r rule*: *nNot.induct*) *auto*

lemma *wf-nPr*[*simp*]: *wf* (*Suc n*) *r* ⟹ *wf n* (*nPr r*)
  by (*induct r rule*: *nPr.induct*) *auto*

lemma *wf-norm*[*simp*]: *wf n r* ⟹ *wf n* (*norm r*)
  by (*induct r arbitrary*: *n*) *auto*

end

context *project*
begin

lemma *Plus-Not-Zero*:
  wf n r ⟹ *lang n* (*Plus* (*Not Zero*) *r*) = *lang n* (*Not Zero*)
  wf n r ⟹ *lang n* (*Plus r* (*Not Zero*)) = *lang n* (*Not Zero*)
  by (*auto dest*!: *lang-subset-lists*)

lemma *Inter-Not-Zero*:
  wf n r ⟹ *lang n* (*Inter* (*Not Zero*) *r*) = *lang n r*
  wf n r ⟹ *lang n* (*Inter r* (*Not Zero*)) = *lang n r*
  by (*auto dest*!: *lang-subset-lists*)

lemma *lang-nPlus*[*simp*]: ⟦*wf n r*; *wf n s*⟧ ⟹ *lang n* (*nPlus r s*) = *lang n* (*Plus r s*)

by (*induct r s rule*: *nPlus.induct*)
(*auto*, *auto dest*!: *lang-subset-lists dest*: *project*
subsetD[*OF conc-subset-lists*, *unfolded in-lists-conv-set*, *rotated* -1]
subsetD[*OF star-subset-lists*, *unfolded in-lists-conv-set*, *rotated* -1])

lemma *lang-nTimes*[*simp*]: *lang n* (*nTimes r s*) = *lang n* (*Times r s*)
by (*induct r s rule*: *nTimes.induct*) (*auto simp*: *conc-assoc conc-Un-distrib*)

lemma *lang-nStar*[*simp*]: *lang n* (*nStar r*) = *lang n* (*Star r*)
by (*induct r rule*: *nStar.induct*) *auto*

lemma *lang-nInter*[*simp*]: ⟦*wf n r*; *wf n s*⟧ ⟹ *lang n* (*nInter r s*) = *lang n* (*Inter r s*)
by (*induct r s rule*: *nInter.induct*)
(*auto*, *auto dest*!: *lang-subset-lists dest*: *project*
subsetD[*OF conc-subset-lists*, *unfolded in-lists-conv-set*, *rotated* -1]
subsetD[*OF star-subset-lists*, *unfolded in-lists-conv-set*, *rotated* -1])

lemma *lang-nNot*[*simp*]: *wf n r* ⟹ *lang n* (*nNot r*) = *lang n* (*Not r*)
by (*induct r rule*: *nNot.induct*) (*auto dest*!: *lang-subset-lists*)

lemma *lang-nPr*[*simp*]: *lang n* (*nPr r*) = *lang n* (*Pr r*)
by (*induct r rule*: *nPr.induct*) *auto*

lemma *lang-norm*[*simp*]: *wf n r* ⟹ *lang n* (*norm r*) = *lang n r*
by (*induct r arbitrary*: *n*) *auto*

end

end

theory *Regular-Operators*
imports *Derivatives* ~~/*src*/*HOL*/*Library*/*While-Combinator*
begin

primrec *REV* :: '*a rexp* ⇒ '*a rexp* **where**
REV Zero = *Zero*
| *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* $\Rightarrow$ *'a rexp* $\Rightarrow$ *'a rexp* **where**
  rderiv - *Zero* = *Zero*
| *rderiv* - *One* = *Zero*
| *rderiv as* (*Atom bs*) = (*if as* = *bs then One else Zero*)
| *rderiv as* (*Plus r s*) = *Plus* (*rderiv as r*) (*rderiv as s*)
| *rderiv as* (*Times r s*) =
    (*let rs'* = *Times r* (*rderiv as s*)
     in if final s then Plus rs' (*rderiv as r*) *else rs'*)
| *rderiv as* (*Star r*) = *Times* (*Star r*) (*rderiv as r*)
| *rderiv as* (*Not r*) = *Not* (*rderiv as r*)
| *rderiv as* (*Inter r s*) = *Inter* (*rderiv as r*) (*rderiv as s*)
| *rderiv as* (*Pr r*) = *Pr* (*PLUS* (*map* (λ*a*. *rderiv a r*) (*embed as*)))

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

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

lemma *rderivs-append*: *rderivs* (*ws* @ *ws'*) *r* = *rderivs ws'* (*rderivs 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 *rderivs-lderivs*: *rderivs w r* = *REV* (*lderivs w* (*REV r*))
  by (*induct w arbitrary*: *r*) (*auto simp*: *rderiv-lderiv*)

lemma *wf-rderiv*[*simp*]: *wf n r* $\Longrightarrow$ *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* $\Longrightarrow$ *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*$\rrbracket \Longrightarrow$ *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*$\rrbracket \Longrightarrow$ *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*: «*REV* «*r*»» = «*REV r*»
proof (*induct r*)
case (*Plus r s*)
show *?case*
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*: «*rderiv as* «*r*»» = «*rderiv as r*»
unfolding *rderiv-lderiv* **by** (*metis ACI-norm-REV ACI-norm-lderiv*)

lemma *ACI-norm-rderivs*: «*rderivs w* «*r*»» = «*rderivs w r*»
unfolding *rderivs-lderivs* **by** (*metis ACI-norm-REV ACI-norm-lderivs*)

theorem *finite-rderivs*: *finite* {«*rderivs xs r*» | *xs* . *True*}
unfolding *rderivs-lderivs*
by (*subst ACI-norm-REV*[*symmetric*]) (*auto intro*: *finite-surj*[*OF finite-lderivs*, *of* - λ*r*. «*REV r*»])

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*]: ⟦*wf n r*; *wf-word n w*; $a \in \Sigma\ n$⟧ ⟹
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*]: ⟦*wf n r*; *wf-word n w*; $a \in \Sigma\ n$⟧ ⟹
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*: ⟦*wf n r*; *wf-word n w1*; *wf-word n w2*⟧ ⟹
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* ⟹ *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*) ⟷ *w1* @ *rev w2* ∈ *lang n r*
using *lang-biderivs*[*OF assms*] *lang-final*[*of biderivs w1 w2 r n*] **by** *auto*

lemma *ACI-norm-biderivs*: ≪*biderivs w1 w2* ≪*r*≫≫ = ≪*biderivs w1 w2 r*≫
unfolding *biderivs-def* **by** (*metis ACI-norm-lderivs ACI-norm-rderivs o-apply*)

lemma *finite* {≪*biderivs w1 w2 r*≫ | *w1 w2* . *True*}
proof −
have {≪*biderivs w1 w2 r*≫ | *w1 w2* . *True*} = ($\bigcup$ *s* ∈ {≪*lderivs as r*≫ | *as* . *True*}. {≪*rderivs bs s*≫ | *bs* . *True*})
unfolding *biderivs-def* **by** (*fastforce simp*: *ACI-norm-rderivs*)
also have *finite* ... **by** (*rule iffD2*[*OF finite-UN*[*OF finite-lderivs*] *ballI*[*OF finite-rderivs*]])
finally show *?thesis* **.**
qed

end

5.2 Quotioning by the same letter

definition *fin-cutSame x xs* = *take* (*LEAST n*. *drop n xs* = *replicate* (*length xs* − *n*) *x*) *xs*

lemma *fin-cutSame-Nil*[*simp*]: *fin-cutSame x* [] = []
unfolding *fin-cutSame-def* **by** *simp*

lemma *Least-fin-cutSame*: (*LEAST n. drop n xs* = *replicate* (*length xs* − *n*) *y*) =
length xs − *length* (*takeWhile* (λ*x*. *x* = *y*) (*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 *length xs* − *m* ≤ *length* (*takeWhile* (λ*x*. *x* = *y*) (*rev xs*))
proof (*intro length-takeWhile-less-P-nth*)
fix *i* **assume** *i* < *length xs* − *m*
hence *rev xs* ! *i* ∈ *set* (*drop m xs*)
by (*induct xs arbitrary*: *i rule*: *rev-induct*) (*auto simp*: *nth-Cons'*)
with ‹*?P m*› **show** *rev xs* ! *i* = *y* **by** *simp*
qed *simp*
thus *?min* ≤ *m* **by** *linarith*
qed

lemma *takeWhile-takes-all*: *length xs* = *m* ⟹ *m* ≤ *length* (*takeWhile P xs*) ⟷
Ball (*set xs*) *P*
by *hypsubst* (*induct xs*, *auto*)

lemma *fin-cutSame-Cons*[*simp*]: *fin-cutSame x* (*y* # *xs*) =
(*if fin-cutSame x xs* = [] *then if x* = *y then* [] *else* [*y*] *else y* # *fin-cutSame x xs*)
unfolding *fin-cutSame-def Least-fin-cutSame*
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-takeWhile-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-takeWhile-le take-Suc-Cons*)
done

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

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

lemma *fin-cutSameE*: *fin-cutSame x xs* = *ys* $\Longrightarrow$ $\exists$ *m*. *xs* = *ys* @ *replicate m x*
  by (*induct xs arbitrary*: *ys*) (*auto*, *metis replicate-Suc*)

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

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

context *embed*
begin

lemma *finite-rderivs-same*: *finite* {«*rderivs* (*replicate m a*) *r*» | *m* . *True*}
  by (*auto intro*: *finite-subset*[*OF* - *finite-rderivs*])

lemma *wf-word-replicate*[*simp*]: *a* $\in$ Σ *n* $\Longrightarrow$ *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* $\in$ *star* {[*x*]}
  thus *xs* $\in$ {*replicate m x* | *m* . *True*} **by** (*induct xs*) (*auto*, *metis replicate-Suc*)
qed (*auto intro*: *Ball-starI*)

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

lemma *wf-samequot*: ⟦*wf n r*; *a* $\in$ Σ *n*⟧ $\Longrightarrow$ *wf n* (*samequot a r*)
  unfolding *samequot-def wf.simps wf-flatten-PLUS*[*OF finite-rderivs-same*] **by** *auto*

lemma *lang-samequot*: ⟦*wf n r*; *a* $\in$ Σ *n*⟧ $\Longrightarrow$
  lang n (*samequot a r*) = *SAMEQUOT a* (*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-cutSameE*[*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-cutSame-replicate*
apply (*erule trans*)
unfolding *fin-cutSame-replicate*
apply (*rule refl*)
done

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

definition *invar-rderiv-and-add as r brs* ≡
(*if fst brs then True else* ≪*rderiv as* (*hd* (*snd brs*))≫ ∈ *set* (*snd brs*)) ∧
snd brs ≠ [] ∧ *distinct* (*snd brs*) ∧
(∀ *i* < *length* (*snd brs*). *snd brs* ! *i* = ≪*rderivs* (*replicate* (*length* (*snd brs*) − *1* − *i*) *as*) *r*≫)

lemma *invar-rderiv-and-add-init*: *invar-rderiv-and-add as r* (*True*, [≪*r*≫])
unfolding *invar-rderiv-and-add-def* **by** *auto*

lemma *invar-rderiv-and-add-step*: *invar-rderiv-and-add as r brs* ⟹ *fst brs* ⟹
invar-rderiv-and-add as r (*rderiv-and-add as brs*)
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*: ⟦≪*rderivs* (*replicate i as*) *r*≫ = ≪*r*≫; *i* > *0*⟧ ⟹
≪*rderivs* (*replicate* (*m* ∗ *i*) *as*) *r*≫ = ≪*r*≫
proof (*induct m arbitrary*: *r*)
case (*Suc m*)
hence ≪*rderivs* (*replicate* (*m* ∗ *i*) *as*) ≪*rderivs* (*replicate i as*) *r*≫≫ = ≪*r*≫
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 ≪*rderivs* (*replicate i as*) *r*≫ = ≪*r*≫ *k* < *i*
shows ≪*rderivs* (*replicate* (*m* ∗ *i* + *k*) *as*) *r*≫ = ≪*rderivs* (*replicate k as*) *r*≫ (**is** *?L* = *?R*)
proof −
have *?L* = ≪*rderivs* (*replicate k as*) ≪*rderivs* (*replicate* (*m* ∗ *i*) *as*) *r*≫≫
by (*simp add*: *ACI-norm-rderivs replicate-add rderivs-append*)
also have ≪*rderivs* (*replicate* (*m* ∗ *i*) *as*) *r*≫ = ≪*r*≫ **using** *assms*
by (*simp add*: *rderivs-replicate-mult*)

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

lemma *rderivs-replicate-mod*:
assumes $\ll$*rderivs* (*replicate i as*) *r*$\gg$ = $\ll$*r*$\gg$ *i* > *0*
shows $\ll$*rderivs* (*replicate m as*) *r*$\gg$ = $\ll$*rderivs* (*replicate* (*m mod i*) *as*) *r*$\gg$ (**is** *?L* = *?R*)
by (*subst mod-div-equality*[*symmetric*, *of m i*])
(*intro rderivs-replicate-mult-rest*[*OF assms*(*1*)] *mod-less-divisor*[*OF assms*(*2*)])

lemma *rderivs-replicate-diff*: $[\![$$\ll$*rderivs* (*replicate i as*) *r*$\gg$ = $\ll$*rderivs* (*replicate j as*) *r*$\gg$; *i* > *j*$]\!] \Longrightarrow$
$\ll$*rderivs* (*replicate* (*i* − *j*) *as*) (*rderivs* (*replicate j as*) *r*)$\gg$ = $\ll$*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*, [$\ll$*r*$\gg$]) = *Some* (*b*, *rs*)
shows *wf n* (*PLUS rs*)
proof −
have ¬ *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*, [$\ll$*r*$\gg$]) = *Some* (*b*, *rs*)
shows *lang n* (*PLUS rs*) = *UNION* {$\ll$*rderivs* (*replicate m as*) *r*$\gg$ |*m*. *True*} (*lang n*)
proof −
have ¬ *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*: *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* = «*rderivs* (*replicate m as*) *r*»
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** «*rderivs* (*replicate m as*) *r*» = «*rderivs* (*replicate m' as*) (*rderivs ?x r*)»
unfolding *replicate-add*[*symmetric*] *rderivs-append*[*symmetric*] **by** (*simp add*: *nat-add-commute*)
also from *cyc* **have** ... = «*rderivs* (*replicate* (*m' mod* (*Suc i*)) *as*) (*rderivs ?x r*)»
by (*elim rderivs-replicate-mod*) *simp*
also from *i* **have** ... = «*rderivs* (*replicate* (*m' mod* (*Suc i*) + *length rs* − *Suc i*) *as*) *r*»
unfolding *rderivs-append*[*symmetric*] *replicate-add*[*symmetric*] **by** (*simp add*: *nat-add-commute*)
also from *m i* **have** ... = «*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*: *nat-add-commute*)
finally have $\exists$ *j* < *length rs*. «*rderivs* (*replicate m as*) *r*» = «*rderivs* (*replicate* (*length rs* − *Suc j*) *as*) *r*»
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* < *length rs*. *m* = *length rs* − *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 *UNION* {«*rderivs* (*replicate m as*) *r*» |*m*. *True*} (*lang n*) ⊆ *lang n* (*PLUS rs*)
by (*fastforce simp*: *in-set-conv-nth lang-PLUS intro*!: *bexI*[*rotated*])
moreover from ∗ **have** *lang n* (*PLUS rs*) ⊆ *UNION* {«*rderivs* (*replicate m as*) *r*» |*m*. *True*} (*lang n*)
unfolding *invar-rderiv-and-add-def* **by** (*fastforce simp*: *in-set-conv-nth lang-PLUS*)
ultimately show *lang n* (*PLUS rs*) = *UNION* {«*rderivs* (*replicate m as*) *r*» |*m*. *True*} (*lang n*) **by** *blast*
qed

lemma *length-subset-card*: ⟦*finite X*; *distinct* (*x* # *xs*); *set* (*x* # *xs*) ⊆ *X*⟧ ⟹ *length xs* < *card X*
by (*metis card-mono distinct-card impossible-Cons not-leE order-trans*)

lemma *samequot-termination*:
assumes *while-option fst* (*rderiv-and-add as*) (*True*, [≪*r*≫]) = *None* (**is** *?cl* = *None*)
shows *False*
proof −
let *?D* = {≪*rderivs* (*replicate m as*) *r*≫ | *m* . *True*}
let *?f* = λ(*b*, *rs*). *card ?D* + *1* − *length rs* + (*if b then 1 else 0*)
have ∃ *st*. *?cl* = *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 *fastforce*
apply *fastforce*
apply (*metis Suc-less-eq nth-Cons-Suc*)
done
with *assms* **show** *False* **by** *auto*
qed

definition *samequot-exec as r* =
Times (*PLUS* (*snd* (*the* (*while-option fst* (*rderiv-and-add as*) (*True*, [≪*r*≫]))))) (*Star* (*Atom as*))

lemma *wf-samequot-exec*: ⟦*wf n r*; *as* ∈ Σ *n*⟧ ⟹ *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*:
⟦*wf n r*; *as* ∈ Σ *n*⟧ ⟹ *lang n* (*samequot-exec as r*) = *SAMEQUOT as* (*lang n r*)
unfolding *samequot-exec-samequot* **by** (*rule lang-samequot*)

end

5.3 Suffix Prefix Languages

definition *Suffix* :: *'a lang* ⇒ *'a lang* **where**

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

definition *Prefix* :: *'a lang* $\Rightarrow$ *'a lang* **where**
Prefix $L = \{w.\ \exists\, u.\ w\ @\ u \in L\}$

lemma *Prefix-Suffix*: *Prefix* L = *rev* ‘ *Suffix* (*rev* ‘ L)
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 $L = \{x\ .\ \exists\, n > 0.\ x\ \hat{}\hat{}\ n \in L\}$

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* :: *nat*
begin

definition *SUFFIX* :: *'a rexp* $\Rightarrow$ *'a rexp* **where**
SUFFIX r = *flatten PLUS* {≪*lderivs w r*≫| *w*. *wf-word n w*}

lemma *finite-lderivs-wf*: *finite* {≪*lderivs w r*≫| *w*. *wf-word n w*}
by (*auto intro*: *finite-subset*[*OF - finite-lderivs*])

definition *PREFIX* :: *'a rexp* $\Rightarrow$ *'a rexp* **where**
PREFIX r = *REV* (*SUFFIX* (*REV r*))

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

lemma *lang-SUFFIX*[*simp*]: *wf n r* $\Longrightarrow$ *lang n* (*SUFFIX r*) = *Suffix* (*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*]: *wf n r* $\Longrightarrow$ *wf n* (*PREFIX r*)
unfolding *PREFIX-def* **by** *auto*

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

end

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

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

end

end

6 Monadic Second-Order Logic Formulas

6.1 Interpretations and Encodings

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

abbreviation *enc-atom-bool I n* $\equiv$ *map* (λx. *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*)

6.2 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* φ_1 φ_2) = *FOV* φ_1 $\cup$ *FOV* φ_2
| *FOV* (*FAnd* φ_1 φ_2) = *FOV* φ_1 $\cup$ *FOV* φ_2
| *FOV* (*FExists* φ) = (λx. $x - 1$) ` (*FOV* φ $-$ $\{0\}$)
| *FOV* (*FEXISTS* φ) = (λx. $x - 1$) ` *FOV* φ

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

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

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

locale *formula* = *embed set o* $(\sigma\ \Sigma)\ \pi\ \varepsilon\ \Sigma$ **for** $\Sigma :: 'a :: linorder\ list$ +
assumes *nonempty*: $\Sigma \neq []$
begin

abbreviation Σ-*combinatorial-product n* $\equiv$
List.maps (λ*bools. map* (λ*a.* (*a, bools*)) Σ) (*bool-combinatorial-product n*)

primrec *pre-wf-formula* :: *nat* $\Rightarrow$ *'a formula* $\Rightarrow$ *bool* **where**
pre-wf-formula n $(FQ\ a\ m) = (a \in set\ \Sigma \wedge m < n)$
| *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* φ
| *pre-wf-formula n* $(FOr\ \varphi_1\ \varphi_2)$ = (*pre-wf-formula n* $\varphi_1 \wedge$ *pre-wf-formula n* φ_2)
| *pre-wf-formula n* $(FAnd\ \varphi_1\ \varphi_2)$ = (*pre-wf-formula n* $\varphi_1 \wedge$ *pre-wf-formula n* φ_2)
| *pre-wf-formula n* $(FExists\ \varphi)$ = (*pre-wf-formula* $(n + 1)\ \varphi \wedge 0 \in FOV\ \varphi \wedge 0 \notin SOV\ \varphi$)
| *pre-wf-formula n* $(FEXISTS\ \varphi)$ = (*pre-wf-formula* $(n + 1)\ \varphi \wedge 0 \notin FOV\ \varphi \wedge 0 \in SOV\ \varphi$)

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

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

lemma *max-idx-vars*: *pre-wf-formula n* $\varphi \Longrightarrow \forall p \in FOV\ \varphi \cup SOV\ \varphi.\ p < n$
by (*induct* φ *arbitrary*: *n*)
(*auto split*: *split-if-asm*, (*metis Un-iff diff-Suc-less less-SucE less-imp-diff-less*)+)

lemma *finite-FOV*: *finite* (*FOV* φ)
by (*induct* φ) (*auto split*: *split-if-asm*)

6.3 ENC

definition *arbitrary-except n pbs xs* $\equiv$
PLUS (*map* (λ*bs. PLUS* (*map* (λ*x. Atom* (*x, fold* (λ(*p, b*). *insert-nth p b*) *pbs*

bs)) *xs*))
  (*bool-combinatorial-product* (n − *length pbs*)))

lemma *wf-rexp-arbitrary-except*:
 ⟦*length pbs* ≤ *n*; *set xs* ⊆ *set* Σ⟧ ⟹ *wf n* (*arbitrary-except n pbs xs*)
 by (*auto simp*: *arbitrary-except-def*) (*force simp*: *σ-def set-n-lists length-fold-insert-nth*)

definition *valid-ENC* :: *nat* ⇒ *nat* ⇒ ('*a* × *bool list*) *rexp* **where**
 valid-ENC n p = (*if n* = *0 then Regular-Exp.Not Zero else*
  TIMES [
   Star (*arbitrary-except n* [(*p*, *False*)] Σ),
   arbitrary-except n [(*p*, *True*)] Σ,
   Star (*arbitrary-except n* [(*p*, *False*)] Σ)])

lemma *wf-rexp-valid-ENC*: *wf n* (*valid-ENC n p*)
 unfolding *valid-ENC-def* **by** (*auto intro*!: *wf-rexp-arbitrary-except*)

definition *ENC* :: *nat* ⇒ '*a formula* ⇒ ('*a* × *bool list*) *rexp* **where**
 ENC n φ = *flatten INTERSECT* (*valid-ENC n* ' *FOV* φ)

lemma *wf-rexp-ENC*: *wf n* (*ENC n* φ)
 unfolding *ENC-def*
 by (*intro iffD2*[*OF wf-flatten-INTERSECT*]) (*auto intro*: *finite-FOV simp*: *wf-rexp-valid-ENC*)

lemma *enc-atom-σ-eq*: *i* < *length w* ⟹
 (*length I* = *n* ∧ *w* ! *i* ∈ *set* Σ) ⟷ *enc-atom I i* (*w* ! *i*) ∈ *set* (σ Σ *n*)
 by (*auto simp*: *σ-def set-n-lists intro*!: *exI*[*of - enc-atom-bool I i*] *imageI*)

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

lemma *enc-atom-bool-take-drop-True*:
 ⟦*r* < *length I*; *case I* ! *r of Inl p*' ⇒ *p* = *p*' | *Inr P* ⇒ *p* ∈ *P*⟧ ⟹
  enc-atom-bool I p = *take r* (*enc-atom-bool I p*) @ *True* # *drop* (*Suc r*)
(*enc-atom-bool I p*)
 by (*auto intro*: *trans*[*OF id-take-nth-drop*])

lemma *enc-atom-bool-take-drop-True2*:
 ⟦*r* < *length I*; *case I* ! *r of Inl p*' ⇒ *p* = *p*' | *Inr P* ⇒ *p* ∈ *P*;
  s < *length I*; *case I* ! *s of Inl p*' ⇒ *p* = *p*' | *Inr P* ⇒ *p* ∈ *P*; *r* < *s*⟧ ⟹
  enc-atom-bool I p = *take r* (*enc-atom-bool I p*) @ *True* #
   take (*s* − *Suc r*) (*drop* (*Suc r*) (*enc-atom-bool I p*)) @ *True* #
   drop (*Suc s*) (*enc-atom-bool I 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*:
 ⟦*r* < *length I*; *case I* ! *r of Inl p*' ⇒ *p* ≠ *p*' | *Inr P* ⇒ *p* ∉ *P*⟧ ⟹
  enc-atom-bool I p = *take r* (*enc-atom-bool I p*) @ *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-arbitrary-except-True*: $[\![r < length\ I;$
  $case\ I\ !\ r\ of\ Inl\ p' \Rightarrow p = p' \mid Inr\ P \Rightarrow p \in P;\ length\ I = n;\ a \in set\ xs;\ set\ xs \subseteq set\ \Sigma]\!] \Longrightarrow$
  $[enc\text{-}atom\ I\ p\ a] \in lang\ n\ (arbitrary\text{-}except\ n\ [(r,\ True)]\ xs)$
  unfolding *arbitrary-except-def*
  by (*auto intro*!: *bexI*[*of* - ($\lambda J.$ *take r J @ drop* $(r + 1)$ *J*) (*enc-atom-bool I p*)]
    intro: *enc-atom-bool-take-drop-True*)

lemma *enc-atom-lang-arbitrary-except-True2*:$[\![r < length\ I;\ s < length\ I;\ r < s;$
  $case\ I\ !\ r\ of\ Inl\ p' \Rightarrow p = p' \mid Inr\ P \Rightarrow p \in P;$
  $case\ I\ !\ s\ of\ Inl\ p' \Rightarrow p = p' \mid Inr\ P \Rightarrow p \in P;\ length\ I = n;\ a \in set\ \Sigma]\!] \Longrightarrow$
  $[enc\text{-}atom\ I\ p\ a] \in lang\ n\ (arbitrary\text{-}except\ n\ [(r,\ True),\ (s,\ True)]\ \Sigma)$
  unfolding *arbitrary-except-def*
  by (*auto intro*!: *bexI*[*of* - ($\lambda J.$ *take r J @ take* $(s - r - 1)$ (*drop* $(r + 1)$ *J*) @ *drop* $(s - r)$ (*drop* $(r + 1)$ *J*)) (*enc-atom-bool I p*)]
    intro!: *enc-atom-bool-take-drop-True2 simp*: *min-absorb2 take-Cons′ drop-Cons′*)

lemma *enc-atom-lang-arbitrary-except-False*:$[\![r < length\ I;$
  $case\ I\ !\ r\ of\ Inl\ p' \Rightarrow p \neq p' \mid Inr\ P \Rightarrow p \notin P;\ length\ I = n;\ a \in set\ xs;\ set\ xs \subseteq set\ \Sigma]\!] \Longrightarrow$
  $[enc\text{-}atom\ I\ p\ a] \in lang\ n\ (arbitrary\text{-}except\ n\ [(r,\ False)]\ xs)$
  unfolding *arbitrary-except-def*
  by (*auto intro*!: *bexI*[*of* - ($\lambda J.$ *take r J @ drop* $(r + 1)$ *J*) (*enc-atom-bool I p*)]
    intro: *enc-atom-bool-take-drop-False*)

lemma *arbitrary-except*:
  $[\![v \in lang\ n\ (arbitrary\text{-}except\ n\ [(r,\ b)]\ S);\ r < n;\ set\ S \subseteq set\ \Sigma]\!] \Longrightarrow$
    $\exists x.\ v = [x] \wedge snd\ x\ !\ r = b \wedge fst\ x \in set\ S$
  unfolding *arbitrary-except-def* **by** (*auto simp*: *nth-append*)

lemma *arbitrary-except2*:
  $[\![v \in lang\ n\ (arbitrary\text{-}except\ n\ [(r,\ b),\ (s,\ b')]\ S);\ r < s;\ r < n;\ s < n;\ set\ S \subseteq set\ \Sigma]\!] \Longrightarrow$
    $\exists x.\ v = [x] \wedge snd\ x\ !\ r = b \wedge snd\ x\ !\ s = b' \wedge fst\ x \in set\ S$
  unfolding *arbitrary-except-def* **by** (*auto simp*: *nth-append min-absorb2*)

lemma *star-arbitrary-except*:
  $[\![v \in star\ (lang\ n\ (arbitrary\text{-}except\ n\ [(r,\ b)]\ \Sigma));\ r < n;\ i < length\ v]\!] \Longrightarrow$
    $snd\ (v\ !\ i)\ !\ r = b$
proof (*induct arbitrary*: *i rule*: *star-induct*)
  case (*append u v*) **thus** *?case* **by** (*cases i*) (*auto dest*: *arbitrary-except*)
qed *simp*

end

end

7 M2L

7.1 Encodings

context *formula*
begin

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

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

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

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

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

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

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

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

lemma *positions-in-row*: *positions-in-row w i* = {*p*. *p* < *length w* ∧ *snd* (*w* ! *p*) ! *i*}
unfolding *positions-in-row-def these-def* **by** (*auto intro*!: *image-eqI*[*of* - *the*])

lemma *positions-in-row-unique*: ∃!*p*. *p* < *length w* ∧ *snd* (*w* ! *p*) ! *i* ⟹ *the-elem* (*positions-in-row w i*) = (*THE p*. *p* < *length w* ∧ *snd* (*w* ! *p*) ! *i*)
by (*rule the1I2*) (*auto simp*: *the-elem-def positions-in-row*)

lemma *positions-in-row-length*: ∃!*p*. *p* < *length w* ∧ *snd* (*w* ! *p*) ! *i* ⟹ *the-elem* (*positions-in-row w i*) < *length w*
unfolding *positions-in-row-unique* **by** (*rule the1I2*) *auto*

lemma *dec-interp-Inl*: ⟦*i* ∈ *FO*; *i* < *n*⟧ ⟹ ∃ *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*: ⟦*dec-interp n FO x* ! *i* = *Inr P*; *i* ∈ *FO*; *i* < *n*⟧ ⟹ *False*
unfolding *dec-interp-def* **using** *nth-map*[*of n* [*0*..<*n*]] **by** *auto*

lemma *dec-interp-Inr*: ⟦*i* ∉ *FO*; *i* < *n*⟧ ⟹ ∃ *P*. *dec-interp n FO x* ! *i* = *Inr P*
unfolding *dec-interp-def* **using** *nth-map*[*of n* [*0*..<*n*]] **by** *auto*

lemma *dec-interp-not-Inl*: ⟦*dec-interp n FO x* ! *i* = *Inl p*; *i* ∉ *FO*; *i* < *n*⟧ ⟹ *False*
unfolding *dec-interp-def* **using** *nth-map*[*of n* [*0*..<*n*]] **by** *auto*

lemma *Inl-dec-interp-length*:
assumes ∀ *i* ∈ *FO*. ∃!*p*. *p* < *length w* ∧ *snd* (*w* ! *p*) ! *i*
shows *Inl p* ∈ *set* (*dec-interp n FO w*) ⟹ *p* < *length w*
unfolding *dec-interp-def* **by** (*auto intro*: *positions-in-row-length*[*OF bspec*[*OF assms*]])

lemma *Inr-dec-interp-length*: ⟦*Inr P* ∈ *set* (*dec-interp n FO w*); *p* ∈ *P*⟧ ⟹ *p* < *length w*
unfolding *dec-interp-def* **by** (*auto simp*: *positions-in-row*)

lemma *enc-atom-dec*:
⟦*wf-word n w*; ∀ *i* ∈ *FO*. *i* < *n* ⟶ (∃!*p*. *p* < *length w* ∧ *snd* (*w* ! *p*) ! *i*); *p* < *length w*⟧ ⟹
enc-atom (*dec-interp n FO w*) *p* (*fst* (*w* ! *p*)) = *w* ! *p*
unfolding *wf-word dec-interp-def map-filter-def Let-def*
apply (*auto simp*: *comp-def intro*!: *trans*[*OF iffD2*[*OF Pair-eq*] *pair-collapse*])
apply (*intro nth-equalityI*)
apply (*auto simp add*: *σ-def set-n-lists dest*!: *nth-mem*) []
apply (*auto simp*: *positions-in-row*)
apply (*drule bspec*)
apply *assumption*
apply (*drule mp*)
apply *assumption*

apply (*frule positions-in-row-unique*)
apply (*erule notE*)
apply (*simp add*: *positions-in-row*)
apply (*erule the1I2*)
apply *simp*

apply (*drule bspec*)
apply *assumption*
apply (*drule mp*)
apply *assumption*
apply (*frule positions-in-row-unique*)
apply (*simp add*: *positions-in-row*)
apply (*rule the-equality*[*symmetric*])
apply *auto*
done

lemma *enc-dec*:
$[\![$*wf-word n w*; $\forall\, i \in FO.\ i < n \longrightarrow (\exists!p.\ p <$ *length w* $\wedge$ *snd* $(w\ !\ p)\ !\ i)]\!] \Longrightarrow$
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*: *dec-word* (*enc* (*w*, *I*)) $=$ *w*
unfolding *dec-word-def* **by** *auto*

lemma *enc-unique*:
assumes *wf-interp w I i* $<$ *length I*
shows $\exists\, p.\ I\ !\ i =$ *Inl p* $\Longrightarrow \exists!p.\ p <$ *length* (*enc* (*w*, *I*)) $\wedge$ *snd* (*enc* (*w*, *I*) ! *p*) ! *i*
proof (*erule exE*)
fix *p* **assume** *I* ! *i* $=$ *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*:
$[\![$*dec-interp n FO* (*enc* (*w*, *I*)) ! *i* $=$ *Inl p'*; *I* ! *i* $=$ *Inl p*; $i \in FO$; $i < n$; *length I* $= n$; $p <$ *length w*; *wf-interp w I*$]\!] \Longrightarrow$
$p = p'$
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*:
$[\![$*dec-interp n FO* (*enc* (*w*, *I*)) ! *i* $=$ *Inr P'*; *I* ! *i* $=$ *Inr P*; $i \notin FO$; $i < n$; *length I* $= n$; $\forall\, p \in P.\ p <$ *length w*$]\!] \Longrightarrow$
$P = P'$
unfolding *dec-interp-def positions-in-row* **by** *auto*

lemma *lang-ENC*:
assumes *wf-formula n* φ
shows *lang n* (*ENC n* φ) − {[]} = {*enc* (*w*, *I*) | *w I* . *length I* = *n* ∧ *wf-interp-for-formula* (*w*, *I*) φ}
(**is** *?L* = *?R*)
proof (*cases FOV* φ = {})
case *True* **with** *assms* **show** *?thesis*
apply (*auto simp*: *ENC-def intro*!: *enc-atom-*σ)
apply (*intro exI conjI*)
apply (*rule trans*[*OF sym*[*OF enc-dec*[*of* - - {}]] *enc.simps*])
apply (*auto simp add*: *wf-word dec-word-def dec-interp-def*)
apply (*auto simp*: σ*-def positions-in-row set-n-lists split*: *sum.splits*)
apply (*drule trans*[*OF sym nth-map*])
apply (*auto intro*: *bspec*[*OF max-idx-vars*])
done
next
case *False*
hence *nonempty*: *valid-ENC n* ‘ *FOV* φ ≠ {} **by** *simp*
have *finite*: *finite* (*valid-ENC n* ‘ *FOV* φ) **by** (*rule finite-imageI*[*OF finite-FOV*])
from *False assms*(*1*) **have** *0* < *n* **by** (*cases n*) (*auto split*: *dest*!: *max-idx-vars*)
with *wf-rexp-valid-ENC* **have** *wf-rexp*: ∀ *x* ∈ *valid-ENC n* ‘ *FOV* φ. *wf n x* **by** *auto*
{ **fix** *r w I* **assume** *r* ∈ *FOV* φ **and** ∗: *length I* = *n wf-interp-for-formula* (*w*, *I*) φ
then obtain *p* **where** *p*: *I* ! *r* = *Inl p* **by** (*cases I* ! *r*) *auto*
moreover from ‹*r* ∈ *FOV* φ› *assms* ∗(*1*) **have** *r* < *length I* **by** (*auto dest*!: *max-idx-vars*)
ultimately have *Inl p* ∈ *set I* **by** (*auto simp add*: *in-set-conv-nth*)
with ∗(*2*) **have** *p* < *length w* **by** *auto*
with ∗(*2*) **obtain** *a* **where** *a*: *w* ! *p* = *a a* ∈ *set* Σ **by** *auto*
from ‹*r* < *length I*› *p* ∗(*1*) ‹*a* ∈ *set* Σ›
have [*enc-atom I p a*] ∈ *lang n* (*arbitrary-except n* [(*r*, *True*)] Σ)
by (*intro enc-atom-lang-arbitrary-except-True*[*OF* - - - - *subset-refl*]) *auto*
moreover
from ‹*r* < *length I*› *p* ∗(*1*) ‹*a* ∈ *set* Σ› ∗(*2*) ‹*p* < *length w*›
have *take p* (*enc* (*w*, *I*)) ∈ *star* (*lang n* (*arbitrary-except n* [(*r*, *False*)] Σ))
by (*auto simp*: *in-set-conv-nth intro*!: *Ball-starI enc-atom-lang-arbitrary-except-False*) *auto*
moreover
from ‹*r* < *length I*› *p* ∗(*1*) ‹*a* ∈ *set* Σ› ∗(*2*) ‹*p* < *length w*›
have *drop* (*Suc p*) (*enc* (*w*, *I*)) ∈ *star* (*lang n* (*arbitrary-except n* [(*r*, *False*)] Σ))
by (*auto simp*: *in-set-conv-nth 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*)) ∈
lang n (*valid-ENC n r*) **using** ‹*0* < *n*› **unfolding** *valid-ENC-def* **by** (*auto simp del*: *append.simps*)

with ‹p < *length w*› *a* **have** *enc* (*w*, *I*) ∈ *lang n* (*valid-ENC n r*)
using *id-take-nth-drop*[*of p enc* (*w*, *I*)] **by** *auto*
}
hence *?R* ⊆ *?L* **using** *lang-flatten-INTERSECT*[*OF finite nonempty wf-rexp*] **by** (*auto simp add*: *ENC-def*)
moreover
{ fix *x* **assume** *x* ∈ (⋂ *r* ∈ *valid-ENC n* ‘ *FOV φ*. *lang n r*)
hence *r*: ∀ *r* ∈ *FOV φ*. *x* ∈ *lang n* (*valid-ENC n r*) **by** *blast*
have *length* (*dec-interp n* (*FOV φ*) *x*) = *n* **unfolding** *dec-interp-def* **by** *simp*
moreover
{ fix *r* **assume** *r* ∈ *FOV φ*
with *assms* **have** *r* < *n* **using** *max-idx-vars*[*of n φ*] **by** *auto*
from ‹*r* ∈ *FOV φ*› *r* **obtain** *u v w* **where** *uvw*: *x* = *u* @ *v* @ *w*
u ∈ *star* (*lang n* (*arbitrary-except n* [(*r*, *False*)] Σ))
v ∈ *lang n* (*arbitrary-except n* [(*r*, *True*)] Σ)
w ∈ *star* (*lang n* (*arbitrary-except n* [(*r*, *False*)] Σ)) **using** ‹*0* < *n*› **unfolding** *valid-ENC-def* **by** *fastforce*
hence *length u* < *length x* ⋀*i*. *i* < *length x* ⟶ *snd* (*x* ! *i*) ! *r* ⟷ *i* = *length u*
by (*auto simp*: *nth-append nth-Cons′ split*: *split-if-asm*
dest!: *arbitrary-except*[*OF* - ‹*r* < *n*›]
dest: *star-arbitrary-except*[*OF* - ‹*r* < *n*›, *of u*]
elim!: *iffD1*[*OF star-arbitrary-except*[*OF* - ‹*r* < *n*›, *of w False*]]) *auto*
hence ∃!*p*. *p* < *length x* ∧ *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* **by** *auto*
with ∗ **have** *x* = *enc* (*dec-word x*, *dec-interp n* (*FOV φ*) *x*) **by** (*intro sym*[*OF enc-dec*]) *auto*
moreover
from ∗ **have** *wf-interp-for-formula* (*dec-word x*, *dec-interp n* (*FOV φ*) *x*) *φ*
using *r False x-wf-word*[*unfolded wf-word*, *unfolded σ-def*] *assms*
apply (*auto simp*: *dec-word-def split*: *sum.splits*)
apply *fastforce*
using *Inl-dec-interp-length*[*OF ballI*] **apply** *blast*
using *Inr-dec-interp-length* **apply** *blast*
using *dec-interp-Inl*[*OF* - *bspec*[*OF max-idx-vars*], *of* - *FOV φ n φ x*] **apply** *force*
using *dec-interp-Inr*[*OF* - *bspec*[*OF max-idx-vars*], *of* - *FOV φ n φ x*] **apply** *fastforce*
done
ultimately have *x* ∈ {*enc* (*w*, *I*) | *w I*. *length I* = *n* ∧ *wf-interp-for-formula* (*w*, *I*) *φ*} **by** *blast*
}
with *False lang-flatten-INTERSECT*[*OF finite nonempty wf-rexp*] **have** *?L* ⊆ *?R* **by** (*auto simp add*: *ENC-def*)
ultimately show *?thesis* **by** *blast*
qed

7.2 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*:
$[\![$*wf-formula* (*length* I) (*FExists* φ); $w \neq [\,]]\!] \Longrightarrow$
wf-interp-for-formula $(w,\ I)$ (*FExists* φ) $\longleftrightarrow (\forall\, p <$ *length* w. *wf-interp-for-formula* $(w,$ *Inl* $p\ \#\ I)\ \varphi)$
apply (*clarsimp split*: *sum.splits split-if-asm*)
apply *safe*
apply (*metis* (*hide-lams*) *DiffI Suc-pred gr0I nth-Cons-0 nth-Cons-Suc singleton-iff sum.simps*(*4*))
apply (*metis diff-Suc-Suc diff-zero nat.exhaust nth-Cons-Suc*)
apply (*metis length-greater-0-conv*)
apply (*metis length-greater-0-conv*)
apply (*metis length-greater-0-conv*)
apply (*metis diff-Suc-Suc gr0-implies-Suc length-greater-0-conv minus-nat.diff-0 nth-Cons-Suc*)
apply (*auto simp*: *nth-Cons'* *split*: *split-if-asm*)
done

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*: *split-if-asm*)

lemma *wf-interp-for-formula-FEXISTS*:
$[\![$*wf-formula* (*length* I) (*FEXISTS* φ); $w \neq [\,]]\!] \Longrightarrow$
wf-interp-for-formula $(w,\ I)$ (*FEXISTS* φ) $\longleftrightarrow (\forall\, P \subseteq \{0\ ..\ $*length* $w - 1\}$. *wf-interp-for-formula* $(w,$ *Inr* $P\ \#\ I)\ \varphi)$
apply (*clarsimp split*: *sum.splits split-if-asm*)
apply *safe*
apply (*cases* w)
apply *auto* [*2*]
apply (*metis Suc-pred gr0I nth-Cons-Suc*)
apply (*metis* (*hide-lams*) *DiffI Suc-pred gr0I nth-Cons-0 nth-Cons-Suc singleton-iff sum.simps*(*4*))
unfolding *sym*[*OF length-greater-0-conv*] *nth-Cons'* *One-nat-def*
apply *auto* [*2*]
apply (*metis empty-subsetI*)
apply (*metis empty-subsetI*)
apply (*metis empty-subsetI neq0-conv*)
done

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*) φ
by (*cases w*) (*auto simp*: *nth-Cons′ split*: *sum.splits split-if-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* (λx. *case x of Inl p* $\Rightarrow$ *0* = *p* | *Inr P* $\Rightarrow$ *0* $\in$ *P*) *I*]
length-map[*of* (λx. *case x of Inl p* $\Rightarrow$ *0* = *p* | *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* φ*1* φ*2*) =
(*wf-interp-for-formula* (*w*, *I*) φ*1* $\wedge$ *wf-interp-for-formula* (*w*, *I*) φ*2*)
by *auto*

lemma *wf-interp-for-formula-FAnd*:
wf-interp-for-formula (*w*, *I*) (*FAnd* φ*1* φ*2*) =
(*wf-interp-for-formula* (*w*, *I*) φ*1* $\wedge$ *wf-interp-for-formula* (*w*, *I*) φ*2*)
by *auto*

lemma *enc-wf-interp*:
assumes *wf-formula* (*length I*) φ *wf-interp-for-formula* (*w*, *I*) φ
shows *wf-interp-for-formula* (*dec-word* (*enc* (*w*, *I*)), *dec-interp* (*length I*) (*FOV* φ) (*enc* (*w*, *I*))) φ
(**is** *wf-interp-for-formula* (-, *?dec*) φ)
unfolding *dec-word-enc*
proof −
{ fix *i* **assume** *i*: $i \in FOV\ \varphi$
with *assms*(*2*) **have** $\exists p.\ I\ !\ i = Inl\ p$ **by** (*cases I ! i*) *auto*
with *i assms* **have** $\exists!p.\ p < length\ (enc\ (w, I)) \wedge snd\ (enc\ (w, I)\ !\ p)\ !\ i$
by (*intro enc-unique*[*of w I i*]) (*auto intro*!: *bspec*[*OF max-idx-vars*] *split*: *sum.splits*)
} note $*$ = *this*
have $\forall x \in set\ ?dec.\ sum\text{-}case\ (\lambda p.\ p < length\ w)\ (\lambda P.\ \forall p{\in}P.\ p < length\ w)\ x$
proof (*intro ballI*, *split sum.split*, *safe*)
fix *p* **assume** *Inl p* $\in$ *set ?dec*
thus *p* < *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* < *length w* **using** *Inr-dec-interp-length* **by** *fastforce*
qed
thus *wf-interp-for-formula* (*w*, *?dec*) φ
using *assms*
dec-interp-Inl[*of - FOV* φ *length I enc* (*w*, *I*), *OF - bspec*[*OF max-idx-vars*]]

dec-interp-Inr[*of* - *FOV* φ *length I enc* (*w*, *I*), *OF* - *bspec*[*OF max-idx-vars*]]
by (*fastforce split*: *sum.splits*)
qed

lemma *enc-welldef*: $\llbracket$*enc* (*w*, *I*) = *enc* (*w*′, *I*′); *wf-formula* (*length I*) φ;
wf-interp-for-formula (*w*, *I*) φ; *wf-interp-for-formula* (*w*′, *I*′) $\varphi\rrbracket \Longrightarrow$
satisfies (*w*, *I*) $\varphi \longleftrightarrow$ *satisfies* (*w*′, *I*′) φ
proof (*induction* φ *arbitrary*: *I I*′)
case (*FQ a m*)
let *?dec* = λ*w I*. (*dec-word* (*enc* (*w*, *I*)), *dec-interp* (*length I*) (*FOV* (*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* ≠ [] **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* = λ*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* ≠ [] **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* = λ*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* φ*1* φ*2*) **show** *?case* **unfolding** *satisfies.simps*(*5*)
proof (*intro disj-cong*)
from *FOr*(*3*−*6*) **show** *satisfies* (w, I) φ*1* = *satisfies* (w', I') φ*1*
by (*intro FOr*(*1*)) *auto*
next
from *FOr*(*3*−*6*) **show** *satisfies* (w, I) φ*2* = *satisfies* (w', I') φ*2*
by (*intro FOr*(*2*)) *auto*
qed
next
case (*FAnd* φ*1* φ*2*) **show** *?case* **unfolding** *satisfies.simps*(*6*)
proof (*intro conj-cong*)
from *FAnd*(*3*−*6*) **show** *satisfies* (w, I) φ*1* = *satisfies* (w', I') φ*1*
by (*intro FAnd*(*1*)) *auto*
next
from *FAnd*(*3*−*6*) **show** *satisfies* (w, I) φ*2* = *satisfies* (w', I') φ*2*
by (*intro FAnd*(*2*)) *auto*
qed
next
case (*FExists* φ)
hence $w \neq []$ $w' \neq []$ **by** *auto*
hence *length*: *length w* = *length* w' *length I* = *length* I'
using *enc-word-length*[*OF FExists.prems*(*1*)] *enc-length*[*OF - FExists.prems*(*1*)] **by** *auto*
show *?case*
proof
assume *satisfies* (w, I) (*FExists* φ)
with *FExists.prems*(*3*) **obtain** *p* **where** *p* < *length w satisfies* (w, *Inl p* # *I*) φ
by (*auto intro*: *ord-less-eq-trans*[*OF le-imp-less-Suc Suc-pred*])
moreover
with *FExists.prems* **have** *satisfies* (w', *Inl p* # I') φ
apply (*intro iffD1*[*OF FExists.IH*[*of Inl p* # *I Inl p* # I']])
apply (*elim enc-extend-interp*)

apply (*auto split*: *sum.splits split-if-asm*) []
apply (*blast dest*!: *wf-interp-for-formula-FExists*[*OF* - ‹*w* ≠ []›])
apply (*blast dest*!: *wf-interp-for-formula-FExists*[*OF* - ‹*w'* ≠ []›, *of I'*, *unfolded length*[*symmetric*]])
apply *assumption*
done
ultimately show *satisfies* (*w'*, *I'*) (*FExists* φ) **using** *length* **by** (*auto intro*!: *exI*[*of* - *p*])
next
assume *satisfies* (*w'*, *I'*) (*FExists* φ)
with *FExists.prems*(*1*,*2*,*4*) **obtain** *p* **where** *p* < *length w'* *satisfies* (*w'*, *Inl p* # *I'*) φ
by (*auto intro*: *ord-less-eq-trans*[*OF le-imp-less-Suc Suc-pred*])
moreover
with *FExists.prems* **have** *satisfies* (*w*, *Inl p* # *I*) φ
apply (*intro iffD2*[*OF FExists.IH*[*of Inl p* # *I Inl p* # *I'*]])
apply (*elim enc-extend-interp*)
apply (*auto split*: *sum.splits split-if-asm*) []
apply (*blast dest*!: *wf-interp-for-formula-FExists*[*OF* - ‹*w* ≠ []›, *of I*, *unfolded length*(*1*)])
apply (*blast dest*!: *wf-interp-for-formula-FExists*[*OF* - ‹*w'* ≠ []›, *of I'*, *unfolded length*(*2*)[*symmetric*]])
apply *assumption*
done
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.prems*(*1*)] *enc-length*[*OF* - *FEXISTS.prems*(*1*)] **by** *auto*
show *?case*
proof
assume *satisfies* (*w*, *I*) (*FEXISTS* φ)
then obtain *P* **where** *P* ⊆ {*0* .. *length w* − *1*} *satisfies* (*w*, *Inr P* # *I*) φ **by** *auto*
moreover
with *FEXISTS.prems* **have** *satisfies* (*w'*, *Inr P* # *I'*) φ
apply (*intro iffD1*[*OF FEXISTS.IH*[*of Inr P* # *I Inr P* # *I'*]])
apply (*elim enc-extend-interp*)
apply (*auto split*: *sum.splits split-if-asm*) []
apply (*blast dest*!: *wf-interp-for-formula-FEXISTS*[*OF* - ‹*w* ≠ []›])
apply (*blast dest*!: *wf-interp-for-formula-FEXISTS*[*OF* - ‹*w'* ≠ []›, *of I'*, *unfolded length*[*symmetric*]])
apply *assumption*
done
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*: $P \subseteq \{0 .. length\ w' - 1\}$ *satisfies* (w', *Inr P* # I') φ **by** *auto*
moreover
with *FEXISTS.prems* **have** *satisfies* (*w*, *Inr P* # *I*) φ
apply (*intro iffD2*[*OF FEXISTS.IH*[*of Inr P* # *I Inr P* # *I'*]])
apply (*elim enc-extend-interp*)
apply (*auto split*: *sum.splits split-if-asm*) []
apply (*blast dest*!: *wf-interp-for-formula-FEXISTS*[*OF* - ‹$w \neq$ []›, *of I*, *unfolded length*(*1*)])
apply (*blast dest*!: *wf-interp-for-formula-FEXISTS*[*OF* - ‹$w' \neq$ []›, *of I'*, *unfolded length*(*2*)[*symmetric*]])
apply *assumption*
done
ultimately show *satisfies* (*w*, *I*) (*FEXISTS* φ) **using** *length* **by** (*auto intro*!: *exI*[*of* - *P*])
qed
qed *auto*

lemma $lang_{M2L}$-*FOr*:
assumes *wf-formula n* (*FOr* φ_1 φ_2)
shows $lang_{M2L}$ *n* (*FOr* φ_1 φ_2) $\subseteq$
($lang_{M2L}$ *n* φ_1 $\cup$ $lang_{M2L}$ *n* φ_2) $\cap$ {*enc* (*w*, *I*) | *w I*. *length I* = *n* $\wedge$ *wf-interp-for-formula* (*w*, *I*) (*FOr* φ_1 φ_2)}
(**is** - $\subseteq$ (*?L1* $\cup$ *?L2*) $\cap$ *?ENC*)
proof (*intro equalityI subsetI*)
fix *x* **assume** $x \in lang_{M2L}$ *n* (*FOr* φ_1 φ_2)
then obtain *w I* **where**
$*$: *x* = *enc* (*w*, *I*) *wf-interp-for-formula* (*w*, *I*) (*FOr* φ_1 φ_2) *length I* = *n length* $w > 0$ **and**
satisfies (*w*, *I*) φ_1 $\vee$ *satisfies* (*w*, *I*) φ_2 **unfolding** $lang_{M2L}$-*def* **by** *auto*
thus $x \in$ (*?L1* $\cup$ *?L2*) $\cap$ *?ENC*
proof (*elim disjE*)
assume *satisfies* (*w*, *I*) φ_1
with $*$ **have** $x \in$ *?L1* **using** *assms* **unfolding** $lang_{M2L}$-*def* **by** (*fastforce*)
with $*$ **show** *?thesis* **by** *auto*
next
assume *satisfies* (*w*, *I*) φ_2
with $*$ **have** $x \in$*?L2* **using** *assms* **unfolding** $lang_{M2L}$-*def* **by** (*fastforce*)
with $*$ **show** *?thesis* **by** *auto*
qed
qed

lemma $lang_{M2L}$-*FAnd*:
assumes *wf-formula n* (*FAnd* φ_1 φ_2)
shows $lang_{M2L}$ *n* (*FAnd* φ_1 φ_2) $\subseteq$
$lang_{M2L}$ *n* φ_1 $\cap$ $lang_{M2L}$ *n* φ_2 $\cap$ {*enc* (*w*, *I*) | *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** $lang_{M2L}$*-def* **by** (*fastforce*)

7.3 From M2L to Regular expressions

fun *rexp-of* :: $nat \Rightarrow 'a\ formula \Rightarrow ('a \times bool\ list)\ rexp$ **where**
 rexp-of n $(FQ\ a\ m)$ =
  Inter (*TIMES* [*rexp.Not Zero*, *arbitrary-except* n [$(m, True)$] [a], *rexp.Not Zero*])
   (*ENC* n $(FQ\ a\ m)$)
| *rexp-of* n $(FLess\ m1\ m2)$ = (*if* $m1 = m2$ *then Zero else*
   Inter (*TIMES* [*rexp.Not Zero*, *arbitrary-except* n [$(m1, True)$] Σ,
    rexp.Not Zero, *arbitrary-except* n [$(m2, True)$] Σ,
    rexp.Not Zero]) (*ENC* n $(FLess\ m1\ m2)$))
| *rexp-of* n $(FIn\ m\ M)$ =
   Inter (*TIMES* [*rexp.Not Zero*, *arbitrary-except* n [$(min\ m\ M, True)$, $(max\ m\ M, True)$] Σ, *rexp.Not Zero*])
   (*ENC* n $(FIn\ m\ M)$)
| *rexp-of* n $(FNot\ \varphi)$ = *Inter* (*rexp.Not* (*rexp-of* n φ)) (*ENC* n $(FNot\ \varphi)$)
| *rexp-of* n $(FOr\ \varphi_1\ \varphi_2)$ = *Inter* (*Plus* (*rexp-of* n φ_1) (*rexp-of* n φ_2)) (*ENC* n $(FOr\ \varphi_1\ \varphi_2)$)
| *rexp-of* n $(FAnd\ \varphi_1\ \varphi_2)$ = *INTERSECT* [*rexp-of* n φ_1, *rexp-of* n φ_2, *ENC* n $(FAnd\ \varphi_1\ \varphi_2)$]
| *rexp-of* n $(FExists\ \varphi)$ = *Pr* (*rexp-of* $(n + 1)$ φ)
| *rexp-of* n $(FEXISTS\ \varphi)$ = *Pr* (*rexp-of* $(n + 1)$ φ)

fun *rexp-of-alt* :: $nat \Rightarrow 'a\ formula \Rightarrow ('a \times bool\ list)\ rexp$ **where**
 rexp-of-alt n $(FQ\ a\ m)$ =
  TIMES [*rexp.Not Zero*, *arbitrary-except* n [$(m, True)$] [a], *rexp.Not Zero*]
| *rexp-of-alt* n $(FLess\ m1\ m2)$ = (*if* $m1 = m2$ *then Zero else*
  TIMES [*rexp.Not Zero*, *arbitrary-except* n [$(m1, True)$] Σ,
   rexp.Not Zero, *arbitrary-except* n [$(m2, True)$] Σ,
   rexp.Not Zero])
| *rexp-of-alt* n $(FIn\ m\ M)$ =
  TIMES [*rexp.Not Zero*, *arbitrary-except* n [$(min\ m\ M, True)$, $(max\ m\ M, True)$] Σ, *rexp.Not Zero*]
| *rexp-of-alt* n $(FNot\ \varphi)$ = *rexp.Not* (*rexp-of-alt* n φ)
| *rexp-of-alt* n $(FOr\ \varphi_1\ \varphi_2)$ = *Plus* (*rexp-of-alt* n φ_1) (*rexp-of-alt* n φ_2)
| *rexp-of-alt* n $(FAnd\ \varphi_1\ \varphi_2)$ = *Inter* (*rexp-of-alt* n φ_1) (*rexp-of-alt* n φ_2)
| *rexp-of-alt* n $(FExists\ \varphi)$ = *Pr* (*Inter* (*rexp-of-alt* $(n + 1)$ φ) (*ENC* $(n + 1)$ φ))
| *rexp-of-alt* n $(FEXISTS\ \varphi)$ = *Pr* (*Inter* (*rexp-of-alt* $(n + 1)$ φ) (*ENC* $(n + 1)$ φ))

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

lemma *length-dec-interp*[*simp*]: *length* (*dec-interp* n *FO* x) = n
 unfolding *dec-interp-def* **by** *auto*

theorem $lang_{M2L}$*-rexp-of*: *wf-formula* n $\varphi \Longrightarrow lang_{M2L}\ n\ \varphi = lang\ n$ (*rexp-of* n

φ) − {[]}
(**is** - $\Longrightarrow$ - = *?L n* φ)
proof (*induct* φ *arbitrary*: *n*)
case (*FQ a m*)
show *?case*
proof (*intro equalityI subsetI*)
fix *x* **assume** *x* $\in$ $lang_{M2L}$ *n* (*FQ a m*)
then obtain *w I* **where**
$*$: *x* = *enc* (*w*, *I*) *wf-interp-for-formula* (*w*, *I*) (*FQ a m*) *satisfies* (*w*, *I*) (*FQ a m*)
length I = *n*
unfolding $lang_{M2L}$*-def* **by** *blast*
with *FQ*(*1*) **obtain** *p* **where** *p*: *p* < *length w I* ! *m* = *Inl p w* ! *p* = *a*
by (*auto simp*: *all-set-conv-all-nth split*: *sum.splits*)
with $*$(*1*) **have** *x* = *take p* (*enc* (*w*, *I*)) @ [*enc-atom I p a*] @ *drop* (*p* + *1*) (*enc* (*w*, *I*))
using *id-take-nth-drop*[*of p enc* (*w*, *I*)] **by** *auto*
moreover from $*$(*4*) *FQ*(*1*) *p*(*2*)
have [*enc-atom I p a*] $\in$ *lang n* (*arbitrary-except n* [(*m*, *True*)] [*a*])
by (*intro enc-atom-lang-arbitrary-except-True*) *auto*
moreover from $*$(*2*,*4*) **have** *take p* (*enc* (*w*, *I*)) $\in$ *lang n* (*rexp.Not Zero*)
by (*auto intro*!: *enc-atom-*σ *dest*!: *in-set-takeD*)
moreover from $*$(*2*,*4*) **have** *drop* (*Suc p*) (*enc* (*w*, *I*)) $\in$ *lang n* (*rexp.Not Zero*)
by (*auto intro*!: *enc-atom-*σ *dest*!: *in-set-dropD*)
ultimately show *x* $\in$ *?L n* (*FQ a m*) **using** $*$(*1*,*2*,*4*)
unfolding *rexp-of.simps lang.simps*(*5*,*8*) *rexp-of-list.simps Int-Diff lang-ENC*[*OF FQ*]
by (*auto elim*: *ssubst simp del*: *o-apply append.simps lang.simps*)
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-interp-for-formula* (*w*, *I*) (*FQ a m*)
unfolding *rexp-of.simps lang.simps lang-ENC*[*OF FQ*] *Int-Diff* **by** *atomize-elim* (*auto split*: *sum.splits*)
hence *wf-interp-for-formula* (*dec-word x*, *dec-interp n* {*m*} *x*) (*FQ a m*) **unfolding** *wI*(*1*)
using *enc-wf-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* (*arbitrary-except n* [(*m*, *True*)] [*a*])
unfolding *rexp-of.simps lang.simps rexp-of-list.simps* **using** *concE* **by** *fast*
with *FQ*(*1*) **obtain** *v* **where** *v*: *x* = *u1* @ [*v*] @ *u2 snd v* ! *m fst v* = *a*
using *arbitrary-except*[*of u n m True* [*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-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*
} **note** $* = this$
from *v* **have** $v = enc\ (w,\ I)\ !\ length\ u1$ **unfolding** *wI*(*1*) **by** *simp*
hence $a = w\ !\ length\ u1$ **using** *nth-map*[*OF u*, *of fst*] **unfolding** *wI*(*1*) *v*(*3*)[*symmetric*] **by** *auto*
with $*$ *m wI* **have** *satisfies* (*dec-word x*, *dec-interp n* $\{m\}$ *x*) (*FQ a 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-*σ)
ultimately show $x \in lang_{M2L}\ n\ (FQ\ a\ m)$ **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** $x \in lang_{M2L}\ n\ (FLess\ m\ m')$
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*)] $\in$ *lang n* (*arbitrary-except n* [(*m*, *True*)] Σ)
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*)] $\in$ *lang n* (*arbitrary-except n* [(m', *True*)] Σ)
by (*intro enc-atom-lang-arbitrary-except-True*) *auto*
moreover from $*(2,4)$ **have** *take p* (*enc* (*w*, *I*)) $\in$ *lang n* (*rexp.Not Zero*)
by (*auto intro*!: *enc-atom-*σ *dest*!: *in-set-takeD*)
moreover from $*(2,4)$ **have** *take* ($q - p - 1$) (*drop* (*Suc p*) (*enc* (*w*, *I*))) $\in$ *lang n* (*rexp.Not Zero*)
by (*auto intro*!: *enc-atom-*σ *dest*!: *in-set-dropD in-set-takeD*)
moreover from $*(2,4)$ **have** *drop* ($q - p$) (*drop* (*Suc p*) (*enc* (*w*, *I*))) $\in$ *lang n* (*rexp.Not Zero*)
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*(*5*,*8*) *rexp-of-list.simps Int-Diff lang-ENC*[*OF FLess*] *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*[*OF FLess*] *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* (*arbitrary-except n* [(*m*, *True*)] Σ)
$u' \in$ *lang n* (*arbitrary-except n* [(m', *True*)] Σ)
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-except*[*of u n m True* Σ] *arbitrary-except*[*of* u' *n* m' *True* Σ] **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*: ∃!*p*. *p* < *length x* ∧ *snd* (*x* ! *p*) ! *m'* **unfolding** *wI*(*1*) **by** (*intro enc-unique*) *auto*
ultimately have *p'* = *?u'* **using** *u'* **by** *auto*
} **note** ∗ = *this* ‹*p* = *length u1*›
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-σ*)
ultimately show $x \in lang_{M2L}$ *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** $x \in lang_{M2L}$ *n* (*FIn m M*)
then obtain *w I* **where**
∗: *x* = *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* < *length w I* ! *m* = *Inl p I* ! *M* = *Inr P p* ∈ *P*
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*
moreover
have [*enc-atom I p* (*w* ! *p*)] ∈ *lang n* (*arbitrary-except n* [(*min m M*, *True*), (*max m M*, *True*)] Σ)
proof (*cases m* < *M*)
case *True* **with** ∗(*2*,*4*) *FIn*(*1*) *p* **show** *?thesis*
by (*intro enc-atom-lang-arbitrary-except-True2*) (*auto simp*: *min-absorb1 max-absorb2*)
next
case *False* **with** ∗(*2*,*4*) *FIn*(*1*) *p* **show** *?thesis*

by (*intro enc-atom-lang-arbitrary-except-True2*) (*auto simp*: *min-absorb2 max-absorb1*)
qed
moreover from ∗(*2*,*4*) **have** *take p* (*enc* (*w*, *I*)) ∈ *lang n* (*rexp.Not Zero*)
by (*auto intro*!: *enc-atom-σ dest*!: *in-set-takeD*)
moreover from ∗(*2*,*4*) **have** *drop* (*Suc p*) (*enc* (*w*, *I*)) ∈ *lang n* (*rexp.Not Zero*)
by (*auto intro*!: *enc-atom-σ dest*!: *in-set-dropD*)
ultimately show *x* ∈ *?L n* (*FIn m M*) **using** ∗(*1*,*2*,*4*)
unfolding *rexp-of.simps lang.simps*(*5*,*8*) *rexp-of-list.simps Int-Diff lang-ENC*[*OF FIn*]
by (*auto elim*: *ssubst simp del*: *o-apply append.simps lang.simps*)
next
fix *x* **assume** *x*: *x* ∈ *?L n* (*FIn m M*)
with *FIn* **obtain** *w I* **where** *wI*: *x* = *enc* (*w*, *I*) *length I* = *n wf-interp-for-formula* (*w*, *I*) (*FIn m M*)
unfolding *rexp-of.simps lang.simps lang-ENC*[*OF FIn*] *Int-Diff* **by** (*fastforce split*: *sum.splits*)
with *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 *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 ∈ *lang n* (*arbitrary-except n* [(*min m M*, *True*), (*max m M*, *True*)] Σ)
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* **and** *snd v* ! *min m M snd v* ! *max m M*
using *arbitrary-except2*[*of u n min m M True max m M True* Σ] **by** *fastforce*
hence *v′*: *snd v* ! *m snd v* ! *M*
by (*induct m* < *M*) (*auto simp*: *min-absorb1 min-absorb2 max-absorb1 max-absorb2*)
from *v* **have** *u*: *length u1* < *length x* **by** *auto*
{ from *v 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*: ∃!*p*. *p* < *length x* ∧ *snd* (*x* ! *p*) ! *m* **unfolding** *wI*(*1*) **by** (*intro enc-unique*) *auto*
ultimately have *p* = *length u1* **using** *u* **by** *auto*
} **note** ∗ = *this*
from *v v′* **have** *v* = *enc* (*w*, *I*) ! *length u1* **unfolding** *wI*(*1*) **by** *simp*
with *v′*(*2*) *m*(*3*,*4*) *u wI*(*1*) **have** *length u1* ∈ *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-σ*)
ultimately show $x \in lang_{M2L}$ *n* (*FIn 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*} *x*]
intro: *sym*[*OF enc-dec*[*OF - ballI*[*OF impI*[*OF enc-unique*[*of w I*, *folded wI*(*1*)]]]]]])
qed
next
case (*FOr* φ_1 φ_2)
from *FOr*(*3*) **have** *IH1*: $lang_{M2L}$ *n* φ_1 = *lang n* (*rexp-of n* φ_1) − {[]}
by (*intro FOr*(*1*)) *auto*
from *FOr*(*3*) **have** *IH2*: $lang_{M2L}$ *n* φ_2 = *lang n* (*rexp-of n* φ_2) − {[]}
by (*intro FOr*(*2*)) *auto*
show *?case*
proof (*intro equalityI subsetI*)
fix *x* **assume** $x \in lang_{M2L}$ *n* (*FOr* φ_1 φ_2) **thus** $x \in$ *lang n* (*rexp-of n* (*FOr* φ_1 φ_2)) − {[]}
using $lang_{M2L}$*-FOr*[*OF FOr*(*3*)] **unfolding** *lang-ENC*[*OF FOr*(*3*)] *rexp-of.simps lang.simps IH1 IH2 Int-Diff* **by** *auto*
next
fix *x* **assume** $x \in$ *lang n* (*rexp-of n* (*FOr* φ_1 φ_2)) − {[]}
then obtain *w I* **where** *or*: $x \in lang_{M2L}$ *n* $\varphi_1 \vee x \in lang_{M2L}$ *n* φ_2 **and** *wI*: *x* = *enc* (*w*, *I*) *length I* = *n*
wf-interp-for-formula (*w*, *I*) (*FOr* φ_1 φ_2)
unfolding *lang-ENC*[*OF FOr*(*3*)] *rexp-of.simps lang.simps IH1 IH2 Int-Diff* **by** *auto*
have *satisfies* (*w*, *I*) φ_1 ∨ *satisfies* (*w*, *I*) φ_2
proof (*intro mp*[*OF disj-mono*[*OF impI impI*] *or*])
assume $x \in lang_{M2L}$ *n* φ_1
with *wI*(*2*,*3*) *FOr*(*3*) **show** *satisfies* (*w*, *I*) φ_1
unfolding $lang_{M2L}$*-def wI*(*1*) *wf-interp-for-formula-FOr*
by (*auto simp del*: *enc.simps dest*!: *iffD2*[*OF enc-welldef*[*of - - - -* φ_1]])
next
assume $x \in lang_{M2L}$ *n* φ_2
with *wI*(*2*,*3*) *FOr*(*3*) **show** *satisfies* (*w*, *I*) φ_2
unfolding $lang_{M2L}$*-def wI*(*1*) *wf-interp-for-formula-FOr*
by (*auto simp del*: *enc.simps dest*!: *iffD2*[*OF enc-welldef*[*of - - - -* φ_2]])
qed
with *wI* **show** $x \in lang_{M2L}$ *n* (*FOr* φ_1 φ_2) **unfolding** $lang_{M2L}$*-def* **by** *auto*
qed
next
case (*FAnd* φ_1 φ_2)
from *FAnd*(*3*) **have** *IH1*: $lang_{M2L}$ *n* φ_1 = *lang n* (*rexp-of n* φ_1) − {[]}
by (*intro FAnd*(*1*)) *auto*
from *FAnd*(*3*) **have** *IH2*: $lang_{M2L}$ *n* φ_2 = *lang n* (*rexp-of n* φ_2) − {[]}

by (*intro FAnd*(*2*)) *auto*
show *?case*
proof (*intro equalityI subsetI*)
fix x **assume** $x \in lang_{M2L}$ n (*FAnd* φ_1 φ_2) **thus** $x \in$ *lang* n (*rexp-of* n (*FAnd* φ_1 φ_2)) $-$ {[]}
using $lang_{M2L}$*-FAnd*[*OF FAnd*(*3*)]
unfolding *lang-ENC*[*OF FAnd*(*3*)] *rexp-of.simps rexp-of-list.simps lang.simps IH1 IH2 Int-Diff* **by** *auto*
next
fix x **assume** $x \in$ *lang* n (*rexp-of* n (*FAnd* φ_1 φ_2)) $-$ {[]}
then obtain w I **where** *and*: $x \in lang_{M2L}$ n $\varphi_1 \wedge x \in lang_{M2L}$ n φ_2 **and** *wI*: x = *enc* (w, I) *length* $I = n$
wf-interp-for-formula (w, I) (*FAnd* φ_1 φ_2)
unfolding *lang-ENC*[*OF FAnd*(*3*)] *rexp-of.simps rexp-of-list.simps lang.simps IH1 IH2 Int-Diff* **by** *auto*
have *satisfies* (w, I) $\varphi_1 \wedge$ *satisfies* (w, I) φ_2
proof (*intro mp*[*OF conj-mono*[*OF impI impI*] *and*])
assume $x \in lang_{M2L}$ n φ_1
with *wI*(*2*,*3*) *FAnd*(*3*) **show** *satisfies* (w, I) φ_1
unfolding $lang_{M2L}$*-def wI*(*1*) *wf-interp-for-formula-FAnd*
by (*auto simp del*: *enc.simps dest*!: *iffD2*[*OF enc-welldef*[*of - - - -* φ_1]])
next
assume $x \in lang_{M2L}$ n φ_2
with *wI*(*2*,*3*) *FAnd*(*3*) **show** *satisfies* (w, I) φ_2
unfolding $lang_{M2L}$*-def wI*(*1*) *wf-interp-for-formula-FAnd*
by (*auto simp del*: *enc.simps dest*!: *iffD2*[*OF enc-welldef*[*of - - - -* φ_2]])
qed
with *wI* **show** $x \in lang_{M2L}$ n (*FAnd* φ_1 φ_2) **unfolding** $lang_{M2L}$*-def* **by** *auto*
qed
next
case (*FNot* φ)
hence *IH*: *?L* n φ = $lang_{M2L}$ n φ **by** *simp*
show *?case*
proof (*intro equalityI subsetI*)
fix x **assume** $x \in lang_{M2L}$ 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*: $\neg$ (*satisfies* (w, I) φ)
unfolding $lang_{M2L}$*-def* **by** *auto*
{ assume $x \in$ *?L* n φ
with *IH* **have** *satisfies* (w, I) φ **using** *enc-welldef*[*of - - w I* φ, *OF - - -* $*$(*2*)] *FNot*(*2*)
unfolding $*$(*1*,*3*) $lang_{M2L}$*-def* **by** *auto*
}
with *unsat* **have** $x \notin$ *?L* n φ **by** *blast*
with $*$ **show** $x \in$ *?L* n (*FNot* φ) **unfolding** *rexp-of.simps lang.simps* **using** *lang-ENC*[*OF FNot*(*2*)]
by (*auto simp*: *comp-def intro*!: *enc-atom-σ*)
next

fix x **assume** $x \in$ *?L n* (*FNot* φ)
with *IH* **have** $x \in$ *lang n* (*ENC n* (*FNot* φ)) $- \{[]\}$ **and** x: $x \notin lang_{M2L}$ 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*[*OF FNot*(*2*)] **by** *blast*
{ assume $\neg$ *satisfies* (w, I) (*FNot* φ)
with $*$ **have** $x \in lang_{M2L}$ n φ **unfolding** $lang_{M2L}$-*def* **by** *auto*
}
with x $*$ **show** $x \in lang_{M2L}$ n (*FNot* φ) **unfolding** $lang_{M2L}$-*def* **by** *blast*
qed
next
case (*FExists* φ)
show *?case*
proof (*intro equalityI subsetI*)
fix x **assume** $x \in lang_{M2L}$ 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* $w > 0$ $p \in \{0$ .. *length* $w - 1\}$ *satisfies* (w, *Inl* p # I) φ
unfolding $lang_{M2L}$-*def* **by** *auto*
with *FExists*(*2*) **have** *enc* (w, *Inl* p # I) $\in$ *?L* (*Suc n*) φ
by (*intro subsetD*[*OF equalityD1*[*OF FExists*(*1*)], *of Suc n enc* (w, *Inl* p # I)])
(*auto simp*: $lang_{M2L}$-*def nth-Cons'* *ord-less-eq-trans*[*OF le-imp-less-Suc Suc-pred*[*OF* $*$(*4*)]]
split: *split-if-asm sum.splits intro*!: *exI*[*of - w*] *exI*[*of - Inl p # I*])
with $*$(*1*) **show** $x \in$ *?L n* (*FExists* φ)
by (*auto simp*: *map-index intro*!: *image-eqI*[*of - map* π] *simp del*: *o-apply*) (*auto simp*: π-*def*)
next
fix x **assume** $x \in$ *?L n* (*FExists* φ)
then obtain x' **where** x: $x = map$ π x' **and** $x' \in$ *?L* (*Suc n*) φ **by** (*auto simp del*: *o-apply*)
with *FExists*(*2*) **have** $x' \in lang_{M2L}$ (*Suc n*) φ
by (*intro subsetD*[*OF equalityD2*[*OF FExists*(*1*)], *of Suc n x'*])
(*auto split*: *split-if-asm sum.splits*)
then obtain w I' **where**
$*$: $x' = enc$ (w, I') *wf-interp-for-formula* (w, I') φ *length* $I' =$ *Suc n satisfies* (w, I') φ
unfolding $lang_{M2L}$-*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 split-if-asm*)
ultimately have $x = enc$ (w, I) *wf-interp-for-formula* (w, I) (*FExists* φ) *length* $I = n$
length $w > 0$ *satisfies* (w, I) (*FExists* φ)**using** *FExists*(*2*) **unfolding** x
by (*auto simp*: *map-tl nth-Cons'* *split*: *split-if-asm simp del*: *o-apply*) (*auto simp*: π-*def*)
thus $x \in lang_{M2L}$ n (*FExists* φ) **unfolding** $lang_{M2L}$-*def* **by** (*auto intro*!:

exI[*of* - *w*] *exI*[*of* - *I*])
qed
next
case (*FEXISTS* φ)
show *?case*
proof (*intro equalityI subsetI*)
fix x **assume** $x \in lang_{M2L}$ *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* $P \subseteq \{0\ ..\ length\ w - 1\}$ *satisfies* (*w*, *Inr P* # *I*) φ
unfolding $lang_{M2L}$*-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*) φ
by (*intro subsetD*[*OF equalityD1*[*OF FEXISTS*(*1*)], *of Suc n enc* (*w*, *Inr P* # *I*)]])
(*auto simp*: $lang_{M2L}$*-def nth-Cons′ split*: *split-if-asm sum.splits*
intro!: *exI*[*of* - *w*] *exI*[*of* - *Inr P* # *I*])
with $*(1)$ **show** $x \in$ *?L n* (*FEXISTS* φ)
by (*auto simp*: *map-index intro*!: *image-eqI*[*of* - *map* π] *simp del*: *o-apply*) (*auto simp*: π*-def*)
next
fix x **assume** $x \in$ *?L n* (*FEXISTS* φ)
then obtain x' **where** *x*: $x = map\ \pi\ x'$ **and** x': *length* $x' > 0$ **and** $x' \in$ *?L* (*Suc n*) φ **by** (*auto simp del*: *o-apply*)
with *FEXISTS*(*2*) **have** $x' \in lang_{M2L}$ (*Suc n*) φ
by (*intro subsetD*[*OF equalityD2*[*OF FEXISTS*(*1*)], *of Suc n x′*])
(*auto split*: *split-if-asm sum.splits*)
then obtain *w* I' **where**
$*$: x' = *enc* (*w*, I') *wf-interp-for-formula* (*w*, I') φ *length* I' = *Suc n satisfies* (*w*, I') φ
unfolding $lang_{M2L}$*-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 = *Inr P*
by (*auto simp*: *nth-Cons′ split*: *sum.splits split-if-asm*)
moreover have *length w* $\geq$ *1* **using** x' $*(1)$ **by** (*cases w*) *auto*
ultimately have x = *enc* (*w*, *I*) *wf-interp-for-formula* (*w*, *I*) (*FEXISTS* φ) *length I* = *n*
length w > *0 satisfies* (*w*, *I*) (*FEXISTS* φ) **using** *FEXISTS*(*2*) **unfolding** *x*
by (*auto simp add*: *map-tl nth-Cons′ split*: *split-if-asm*
intro!: *exI*[*of* - *P*] *simp del*: *o-apply*) (*auto simp*: π*-def*)
thus $x \in lang_{M2L}$ *n* (*FEXISTS* φ) **unfolding** $lang_{M2L}$*-def* **by** (*auto intro*!: *exI*[*of* - *w*] *exI*[*of* - *I*])
qed
qed

lemma *wf-rexp-of*: *wf-formula n* $\varphi \Longrightarrow$ *wf n* (*rexp-of n* φ)
by (*induct* φ *arbitrary*: *n*) (*auto simp*: *wf-rexp-ENC intro*: *wf-rexp-arbitrary-except split*: *sum.splits split-if-asm*)

lemma *wf-rexp-of'*: *wf-formula n* $\varphi \Longrightarrow$ *wf n* (*rexp-of' n* φ)
unfolding *rexp-of'-def*
by (*induct* φ *arbitrary*: *n*) (*auto simp*: *wf-rexp-ENC intro*: *wf-rexp-arbitrary-except split*: *sum.splits split-if-asm*)

lemma *ENC-Not*: *ENC n* (*FNot* φ) = *ENC n* φ
unfolding *ENC-def* **by** *auto*

lemma *ENC-And*:
wf-formula n (*FAnd* φ ψ) $\Longrightarrow$ *lang n* (*ENC n* (*FAnd* φ ψ)) $-$ {[]} $\subseteq$ *lang n* (*ENC n* φ) $\cap$ *lang n* (*ENC n* ψ) $-$ {[]}
proof
fix *x* **assume** *wf*: *wf-formula n* (*FAnd* φ ψ) **and** *x*: $x \in$ *lang n* (*ENC n* (*FAnd* φ ψ)) $-$ {[]}
hence *wf1*: *wf-formula n* φ **and** *wf2*: *wf-formula n* ψ **by** *auto*
from *x* **obtain** *w I* **where** *wI*: *x* = *enc* (*w*, *I*) *wf-interp-for-formula* (*w*, *I*) (*FAnd* φ ψ) *length I* = *n*
using *lang-ENC*[*OF wf*] **by** *auto*
hence *wf-interp-for-formula* (*w*, *I*) φ *wf-interp-for-formula* (*w*, *I*) ψ
unfolding *wf-interp-for-formula-FAnd* **by** *auto*
hence $x \in$ (*lang n* (*ENC n* φ) $-$ {[]}) $\cap$ (*lang n* (*ENC n* ψ) $-$ {[]})
unfolding *lang-ENC*[*OF wf1*] *lang-ENC*[*OF wf2*] **using** *wI* **by** *auto*
thus $x \in$ *lang n* (*ENC n* φ) $\cap$ *lang n* (*ENC n* ψ) $-$ {[]} **by** *blast*
qed

lemma *ENC-Or*:
wf-formula n (*FOr* φ ψ) $\Longrightarrow$ *lang n* (*ENC n* (*FOr* φ ψ)) $-$ {[]} $\subseteq$ *lang n* (*ENC n* φ) $\cap$ *lang n* (*ENC n* ψ) $-$ {[]}
proof
fix *x* **assume** *wf*: *wf-formula n* (*FOr* φ ψ) **and** *x*: $x \in$ *lang n* (*ENC n* (*FOr* φ ψ)) $-$ {[]}
hence *wf1*: *wf-formula n* φ **and** *wf2*: *wf-formula n* ψ **by** *auto*
from *x* **obtain** *w I* **where** *wI*: *x* = *enc* (*w*, *I*) *wf-interp-for-formula* (*w*, *I*) (*FOr* φ ψ) *length I* = *n*
using *lang-ENC*[*OF wf*] **by** *auto*
hence *wf-interp-for-formula* (*w*, *I*) φ *wf-interp-for-formula* (*w*, *I*) ψ
unfolding *wf-interp-for-formula-FOr* **by** *auto*
hence $x \in$ (*lang n* (*ENC n* φ) $-$ {[]}) $\cap$ (*lang n* (*ENC n* ψ) $-$ {[]})
unfolding *lang-ENC*[*OF wf1*] *lang-ENC*[*OF wf2*] **using** *wI* **by** *auto*
thus $x \in$ *lang n* (*ENC n* φ) $\cap$ *lang n* (*ENC n* ψ) $-$ {[]} **by** *blast*
qed

lemma *project-enc*: *map* π (*enc* (*w*, *x* # *I*)) = *enc* (*w*, *I*)
unfolding *π-def* **by** *auto*

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

lang (*Suc n*) (*ENC* (*Suc n*) φ) − {[]}
proof (*intro equalityI subsetI*)
fix *x* **assume** *wf*: *wf-formula n* (*FExists* φ) **and** *x*: $x \in$ *lang n* (*ENC n* (*FExists* φ)) − {[]}
hence *wf1*: *wf-formula* (*Suc n*) φ **by** *auto*
from *x* **obtain** *w I* **where** *wI*: x = *enc* (*w*, *I*) *wf-interp-for-formula* (*w*, *I*) (*FExists* φ) *length I* = *n*
using *lang-ENC*[*OF wf*] **by** *auto*
with *x* **have** $w \neq$ [] **by** (*cases w*) *auto*
from *wI*(*2*) **obtain** *p* **where** *p* < *length w wf-interp-for-formula* (*w*, *Inl p* # *I*) φ
using *wf-interp-for-formula-FExists*[*OF wf*[*folded wI*(*3*)] ‹*w* ≠ []›] **by** *auto*
with *wI*(*3*) **have** $x \in$ *map* π ‘ (*lang* (*Suc n*) (*ENC* (*Suc n*) φ) − {[]})
unfolding *wI*(*1*) *lang-ENC*[*OF wf1*] *project-enc*[*symmetric*, *of w I Inl p*]
by (*intro imageI CollectI exI*[*of - w*] *exI*[*of - Inl p* # *I*]) *auto*
thus $x \in$ *map* π ‘ *lang* (*Suc n*) (*ENC* (*Suc n*) φ) − {[]} **by** *blast*
next
fix *x* **assume** *wf*: *wf-formula n* (*FExists* φ) **and** $x \in$ *map* π ‘ *lang* (*Suc n*) (*ENC* (*Suc n*) φ) − {[]}
hence *wf1*: *wf-formula* (*Suc n*) φ **and** *0* ∈ *FOV* φ **and** *x*: $x \in$ *map* π ‘ (*lang* (*Suc n*) (*ENC* (*Suc n*) φ) − {[]}) **by** *auto*
from *x* **obtain** *w I* **where** *wI*: x = *map* π (*enc* (*w*, *I*)) *wf-interp-for-formula* (*w*, *I*) φ *length I* = *Suc n*
using *lang-ENC*[*OF wf1*] **by** *auto*
with ‹*0* ∈ *FOV* φ› **obtain** *p I′* **where** *I*: *I* = *Inl p* # *I′* **by** (*cases I*) (*fastforce split*: *sum.splits*)+
with *wI* **have** *wtlI*: x = *enc* (*w*, *I′*) *length I′* = *n* **unfolding** π*-def* **by** *auto*
with *x* **have** $w \neq$ [] **by** (*cases w*) *auto*
have *wf-interp-for-formula* (*w*, *I′*) (*FExists* φ)
using *wf-interp-for-formula-FExists*[*OF wf*[*folded wtlI*(*2*)] ‹*w* ≠ []›]
wf-interp-for-formula-any-Inl[*OF wI*(*2*)[*unfolded I*]] **..**
with *wtlI* **show** $x \in$ *lang n* (*ENC n* (*FExists* φ)) − {[]} **unfolding** *lang-ENC*[*OF wf*] **by** *blast*
qed

lemma *ENC-EXISTS*:
wf-formula n (*FEXISTS* φ) ⟹ *lang n* (*ENC n* (*FEXISTS* φ)) − {[]} = *map* π ‘ *lang* (*Suc n*) (*ENC* (*Suc n*) φ) − {[]}
proof (*intro equalityI subsetI*)
fix *x* **assume** *wf*: *wf-formula n* (*FEXISTS* φ) **and** *x*: $x \in$ *lang n* (*ENC n* (*FEXISTS* φ)) − {[]}
hence *wf1*: *wf-formula* (*Suc n*) φ **by** *auto*
from *x* **obtain** *w I* **where** *wI*: x = *enc* (*w*, *I*) *wf-interp-for-formula* (*w*, *I*) (*FEXISTS* φ) *length I* = *n*
using *lang-ENC*[*OF wf*] **by** *auto*
with *x* **have** $w \neq$ [] **by** (*cases w*) *auto*
from *wI*(*2*) **obtain** *P* **where** ∀ *p* ∈ *P*. *p* < *length w wf-interp-for-formula* (*w*, *Inr P* # *I*) φ
using *wf-interp-for-formula-FEXISTS*[*OF wf*[*folded wI*(*3*)] ‹*w* ≠ []›] **by** *auto*

with $wI(3)$ **have** $x \in map\ \pi\ `\ (lang\ (Suc\ n)\ (ENC\ (Suc\ n)\ \varphi) - \{[]\})$
unfolding $wI(1)$ $lang\text{-}ENC[OF\ wf1]$ $project\text{-}enc[symmetric,\ of\ w\ I\ Inr\ P]$
by $(intro\ imageI\ CollectI\ exI[of\ \text{-}\ w]\ exI[of\ \text{-}\ Inr\ P\ \#\ I])$ *auto*
thus $x \in map\ \pi\ `\ lang\ (Suc\ n)\ (ENC\ (Suc\ n)\ \varphi) - \{[]\}$ **by** *blast*
next
fix x **assume** wf: $wf\text{-}formula\ n\ (FEXISTS\ \varphi)$ **and** $x \in map\ \pi\ `\ lang\ (Suc\ n)\ (ENC\ (Suc\ n)\ \varphi) - \{[]\}$
hence $wf1$: $wf\text{-}formula\ (Suc\ n)\ \varphi$ **and** $0 \in SOV\ \varphi$ **and** x: $x \in map\ \pi\ `\ (lang\ (Suc\ n)\ (ENC\ (Suc\ n)\ \varphi) - \{[]\})$ **by** *auto*
from x **obtain** $w\ I$ **where** wI: $x = map\ \pi\ (enc\ (w,\ I))$ $wf\text{-}interp\text{-}for\text{-}formula\ (w,\ I)\ \varphi$ $length\ I = Suc\ n$
using $lang\text{-}ENC[OF\ wf1]$ **by** *auto*
with $\langle 0 \in SOV\ \varphi\rangle$ **obtain** $P\ I'$ **where** I: $I = Inr\ P\ \#\ I'$ **by** $(cases\ I)$ $(fastforce\ split\text{: }sum.splits)+$
with wI **have** $wtlI$: $x = enc\ (w,\ I')$ $length\ I' = n$ **unfolding** $\pi\text{-}def$ **by** *auto*
with x **have** $w \neq []$ **by** $(cases\ w)$ *auto*
have $wf\text{-}interp\text{-}for\text{-}formula\ (w,\ I')\ (FEXISTS\ \varphi)$
using $wf\text{-}interp\text{-}for\text{-}formula\text{-}FEXISTS[OF\ wf[folded\ wtlI(2)]\ \langle w \neq []\rangle]$
$wf\text{-}interp\text{-}for\text{-}formula\text{-}any\text{-}Inr[OF\ wI(2)[unfolded\ I]]$ **..**
with $wtlI$ **show** $x \in lang\ n\ (ENC\ n\ (FEXISTS\ \varphi)) - \{[]\}$ **unfolding** $lang\text{-}ENC[OF\ wf]$ **by** *blast*
qed

lemma $lang_{M2L}\text{-}rexp\text{-}of\text{-}rexp\text{-}of'$:
$wf\text{-}formula\ n\ \varphi \Longrightarrow lang\ n\ (rexp\text{-}of\ n\ \varphi) - \{[]\} = lang\ n\ (rexp\text{-}of'\ n\ \varphi) - \{[]\}$
unfolding $rexp\text{-}of'\text{-}def$ **proof** $(induction\ \varphi\ arbitrary\text{: }n)$
case $(FNot\ \varphi)$
hence $wf\text{-}formula\ n\ \varphi$ **by** *simp*
with $FNot.IH$ **show** *?case* **unfolding** $rexp\text{-}of.simps$ $rexp\text{-}of\text{-}alt.simps$ $lang.simps$ $ENC\text{-}Not$ **by** *blast*
next
case $(FAnd\ \varphi_1\ \varphi_2)$
hence $wf1$: $wf\text{-}formula\ n\ \varphi_1$ **and** $wf2$: $wf\text{-}formula\ n\ \varphi_2$ **by** *force+*
from $FAnd.IH(1)[OF\ wf1]$ $FAnd.IH(2)[OF\ wf2]$ **show** *?case* **using** $ENC\text{-}And[OF\ FAnd.prems]$
unfolding $rexp\text{-}of.simps$ $rexp\text{-}of\text{-}alt.simps$ $lang.simps$ $rexp\text{-}of\text{-}list.simps$ **by** *blast*
next
case $(FOr\ \varphi_1\ \varphi_2)$
hence $wf1$: $wf\text{-}formula\ n\ \varphi_1$ **and** $wf2$: $wf\text{-}formula\ n\ \varphi_2$ **by** *force+*
from $FOr.IH(1)[OF\ wf1]$ $FOr.IH(2)[OF\ wf2]$ **show** *?case* **using** $ENC\text{-}Or[OF\ FOr.prems]$
unfolding $rexp\text{-}of.simps$ $rexp\text{-}of\text{-}alt.simps$ $lang.simps$ **by** *blast*
next
case $(FExists\ \varphi)$
hence wf: $wf\text{-}formula\ (n + 1)\ \varphi$ **by** *auto*
have $*$: $\bigwedge A.\ map\ \pi\ `\ A - \{[]\} = map\ \pi\ `\ (A - \{[]\})$ **by** *auto*
show *?case* **using** $ENC\text{-}Exists[OF\ FExists.prems]$
unfolding $rexp\text{-}of.simps$ $rexp\text{-}of\text{-}alt.simps$ $lang.simps$ $*$ $FExists.IH[OF\ wf]$ **by** *auto*

next
  case $(FEXISTS\ \varphi)$
  hence wf: $wf\text{-}formula\ (n + 1)\ \varphi$ **by** $auto$
  have $*$: $\bigwedge A.\ map\ \pi$ ' $A - \{[]\} = map\ \pi$ ' $(A - \{[]\})$ **by** $auto$
  show $?case$ **using** $ENC\text{-}EXISTS[OF\ FEXISTS.prems]$
    unfolding $rexp\text{-}of.simps\ rexp\text{-}of\text{-}alt.simps\ lang.simps * FEXISTS.IH[OF\ wf]$
by $auto$
qed $auto$

theorem $lang_{M2L}\text{-}rexp\text{-}of'$: $wf\text{-}formula\ n\ \varphi \Longrightarrow lang_{M2L}\ n\ \varphi = lang\ n\ (rexp\text{-}of'\ n\ \varphi) - \{[]\}$
  unfolding $lang_{M2L}\text{-}rexp\text{-}of\text{-}rexp\text{-}of'[symmetric]$ **by** $(rule\ lang_{M2L}\text{-}rexp\text{-}of)$

end

end

8 Normalization of M2L Formulas

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

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

context $formula$
begin

lemma $satisfies\text{-}nNot[simp]$: $satisfies\ (w,\ I)\ (nNot\ \varphi) = satisfies\ (w,I)\ (FNot\ \varphi)$
  by $(induct\ \varphi\ rule\text{: } nNot.induct)\ auto$

lemma $FOV\text{-}nNot[simp]$: $FOV\ (nNot\ \varphi) = FOV\ (FNot\ \varphi)$
  by $(induct\ \varphi\ rule\text{: } nNot.induct)\ auto$

lemma $SOV\text{-}nNot[simp]$: $SOV\ (nNot\ \varphi) = SOV\ (FNot\ \varphi)$
  by $(induct\ \varphi\ rule\text{: } nNot.induct)\ auto$

lemma *pre-wf-formula-nNot*[*simp*]: *pre-wf-formula n* (*nNot* φ) = *pre-wf-formula n* (*FNot* φ)
by (*induct* φ *rule*: *nNot.induct*) *auto*

lemma *FOV-norm*[*simp*]: *FOV* (*norm* φ) = *FOV* φ
by (*induct* φ) *auto*

lemma *SOV-norm*[*simp*]: *SOV* (*norm* φ) = *SOV* φ
by (*induct* φ) *auto*

lemma *pre-wf-formula-norm*[*simp*]: *pre-wf-formula n* (*norm* φ) = *pre-wf-formula n* φ
by (*induct* φ *arbitrary*: *n*) *auto*

lemma *satisfies-norm*[*simp*]: *satisfies* (*w*, *I*) (*norm* φ) = *satisfies* (*w*, *I*) φ
by (*induct* φ *arbitrary*: *I*) *auto*

lemma $lang_{M2L}$*-norm*[*simp*]: $lang_{M2L}$ *n* (*norm* φ) = $lang_{M2L}$ *n* φ
unfolding $lang_{M2L}$*-def* **by** *auto*

end

end

9 Deciding Equivalence of M2L Formulas

type-synonym *'a T* = *'a* × *bool list*
abbreviation $\mathfrak{L} \equiv \lambda\Sigma.$ *project.lang* (*set o* (σ Σ)) π

definition *wf-rexp* **where** [*code del*]:
wf-rexp Σ = *alphabet.wf* (*set o* σ Σ)

interpretation *project set o* σ Σ π
where *alphabet.wf* (*set o* σ Σ) = *wf-rexp* Σ
by (*unfold-locales*) (*auto simp*: σ*-def* π*-def* *wf-rexp-def*)

definition *norm-lderiv* **where** [*code del*]:
norm-lderiv $\equiv \lambda\Sigma.$ *embed.lderiv* (ε Σ)

interpretation *embed set* $\circ$ (σ (Σ :: *'a* :: *linorder list*)) π ε Σ
where *embed.lderiv* (ε Σ) = *norm-lderiv* Σ
by (*unfold-locales*) (*auto simp*: *norm-lderiv-def* σ*-def* π*-def* ε*-def*)

definition *norm-step'* **where** [*code del*]:
norm-step' $\equiv \lambda\Sigma.$ *equivalence-checker.step'* (σ Σ) (ε Σ) (*Smart-Constructors-Normalization.norm* :: *'a*::*linorder T rexp* $\Rightarrow$ *'a T rexp*)
definition *norm-closure'* **where** [*code del*]:

norm-closure' ≡ λΣ. *equivalence-checker.closure'* (σ Σ) (ε Σ) (*Smart-Constructors-Normalization.norm* :: *'a::linorder T rexp* ⇒ *'a T rexp*)
definition *norm-check-eqv'* **where** [*code del*]:
norm-check-eqv' ≡ λΣ. *equivalence-checker.check-eqv'* (σ Σ) (ε Σ) (*Smart-Constructors-Normalization.norm* :: *'a::linorder T rexp* ⇒ *'a T rexp*)
definition *norm-step* **where** [*code del*]:
norm-step ≡ λΣ. *equivalence-checker.step* (σ Σ) (ε Σ) (*Smart-Constructors-Normalization.norm* :: *'a::linorder T rexp* ⇒ *'a T rexp*)
definition *norm-closure* **where** [*code del*]:
norm-closure ≡ λΣ. *equivalence-checker.closure* (σ Σ) (ε Σ) (*Smart-Constructors-Normalization.norm* :: *'a::linorder T rexp* ⇒ *'a T rexp*)
definition *norm-check-eqv* **where** [*code del*]:
norm-check-eqv ≡ λΣ. *equivalence-checker.check-eqv* (σ Σ) (ε Σ) (*Smart-Constructors-Normalization.norm* :: *'a::linorder T rexp* ⇒ *'a T rexp*)
definition *norm-check-eqv-counterexample* **where** [*code del*]:
norm-check-eqv-counterexample ≡ λΣ. *equivalence-checker.check-eqv-counterexample* (σ Σ) (ε Σ) (*Smart-Constructors-Normalization.norm* :: *'a::linorder T rexp* ⇒ *'a T rexp*)

lemmas *norm-defs* = *wf-rexp-def*
norm-check-eqv-def norm-closure-def norm-step-def norm-check-eqv-counterexample-def
norm-check-eqv'-def norm-closure'-def norm-step'-def

interpretation *norm*: *equivalence-checker* σ Σ π ε Σ *Smart-Constructors-Normalization.norm* 𝔏 Σ
where *norm.check-eqv'* = *norm-check-eqv'* Σ
and *norm.check-eqv* = *norm-check-eqv* Σ
and *norm.check-eqv-counterexample* = *norm-check-eqv-counterexample* Σ
and *norm.closure'* = *norm-closure'* Σ
and *norm.closure* = *norm-closure* Σ
and *norm.step'* = *norm-step'* Σ
and *norm.step* = *norm-step* Σ
by *unfold-locales* (*auto simp*: *norm-defs trans*[*OF lang-norm*[*OF iffD2*[*OF ACI-norm-wf*]] *ACI-norm-lang*])

abbreviation *ext* Σ ≡ *None* # *map Some* (Σ :: *'a* :: *linorder list*)

definition *pre-wf-formula* **where** [*code del*]:
pre-wf-formula ≡ λΣ. *formula.pre-wf-formula* (*ext* Σ)
definition *wf-formula* **where** [*code del*]:
wf-formula ≡ λΣ. *formula.wf-formula* (*ext* Σ)
definition *valid-ENC* **where** [*code del*]: *valid-ENC* ≡ λΣ. *formula.valid-ENC* (*ext* Σ)
definition *ENC* **where** [*code del*]: *ENC* ≡ λΣ. *formula.ENC* (*ext* Σ)
definition *rexp-of* **where** [*code del*]: *rexp-of* ≡ λΣ. *formula.rexp-of* (*ext* Σ)
definition *rexp-of-alt* **where** [*code del*]: *rexp-of-alt* ≡ λΣ. *formula.rexp-of-alt* (*ext* Σ)
definition *rexp-of'* **where** [*code del*]: *rexp-of'* ≡ λΣ. *formula.rexp-of'* (*ext* Σ)

lemmas *formula-defs* = *pre-wf-formula-def wf-formula-def rexp-of-def rexp-of′-def rexp-of-alt-def ENC-def valid-ENC-def FOV-def SOV-def*

interpretation Φ: *formula ext* (Σ :: *′a* :: *linorder list*)
where *alphabet.wf* (*set o* σ Σ) = *wf-rexp* Σ
and Φ.*pre-wf-formula* = *pre-wf-formula* Σ
and Φ.*wf-formula* = *wf-formula* Σ
and Φ.*rexp-of* = *rexp-of* Σ
and Φ.*rexp-of-alt* = *rexp-of-alt* Σ
and Φ.*rexp-of′* = *rexp-of′* Σ
and Φ.*valid-ENC* = *valid-ENC* Σ
and Φ.*ENC* = *ENC* Σ
by (*unfold-locales*) (*auto simp*: σ-*def* π-*def wf-rexp-def formula-defs*)

definition *check-eqv* **where**
check-eqv Σ *n* φ ψ $\longleftrightarrow$ *wf-formula* Σ *n* (*FOr* φ ψ) $\wedge$
norm-check-eqv′ (*ext* Σ) *n* (*Plus* (*rexp-of* Σ *n* (*norm* φ)) *One*) (*Plus* (*rexp-of* Σ *n* (*norm* ψ)) *One*)

definition *check-eqv-counterexample* **where**
check-eqv-counterexample Σ *n* φ ψ =
norm-check-eqv-counterexample (*ext* Σ) *n* (*Plus* (*rexp-of* Σ *n* (*norm* φ)) *One*) (*Plus* (*rexp-of* Σ *n* (*norm* ψ)) *One*)

definition *check-eqv′* **where**
check-eqv′ Σ *n* φ ψ $\longleftrightarrow$ Φ.*wf-formula* Σ *n* (*FOr* φ ψ) $\wedge$
norm-check-eqv′ (*ext* Σ) *n* (*Plus* (*rexp-of′* Σ *n* (*norm* φ)) *One*) (*Plus* (*rexp-of′* Σ *n* (*norm* ψ)) *One*)

lemma *lang-Plus-Zero*: $\mathfrak{L}$ Σ *n* (*Plus r One*) = $\mathfrak{L}$ Σ *n* (*Plus s One*) $\longleftrightarrow$ $\mathfrak{L}$ Σ *n r* $-$ {[]} = $\mathfrak{L}$ Σ *n s* $-$ {[]}
by *auto*

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

lemma *soundness*: *check-eqv* Σ *n* φ ψ $\Longrightarrow$ Φ.$lang_{M2L}$ Σ *n* φ = Φ.$lang_{M2L}$ Σ *n* ψ
by (*rule box-equals*[*OF iffD1*[*OF lang-Plus-Zero*, *OF norm.soundness′*] *sym*[*OF trans*[*OF* $lang_{M2L}$-*rexp-of-norm*]] *sym*[*OF trans*[*OF* $lang_{M2L}$-*rexp-of-norm*]]])
(*auto simp*: *check-eqv-def split*: *sum.splits option.splits*)

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

lemma *soundness′*: *check-eqv′* Σ *n* φ ψ $\Longrightarrow$ Φ.$lang_{M2L}$ Σ *n* φ = Φ.$lang_{M2L}$ Σ *n* ψ
by (*rule box-equals*[*OF iffD1*[*OF lang-Plus-Zero*, *OF norm.soundness′*] *sym*[*OF trans*[*OF* $lang_{M2L}$-*rexp-of′-norm*]] *sym*[*OF trans*[*OF* $lang_{M2L}$-*rexp-of′-norm*]]])
(*auto simp*: *check-eqv′-def split*: *sum.splits option.splits*)

lemma *completeness*:
assumes $\Phi.lang_{M2L}\ \Sigma\ n\ \varphi = \Phi.lang_{M2L}\ \Sigma\ n\ \psi$ *wf-formula* Σ *n* (*FOr* φ ψ)
shows *check-eqv* Σ *n* φ ψ
  using *assms*(*2*) **unfolding** *check-eqv-def*
  by (*intro conjI*[*OF assms*(*2*) *norm.completeness'*[*OF iffD2*[*OF lang-Plus-Zero*]],
    OF box-equals[*OF assms*(*1*) $lang_{M2L}$*-rexp-of-norm* $lang_{M2L}$*-rexp-of-norm*]])
  (*auto simp*: *wf-rexp-def*[*symmetric*] *split*: *sum.splits option.splits intro*!: Φ.*wf-rexp-of*)

lemma *completeness'*:
assumes $\Phi.lang_{M2L}\ \Sigma\ n\ \varphi = \Phi.lang_{M2L}\ \Sigma\ n\ \psi$ *wf-formula* Σ *n* (*FOr* φ ψ)
shows *check-eqv'* Σ *n* φ ψ
  using *assms*(*2*) **unfolding** *check-eqv'-def*
  by (*intro conjI*[*OF assms*(*2*) *norm.completeness'*[*OF iffD2*[*OF lang-Plus-Zero*]],
    OF box-equals[*OF assms*(*1*) $lang_{M2L}$*-rexp-of'-norm* $lang_{M2L}$*-rexp-of'-norm*]])
  (*auto simp*: *wf-rexp-def*[*symmetric*] *split*: *sum.splits option.splits intro*!: Φ.*wf-rexp-of'*)

end

10 WS1S

10.1 Encodings

definition *cutSame x s* = *stake* (*LEAST n. sdrop n s* = *same x*) *s*

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

lemma *shift-snth*: (*xs* @− *s*) !! *n* = (*if n* < *length xs then xs* ! *n else s* !! (*n* − *length xs*))
  by *auto*

lemma *stream-map-stream-map2*[*simp*]:
  stream-map f (*stream-map2 g s1 s2*) = *stream-map2* (λ*x y. f* (*g x y*)) *s1 s2*
  unfolding *stream-map2-szip stream.map-comp' o-def split-def* **..**

lemma *stream-map2-alt*:
  (*stream-map2 f s1 s2* = *s*) = ($\forall$*n. f* (*s1* !! *n*) (*s2* !! *n*) = *s* !! *n*)
  unfolding *stream-map2-szip stream-map-alt* **by** *auto*

lemma *snth-stream-map2*[*simp*]:
  stream-map2 f s1 s2 !! *n* = *f* (*s1* !! *n*) (*s2* !! *n*)
  by (*induct n arbitrary*: *s1 s2*) *auto*

lemma *stake-stream-map2*[*simp*]:
  stake n (*stream-map2 f s1 s2*) = *map* (*split f*) (*zip* (*stake n s1*) (*stake n s2*))
  by (*induct n arbitrary*: *s1 s2*) *auto*

lemma *sdrop-stream-map2*[*simp*]:

sdrop n (*stream-map2 f s1 s2*) = *stream-map2 f* (*sdrop n s1*) (*sdrop n s2*)
by (*induct n arbitrary*: *s1 s2*) *auto*

lemma *stake-szip*[*simp*]:
stake n (*szip s1 s2*) = *zip* (*stake n s1*) (*stake n s2*)
by (*induct n arbitrary*: *s1 s2*) *auto*

lemma *sdrop-szip*[*simp*]: *sdrop n* (*szip s1 s2*) = *szip* (*sdrop n s1*) (*sdrop n s2*)
by (*induct n arbitrary*: *s1 s2*) *auto*

lemma *take-stake*: *take n* (*stake m s*) = *stake* (*min n m*) *s*
proof (*induct m arbitrary*: *s n*)
case (*Suc m*) **thus** *?case* **by** (*cases n*) *auto*
qed *simp*

lemma *drop-stake*: *drop n* (*stake m s*) = *stake* (*m* − *n*) (*sdrop n s*)
proof (*induct m arbitrary*: *s n*)
case (*Suc m*) **thus** *?case* **by** (*cases n*) *auto*
qed *simp*

lemma *stream-map-same*[*simp*]: *stream-map f* (*same x*) = *same* (*f x*)
by (*coinduct rule*: *stream.coinduct*[*of* λ*s1 s2*. *s1* = *stream-map f* (*same x*) ∧ *s2* = *same* (*f x*)]) *auto*

lemma (**in** *wellorder*) *min-Least*:
$[\![\exists\, n.\ P\ n;\ \exists\, n.\ Q\ n]\!] \Longrightarrow$ *min* (*Least P*) (*Least Q*) = (*LEAST n*. *P n* ∨ *Q n*)
proof (*intro sym*[*OF Least-equality*])
fix *y* **assume** *P y* ∨ *Q y*
thus *min* (*Least P*) (*Least Q*) ≤ *y*
proof (*elim disjE*)
assume *P y*
hence *Least P* ≤ *y* **by** (*auto intro*: *LeastI2-wellorder*)
thus *min* (*Least P*) (*Least Q*) ≤ *y* **unfolding** *min-def* **by** *auto*
next
assume *Q y*
hence *Least Q* ≤ *y* **by** (*auto intro*: *LeastI2-wellorder*)
thus *min* (*Least P*) (*Least Q*) ≤ *y* **unfolding** *min-def* **by** *auto*
qed
qed (*metis LeastI-ex min-def*)

lemma *sdrop-snth*: *sdrop n s* !! *m* = *s* !! (*n* + *m*)
by (*induct n arbitrary*: *m s*) *auto*

context *formula*
begin

definition *any* ≡ *hd* Σ

lemma *any*-Σ[*simp*]: *any* ∈ *set* Σ

unfolding *any-def* **by** (*auto simp*: *nonempty intro*: *someI*[*of* - *hd* Σ])

lemma *enc-atom-any-σ*[*simp*]: *length I* = *n* ⟹ *enc-atom I m any* ∈ *set* (σ Σ *n*)
by (*auto simp*: *σ-def image-iff set-n-lists*)

fun *stream-enc* :: *'a interp* ⇒ (*'a* × *bool list*) *stream* **where**
stream-enc (*w*, *I*) = *stream-map2* (*enc-atom I*) *nats* (*w* @− *same any*)

lemma *tl-stream-enc*[*simp*]: *stream-map* π (*stream-enc* (*w*, *x* # *I*)) = *stream-enc* (*w*, *I*)
by (*auto simp*: *comp-def π-def*)

lemma *enc-atom-max*: ⟦∀ *x*∈*set I*. *case x of Inl p* ⇒ *p* ≤ *n* | *Inr P* ⇒ ∀ *p*∈*P*. *p* ≤ *n*; *n* ≤ *n'*⟧ ⟹
enc-atom I (*Suc n'*) *a* = (*a*, *replicate* (*length I*) *False*)
by (*induct I*) (*auto split*: *sum.splits*)

lemma *ex-Loop-stream-enc*:
assumes ∀ *x* ∈ *set I*. *case x of Inr P* ⇒ *finite P* | - ⇒ *True*
shows ∃ *n*. *sdrop n* (*stream-enc* (*w*, *I*)) = *same* (*any*, *replicate* (*length I*) *False*)
proof −
from *assms* **have** ∃ *n* > *length w*. ∀ *x*∈*set I*. *case x of Inl p* ⇒ *p* ≤ *n* | *Inr P* ⇒ ∀ *p*∈*P*. *p* ≤ *n*
proof (*induct I*)
case (*Cons x I*)
then obtain *n* **where** *IH*: *length w* < *n*
∀ *x*∈*set I*. *case x of Inl p* ⇒ *p* ≤ *n* | *Inr P* ⇒ ∀ *p*∈*P*. *p* ≤ *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 guess *n* **by** (*elim exE conjE*)
hence *sdrop* (*Suc n*) (*stream-enc* (*w*, *I*)) = *same* (*any*, *replicate* (*length I*) *False*)
(**is** *?s1 n* = *?s2*)
by (*coinduct rule*: *stream.coinduct*[*of* λ*s1 s2*. ∃ *n'*≥ *n*. *s1* = *?s1 n'* ∧ *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 *stream-set-same*: $[\![y \in \textit{stream-set } s;\ s = \textit{same } x]\!] \Longrightarrow y = x$
  by (*induct rule*: *stream-set-induct1*) *auto*

lemma *same-alt*: $s = \textit{same } x \longleftrightarrow \textit{stream-set } s = \{x\}$
  using *stream-set-same*[*of - same x x*]
  apply *auto*
  apply (*metis shd-stream-set*)
  apply (*coinduct rule*: *stream.coinduct*[*of* $\lambda s1\ s2.\ s2 = \textit{same } x \wedge \textit{stream-set } s1 = \{x\}$])
  apply *auto*
  apply (*metis shd-stream-set singleton-iff*)
  apply (*metis stl-stream-set singleton-iff*)
  by (*metis* (*full-types*) *empty-iff insert-iff shd-stream-set stl-stream-set*)

lemma *sdrop-sameE*: $[\![\textit{sdrop } n\ (w\ @-\ \textit{same } y) = \textit{same } y;\ p < \textit{length } w;\ \neg\ p < n]\!] \Longrightarrow w\ !\ p = y$
  unfolding *not-less same-alt*
  apply (*induct p arbitrary*: *w n*)
  apply *simp*
  apply (*metis hd-conv-nth shd-stream-set shift-simps*(*1*) *singletonE stream-set-shift*)
  apply (*case-tac w*)
  apply *simp-all*
  apply (*case-tac n*)
  apply *simp-all*
  by (*metis equals0D nth-mem singletonE subset-singletonD*)

lemma *less-length-cutSame*:
  $[\![(w\ @-\ \textit{same } y)\ !!\ p = a]\!] \Longrightarrow a = y \vee (p < \textit{length } (\textit{cutSame } y\ (w\ @-\ \textit{same } y)) \wedge w\ !\ p = a)$
  unfolding *cutSame-def length-stake*
  by (*rule LeastI2-ex*[*OF exI*[*of - length w*]])
   (*auto simp*: *sdrop-shift shift-snth split*: *split-if-asm elim*: *sdrop-sameE*)

lemma *less-length-cutSame-Inl*:
  $[\![(\forall x \in \textit{set } I.\ \textit{case } x \textit{ of Inr } P \Rightarrow \textit{finite } P \mid \text{-} \Rightarrow \textit{True});\ r < \textit{length } I;\ I\ !\ r = \textit{Inl } p]\!] \Longrightarrow$
  $p < \textit{length } (\textit{cutSame } (\textit{any}, \textit{replicate } (\textit{length } I)\ \textit{False})\ (\textit{stream-enc } (w, I)))$
  unfolding *cutSame-def length-stake*
  by (*erule LeastI2-ex*[*OF ex-Loop-stream-enc ccontr*])
   (*auto simp*: *stream-map2-alt dest*!: *add-diff-inverse*,
   metis (*lifting*, *full-types*) *nth-map nth-replicate sum.simps*(*5*))

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

by (*rule ballI*, *erule LeastI2-ex*[*OF ex-Loop-stream-enc ccontr*])
(*auto simp*: *stream-map2-alt dest*!: *add-diff-inverse*,
metis nth-map nth-replicate sum.simps(*6*))

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

lemma *cutSame-all*[*simp*]: *cutSame x* (*same x*) = []
unfolding *cutSame-def* **by** (*auto intro*: *Least-equality*)

lemma *cutSame-stop*[*simp*]:
assumes $x \neq y$
shows *cutSame x* (*xs* @− *Stream y* (*same x*)) = *xs* @ [*y*] (**is** *cutSame x ?s* = -)
proof −
have (*LEAST n*. *sdrop n ?s* = *same x*) = *Suc* (*length xs*)
proof (*rule Least-equality*)
show *sdrop* (*Suc* (*length xs*)) *?s* = *same x*
by (*metis sdrop-shift sdrop-simps*(*2*) *stream.sels*(*2*))
next
fix *m* **assume** ∗: *sdrop m ?s* = *same x*
{ **assume** *m* < *Suc* (*length xs*)
hence $m \leq$ *length xs* **by** *simp*
then obtain *ys* **where** *sdrop m ?s* = *ys* @− *Stream y* (*same x*)
by *atomize-elim* (*induct m arbitrary*: *xs*, *auto*)
with ∗ **obtain** *ys* @− *Stream y* (*same x*) = *same x* **by** *simp*
hence *Stream y* (*same x*) = *same x* **by** (*metis sdrop-same sdrop-shift*)
with *assms* **have** *False* **by** (*metis same-simps*(*1*) *stream.sels*(*1*))
}
thus *Suc* (*length xs*) $\leq$ *m* **by** (*blast intro*: *leI*)
qed
thus *?thesis* **unfolding** *cutSame-def*
by (*metis length-append-singleton shift.simps shift-append stake-shift*)
qed

lemma *cutSame-shift-same*: $\exists$ *n*. *w* = *cutSame x* (*w* @− *same x*) @ *replicate n x*
proof (*induct w rule*: *rev-induct*)
case (*snoc a w*)
then obtain *n* **where** *w* = *cutSame x* (*w* @− *same x*) @ *replicate n x* **by** *blast*
thus *?case*
by (*cases a* = *x*)
(*auto simp*: *same-unfold*[*symmetric*] *replicate-append-same*[*symmetric*] *intro*!: *exI*[*of* - *Suc n*])
qed *simp*

lemma *set-cutSame*: *set* (*cutSame x* (*w* @− *same x*)) $\subseteq$ *set w*
proof (*induct w rule*: *rev-induct*)
case (*snoc a w*)

thus *?case* **by** (*cases a* = *x*) (*auto simp*: *same-unfold*[*symmetric*])
qed *simp*

lemma *stream-enc-cutSame*:
assumes ($\forall x \in$ *set I*. *case x of Inr P* $\Rightarrow$ *finite P* | - $\Rightarrow$ *True*)
shows *stream-enc* (*w*, *I*) = *cutSame* (*any*, *replicate* (*length I*) *False*) (*stream-enc* (*w*, *I*)) @−
same (*any*, *replicate* (*length I*) *False*)
unfolding *cutSame-def*
by (*rule trans*[*OF sym*[*OF stake-sdrop*] *arg-cong2*[*of - - - - op* @−, *OF refl*]])
(*rule LeastI-ex*[*OF ex-Loop-stream-enc*[*OF assms*]])

lemma *map-fst-zip-min*[*simp*]: *map fst* (*zip xs ys*) $\equiv$ *take* (*min* (*length xs*) (*length ys*)) *xs*
proof (*induct ys arbitrary*: *xs*)
case *Cons* **thus** *?case* **by** (*case-tac xs*) *auto*
qed *simp*

lemma *map-snd-zip-min*[*simp*]: *map snd* (*zip xs ys*) $\equiv$ *take* (*min* (*length xs*) (*length ys*)) *ys*
proof (*induct ys arbitrary*: *xs*)
case *Cons* **thus** *?case* **by** (*case-tac xs*) *auto*
qed *simp*

lemma *stream-enc-enc*:
assumes ($\forall x \in$ *set I*. *case x of Inr P* $\Rightarrow$ *finite P* | - $\Rightarrow$ *True*) **and** *v*: *v* $\in$ *enc* (*w*, *I*)
shows *stream-enc* (*w*, *I*) = *v* @− *same* (*any*, *replicate* (*length I*) *False*)
(**is** *?s* = *?v* @− *same ?F*)
proof −
from *assms*(*1*) **obtain** *n* **where** *sdrop n* (*stream-enc* (*w*, *I*)) = *same ?F* **by** (*metis ex-Loop-stream-enc*)
moreover from *v* **obtain** *m* **where** *?v* = *cutSame ?F ?s* @ *replicate m ?F* **by** *auto*
ultimately show *?s* = *v* @− *same ?F*
by (*auto simp del*: *stream-enc.simps intro*: *stream-enc-cutSame*[*OF assms*(*1*)])
qed

lemma *stream-enc-enc-some*:
assumes ($\forall x \in$ *set I*. *case x of Inr P* $\Rightarrow$ *finite P* | - $\Rightarrow$ *True*)
shows *stream-enc* (*w*, *I*) = (*SOME v*. *v* $\in$ *enc* (*w*, *I*)) @− *same* (*any*, *replicate* (*length I*) *False*)
by (*rule stream-enc-enc*[*OF assms*], *rule someI-ex*) *auto*

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

lemma *sdrop-same*: *sdrop n s* = *same x* $\Longrightarrow$ $n \leq m$ $\Longrightarrow$ *s* !! *m* = *x* by (*metis le-iff-add sdrop-snth snth-same*)

lemma *fin-cutSame-tl*:
assumes $\exists\, n.$ *sdrop n s* = *same x*
shows *fin-cutSame* (π *x*) (*map* π (*cutSame x s*)) = *cutSame* (π *x*) (*stream-map* π *s*)
proof −
def *min* $\equiv$ *LEAST n. sdrop n s* = *same x*
from *assms* **have** *min*: *sdrop min s* = *same x* $\bigwedge$*m. sdrop m s* = *same x* $\Longrightarrow$ *min* $\leq$ *m*
unfolding *min-def* **by** (*auto intro*: *LeastI Least-le*)
have *Ex*: $\exists\, n.$ *drop n* (*map* π (*stake min s*)) = *replicate* (*length* (*map* π (*stake min s*)) − *n*) (π *x*)
by (*auto intro*: *exI*[*of - length* (*map* π (*stake min s*))])
have *fin-cutSame* (π *x*) (*map* π (*cutSame x s*)) =
map π (*stake* (*LEAST n.*
map π (*stake* (*min* − *n*) (*sdrop n s*)) = *replicate* (*min* − *n*) (π *x*) $\vee$ *sdrop n s* = *same x*) *s*)
unfolding *fin-cutSame-def cutSame-def take-map take-stake min-Least*[*OF Ex assms*, *folded min-def*]
min-def[*symmetric*] **by** (*auto simp*: *drop-map drop-stake*)
also have (λ*n. map* π (*stake* (*min* − *n*) (*sdrop n s*)) = *replicate* (*min* − *n*) (π *x*) $\vee$ *sdrop n s* = *same x*) =
(λ*n. stream-map* π (*sdrop n s*) = *same* (π *x*))
proof (*rule ext*, *unfold stream-map-alt snth-same*, *safe*)
fix *n m*
assume *map* π (*stake* (*min* − *n*) (*sdrop n s*)) = *replicate* (*min* − *n*) (π *x*)
hence $\forall$ *y*$\in$*set* (*stake* (*min* − *n*) (*sdrop n s*)). π *y* = π *x*
by (*intro iffD1*[*OF map-eq-conv*]) (*metis length-stake map-replicate-const*)
hence $\forall$ *i*<*min* − *n*. π (*sdrop n s* !! *i*) = π *x*
unfolding *all-set-conv-all-nth* **by** (*auto simp*: *sdrop-snth*)
thus π (*sdrop n s* !! *m*) = π *x*
proof (*cases m* < *min* − *n*)
case *False*
hence *min* $\leq$ *n* + *m* **by** *linarith*
hence *sdrop n s* !! *m* = *x* **unfolding** *sdrop-snth* **by** (*rule sdrop-same*[*OF min*(*1*)])
thus *?thesis* **by** *simp*
qed *auto*
next
fix *n*
assume $\forall$ *m*. π (*sdrop n s* !! *m*) = π *x*
thus *map* π (*stake* (*min* − *n*) (*sdrop n s*)) = *replicate* (*min* − *n*) (π *x*)
unfolding *stake-stream-map*[*symmetric*] **by** (*metis snth-same stake-same stream-map-alt*)
qed *auto*
finally show *?thesis* **unfolding** *cutSame-def sdrop-stream-map stake-stream-map*
.
qed

lemma *tl-enc*[*simp*]:
  assumes $\forall x \in$ *set* $(x \# I)$. *case x of Inr P* $\Rightarrow$ *finite P* | - $\Rightarrow$ *True*
  shows *SAMEQUOT* (*any*, *replicate* (*length I*) *False*) (*map* π ' *enc* (*w*, *x* # *I*)) = *enc* (*w*, *I*)
  unfolding *SAMEQUOT-def*
  by (*fastforce simp*: *assms* π*-def*
    fin-cutSame-tl[*OF ex-Loop-stream-enc*[*OF assms*], *unfolded* π*-def*, *simplified*, *symmetric*])

lemma *encD*:
  $[\![v \in$ *enc* (*w*, *I*); ($\forall x \in$ *set I*. *case x of Inr P* $\Rightarrow$ *finite P* | - $\Rightarrow$ *True*)$]\!] \Longrightarrow$
  v = *map* (*split* (*enc-atom I*)) (*zip* [*0* ..< *length v*] (*stake* (*length v*) (*w* @− *same 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*] *simp del*: *stake-add*)

lemma *enc-Inl*: $[\![x \in$ *enc* (*w*, *I*); ($\forall x \in$ *set I*. *case x of Inr P* $\Rightarrow$ *finite P* | - $\Rightarrow$ *True*);
  m < *length I*; *I* ! *m* = *Inl p*$]\!] \Longrightarrow$ *p* < *length x* $\wedge$ *snd* (*x* ! *p*) ! *m*
  by (*auto dest*!: *less-length-cutSame-Inl*[*of* - - - *w*] *simp*: *nth-append cutSame-def*)

lemma *enc-Inr*: **assumes** $x \in$ *enc* (*w*, *I*) $\forall x \in$ *set I*. *case x of Inr P* $\Rightarrow$ *finite P* | - $\Rightarrow$ *True*
  M < *length I I* ! *M* = *Inr P*
  shows $p \in P \longleftrightarrow p <$ *length x* $\wedge$ *snd* (*x* ! *p*) ! *M*
proof
  assume $p \in P$ **with** *assms* **show** *p* < *length x* $\wedge$ *snd* (*x* ! *p*) ! *M*
   by (*auto dest*!: *less-length-cutSame-Inr*[*of* - - - *w*] *simp*: *nth-append cutSame-def*)
next
  assume *p* < *length x* $\wedge$ *snd* (*x* ! *p*) ! *M*
  thus $p \in P$ **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* = λ*w I*. *cutSame* (*any*, *replicate* (*length I*) *False*) (*stream-enc* (*w*, *I*))
  let *?w* = λ*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*) *?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*:
  $[\![(\forall x \in set\ I.\ case\ x\ of\ Inr\ P \Rightarrow finite\ P \mid \text{-} \Rightarrow True);$
  $(\forall x \in set\ I'.\ case\ x\ of\ Inr\ P \Rightarrow finite\ P \mid \text{-} \Rightarrow True);$
  $enc\ (w,\ I) = enc\ (w',\ I')]\!] \Longrightarrow stream\text{-}enc\ (w,\ I) = stream\text{-}enc\ (w',\ I')$
 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*)

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

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

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

lemma *encD-ex*: $[\![x \in enc\ (w,\ I);\ (\forall x \in set\ I.\ case\ x\ of\ Inr\ P \Rightarrow finite\ P \mid \text{-} \Rightarrow True)]\!] \Longrightarrow$
 $\exists n.\ x = map\ (split\ (enc\text{-}atom\ I))\ (zip\ [0\ ..<\ n]\ (stake\ n\ (w\ @-\ same\ any)))$
 by (*auto dest*!: *encD simp del*: *enc.simps*)

lemma *enc-set-σ*: $[\![x \in enc\ (w,\ I);\ (\forall x \in set\ I.\ case\ x\ of\ Inr\ P \Rightarrow finite\ P \mid \text{-} \Rightarrow True);$
 $length\ I = n;\ a \in set\ x;\ set\ w \subseteq set\ \Sigma]\!] \Longrightarrow a \in set\ (\sigma\ \Sigma\ n)$
 apply (*auto dest*!: *encD-ex simp*: *in-set-zip simp del*: *enc.simps*)
 apply (*case-tac na* < *length w*)
 apply (*auto intro*!: *enc-atom-σ*)
 done

definition *positions-in-row s i* =
 Option.these (*stream-set* (*stream-map2* (λ*p* (-, *bs*). *if nth bs i then Some p else None*) *nats s*))

lemma *positions-in-row*: *positions-in-row s i* = {*p*. *snd* (*s* !! *p*) ! *i*}
  unfolding *positions-in-row-def these-def stream-map2-szip stream.set-natural′ stream-set-range*
  by (*auto split*: *split-if-asm intro*!: *image-eqI*[*of* - *the*] *split*: *prod.splits*)

lemma *positions-in-row-unique*: $\exists !$*p*. *snd* (*s* !! *p*) ! *i* $\Longrightarrow$
  the-elem (*positions-in-row s i*) = (*THE p*. *snd* (*s* !! *p*) ! *i*)
  by (*rule the1I2*) (*auto simp*: *the-elem-def positions-in-row*)

lemma *positions-in-row-nth*: $\exists !$*p*. *snd* (*s* !! *p*) ! *i* $\Longrightarrow$
  snd (*s* !! *the-elem* (*positions-in-row s i*)) ! *i*
  unfolding *positions-in-row-unique* **by** (*rule the1I2*) *auto*

definition *dec-word s* = *cutSame any* (*stream-map fst s*)

lemma *dec-word-stream-enc*: *dec-word* (*stream-enc* (*w*, *I*)) = *cutSame any* (*w* @− *same any*)
  unfolding *dec-word-def* **by** (*auto intro*!: *arg-cong*[*of* - - *cutSame any*] *simp*: *stream-map2-alt*)

definition *stream-dec n FO s* = *map* (λ*i*.
  if i $\in$ *FO*
  then Inl (*the-elem* (*positions-in-row s i*))
  else Inr (*positions-in-row s i*)) [*0*..<*n*]

lemma *stream-dec-Inl*: $[\![$*i* $\in$ *FO*; *i* < *n*$]\!]$ $\Longrightarrow$ $\exists$ *p*. *stream-dec n FO s* ! *i* = *Inl p*
  unfolding *stream-dec-def* **using** *nth-map*[*of n* [*0*..<*n*]] **by** *auto*

lemma *stream-dec-not-Inr*: $[\![$*stream-dec n FO s* ! *i* = *Inr P*; *i* $\in$ *FO*; *i* < *n*$]\!]$ $\Longrightarrow$ *False*
  unfolding *stream-dec-def* **using** *nth-map*[*of n* [*0*..<*n*]] **by** *auto*

lemma *stream-dec-Inr*: $[\![$*i* $\notin$ *FO*; *i* < *n*$]\!]$ $\Longrightarrow$ $\exists$ *P*. *stream-dec n FO s* ! *i* = *Inr P*
  unfolding *stream-dec-def* **using** *nth-map*[*of n* [*0*..<*n*]] **by** *auto*

lemma *stream-dec-not-Inl*: $[\![$*stream-dec n FO s* ! *i* = *Inl p*; *i* $\notin$ *FO*; *i* < *n*$]\!]$ $\Longrightarrow$ *False*
  unfolding *stream-dec-def* **using** *nth-map*[*of n* [*0*..<*n*]] **by** *auto*

lemma *Inr-dec-finite*: $[\![\forall$ *i*<*n*. *finite* {*p*. *snd* (*s* !! *p*) ! *i*}; *Inr P* $\in$ *set* (*stream-dec n FO s*)$]\!]$ $\Longrightarrow$
  finite P
  unfolding *stream-dec-def* **by** (*auto simp*: *positions-in-row*)

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

by (*rule sym*, *subst surjective-pairing*[*of s* !! *p*])
(*auto intro*!: *nth-equalityI simp*: *positions-in-row simp del*: *pair-collapse split*: *split-if-asm*,
(*metis positions-in-row positions-in-row-nth*)+)

lemma *length-stream-dec*[*simp*]: *length* (*stream-dec n FO x*) = *n*
unfolding *stream-dec-def* **by** *auto*

lemma *stream-enc-dec*:
$[\![\exists\, n.$ *sdrop n* (*stream-map fst s*) = *same any*;
stream-all (λx. *length* (*snd x*) = *n*) *s*; $\forall\, i \in$ *FO*. ($\exists$!*p*. *snd* (*s* !! *p*) ! *i*)$]\!] \Longrightarrow$
stream-enc (*dec-word s*, *stream-dec n FO s*) = *s*
unfolding *dec-word-def snth-fromN*
by (*drule LeastI-ex*)
(*auto intro*!: *enc-atom-dec simp*: *stream-map2-alt cutSame-def*
simp del: *stake-stream-map sdrop-stream-map*
intro!: *trans*[*OF arg-cong2*[*of - - - - op* !!] *snth-stream-map*]
trans[*OF arg-cong2*[*of - - - - op* @−] *stake-sdrop*])

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

lemma *stream-dec-enc-Inl*:
$[\![$*stream-dec n FO* (*stream-enc* (*w*, *I*)) ! *i* = *Inl p′*; *I* ! *i* = *Inl p*; *i* ∈ *FO*; *i* < *n*; *length I* = *n*$]\!] \Longrightarrow$
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*:
$[\![$*stream-dec n FO* (*stream-enc* (*w*, *I*)) ! *i* = *Inr P′*; *I* ! *i* = *Inr P*; *i* ∉ *FO*; *i* < *n*; *length I* = *n*$]\!] \Longrightarrow$
P = *P′*
unfolding *stream-dec-def positions-in-row* **by** *auto*

lemma *Collect-snth*: {*p*. *P* (*Stream x s* !! *p*)} ⊆ {*0*} ∪ *Suc* ‘ {*p*. *P* (*s* !! *p*)}
unfolding *image-def* **by** (*auto simp*: *gr0-conv-Suc*)

lemma *finite-True-in-row*: ∀ *i* < *n*. *finite* {*p*. *snd* ((*w* @− *same* (*any*, *replicate n False*)) !! *p*) ! *i*}
by (*induct w*) (*auto intro*: *finite-subset*[*OF Collect-snth*])

lemma *lang-ENC*:
assumes *wf-formula n* φ
shows *lang n* (*ENC n* φ) = $\bigcup$ {*enc* (*w*, *I*) | *w I* . *length I* = *n* ∧ *wf-interp-for-formula* (*w*, *I*) φ}
(**is** *?L* = *?R*)

proof (*intro equalityI subsetI*)
 fix x **assume** L: $x \in ?L$
 hence $*$: *set* $x \subseteq$ *set* $(\sigma\ \Sigma\ n)$ **using** *wf-lang-wf-word*[*OF wf-rexp-ENC*] **by** (*auto simp*: *wf-word*)
 let $?s = x$ @− *same* (*any*, *replicate n False*)
 have *list-all* (λbs. *length* (*snd bs*) $= n$) x
  using *bspec*[*OF wf-lang-wf-word*[*OF wf-rexp-ENC*], *OF* ‹$x \in ?L$›]
  by (*auto simp*: *list-all-iff wf-word*) (*auto simp*: σ*-def set-n-lists*)
 hence *stream-all* (λx. *length* (*snd* x) $= n$) (x @− *same* (*any*, *replicate n False*))
  by (*auto simp only*: *stream-all-shift stream-all-same length-replicate snd-conv*)
 moreover
 { fix m **assume** $m \in FOV\ \varphi$
  with *assms* **have** $m < n$ **by** (*auto simp*: *max-idx-vars*)
  with L ‹$m \in FOV\ \varphi$› **obtain** $u\ z\ v$ **where** *uzv*: $x = u$ @ z @ v
   $u \in$ *star* (*lang n* (*arbitrary-except n* [(m, *False*)] Σ))
   $z \in$ *lang n* (*arbitrary-except n* [(m, *True*)] Σ)
   $v \in$ *star* (*lang n* (*arbitrary-except n* [(m, *False*)] Σ)) **unfolding** *ENC-def*
  by (*auto simp*: *wf-rexp-valid-ENC finite-FOV dest*!: *iffD1*[*OF lang-flatten-INTERSECT*, *rotated* −*1*])
    (*fastforce simp*: *valid-ENC-def*)
  with ‹$m < n$› **have** $\exists!p$. *snd* (x ! p) ! $m \wedge p <$ *length* x
  proof (*intro ex1I*[*of - length u*])
   fix p **assume** $m < n$ *snd* (x ! p) ! $m \wedge p <$ *length* x
  with *star-arbitrary-except*[*OF uzv*(*2*)] *arbitrary-except*[*OF uzv*(*3*)] *star-arbitrary-except*[*OF uzv*(*4*)]
    show $p =$ *length* u **by** (*cases rule*: *nat-less-cases*) (*auto simp*: *nth-append uzv*(*1*))
  qed (*auto dest*!: *arbitrary-except*)
  then obtain p **where** p: $p <$ *length* x *snd* (x ! p) ! m
   $\bigwedge q$. *snd* (x ! q) ! $m \wedge q <$ *length* $x \longrightarrow q = p$ **by** *auto*
  hence $\exists!p$. *snd* ($?s$!! p) ! m
  proof (*intro ex1I*[*of - p*])
   fix q **from** p ‹$m < n$› **show** *snd* ($?s$!! q) ! $m \Longrightarrow q = p$ **by** (*cases* $q <$ *length* x) *auto*
  qed *auto*
 }
 moreover have *sdrop* (*length* x) (*stream-map fst* (x @− *same* (*any*, *replicate n False*))) $=$ *same any*
  unfolding *sdrop-stream-map* **by** (*subst sdrop-shift*[*OF refl refl*]) *simp*
 ultimately have *enc-dec*: *stream-enc* (*dec-word* $?s$, *stream-dec n* (*FOV* φ) $?s$) $=$
  x @− *same* (*any*, *replicate n False*) **by** (*intro stream-enc-dec*) *auto*
 def $I \equiv$ *stream-dec n* (*FOV* φ) $?s$
 with *assms* **have** *wf-interp-for-formula* (*dec-word* $?s$, I) φ **unfolding** *I-def dec-word-def*
  by (*auto dest*: *stream-dec-not-Inr stream-dec-not-Inl simp* :σ*-def max-idx-vars dest*!: *set-mp*[*OF set-cutSame*[*of any map fst x*]] *set-mp*[*OF* $*$] *split*: *sum.splits*)
   (*auto simp*: *stream-dec-def positions-in-row finite-True-in-row*)
 moreover have *length* $I = n$ **unfolding** *I-def* **by** *simp*

moreover have $x \in$ *enc* (*dec-word* ?*s*, *I*) **unfolding** *I-def*
by (*simp add*: *enc-dec cutSame-shift-same del*: *stream-enc.simps*)
ultimately show $x \in$?*R* **by** *blast*
next
fix x **assume** $x \in$?*R*
then obtain *w I* **where** *I*: $x \in$ *enc* (*w*, *I*) *wf-interp-for-formula* (*w*, *I*) φ *length* $I = n$ **by** *blast*
{ fix *i* **from** *I*(*2*) **have** (*w* @− *same any*) !! *i* $\in$ *set* Σ **by** (*cases* $i <$ *length w*) *auto* **} note** $*$ = *this*
from *I* **have** *x* @− *same* (*any*, *replicate* (*length I*) *False*) = *stream-enc* (*w*, *I*) (**is** *x* @− ?*F* = ?*s*)
by (*intro stream-enc-enc*[*symmetric*]) *auto*
with $*$ ‹*length I* = *n*› **have** $\forall x \in$ *set x*. *length* (*snd x*) = *n* $\wedge$ *fst x* $\in$ *set* Σ
by (*auto dest*!: *shift-snth-less*[*of* - - ?*F*, *symmetric*] *simp*: *in-set-conv-nth*)
thus $x \in$?*L*
proof (*cases FOV* φ = {})
case *False*
hence *nonempty*: *valid-ENC n* ‘ *FOV* $\varphi \neq$ {} **by** *simp*
have *finite*: *finite* (*valid-ENC n* ‘ *FOV* φ) **by** (*rule finite-imageI*[*OF finite-FOV*])
from *False assms*(*1*) **have** $0 < n$ **by** (*cases n*) (*auto split*: *dest*!: *max-idx-vars*)
with *wf-rexp-valid-ENC* **have** *wf-rexp*: $\forall x \in$ *valid-ENC n* ‘ *FOV* φ. *wf n x* **by** *auto*
{ fix *r* **assume** $r \in$ *FOV* φ
with *I*(*2*) **obtain** *p* **where** *p*: *I* ! *r* = *Inl p* **by** (*cases I* ! *r*) *auto*
from ‹$r \in$ *FOV* φ› *assms I*(*2*,*3*) **have** *r*: $r <$ *length I* **by** (*auto dest*!: *max-idx-vars*)
from *p I*(*1*,*2*) *r* **have** $p <$ *length x*
using *less-length-cutSame-Inl*[*of I r p w*] **by** *auto*
with *p I r* $*$ **have** [*x* ! *p*] $\in$ *lang n* (*arbitrary-except n* [(*r*, *True*)] Σ)
by (*subst encD*[*of x*]) (*auto intro*!: *enc-atom-lang-arbitrary-except-True*)
moreover
from *p I r* $*$ **have** *take p x* $\in$ *star* (*lang n* (*arbitrary-except n* [(*r*, *False*)] Σ))
by (*subst encD*[*of x*]) (*auto simp*: *in-set-conv-nth intro*!: *Ball-starI enc-atom-lang-arbitrary-except-False*)
moreover
from *p I r* $*$ **have** *drop* (*Suc p*) *x* $\in$ *star* (*lang n* (*arbitrary-except n* [(*r*, *False*)] Σ))
by (*subst encD*[*of x*]) (*auto simp*: *in-set-conv-nth simp del*: *snth.simps intro*!: *Ball-starI enc-atom-lang-arbitrary-except-False*)
ultimately have *take p x* @ [*x* ! *p*] @ *drop* (*p* + *1*) *x* $\in$ *lang n* (*valid-ENC n r*)
using ‹$0 < n$› **unfolding** *valid-ENC-def* **by** (*auto simp del*: *append.simps*)
hence $x \in$ *lang n* (*valid-ENC n r*) **using** *id-take-nth-drop*[*OF* ‹$p <$ *length x*›] **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*: σ-*def set-n-lists image-iff*)
qed

10.2 Welldefinedness of enc wrt. Models

lemma *wf-interp-for-formula-FExists*:
$[\![$*wf-formula* (*length* I) (*FExists* φ)$]\!] \Longrightarrow$
wf-interp-for-formula (w, I) (*FExists* φ) $\longleftrightarrow$ ($\forall$ p. *wf-interp-for-formula* (w, *Inl* p # I) φ)
by (*auto simp*: *nth-Cons*′ *split*: *split-if-asm*)

lemma *wf-interp-for-formula-any-Inl*: *wf-interp-for-formula* (w, *Inl* p # I) $\varphi \Longrightarrow$
$\forall$ p. *wf-interp-for-formula* (w, *Inl* p # I) φ
by (*auto simp*: *nth-Cons*′ *split*: *split-if-asm*)

lemma *wf-interp-for-formula-FEXISTS*:
$[\![$*wf-formula* (*length* I) (*FEXISTS* φ)$]\!] \Longrightarrow$
wf-interp-for-formula (w, I) (*FEXISTS* φ) $\longleftrightarrow$ ($\forall$ P. *finite* P $\longrightarrow$ *wf-interp-for-formula* (w, *Inr* P # I) φ)
by (*auto simp*: *nth-Cons*′ *split*: *split-if-asm*)

lemma *wf-interp-for-formula-any-Inr*: *wf-interp-for-formula* (w, *Inr* P # I) $\varphi \Longrightarrow$
$\forall$ P. *finite* P $\longrightarrow$ *wf-interp-for-formula* (w, *Inr* P # I) φ
by (*auto simp*: *nth-Cons*′ *split*: *split-if-asm*)

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

lemma *wf-interp-for-formula-FAnd*:
wf-interp-for-formula (w, I) (*FAnd* φ*1* φ*2*) =
(*wf-interp-for-formula* (w, I) φ*1* $\wedge$ *wf-interp-for-formula* (w, I) φ*2*)
by *auto*

lemma *enc-wf-interp*:
$[\![$*wf-formula* (*length* I) φ; *wf-interp-for-formula* (w, I) φ; $x \in$ *enc* (w, I)$]\!] \Longrightarrow$
wf-interp-for-formula (*dec-word* (x @− *same* (*any*, *replicate* (*length* I) *False*)),
stream-dec (*length* I) (*FOV* φ) (x @− *same* (*any*, *replicate* (*length* I) *False*)))
φ
using
stream-dec-Inl[*of* - *FOV* φ *length* I *stream-enc* (w, I), *OF* - *bspec*[*OF max-idx-vars*]]
stream-dec-Inr[*of* - *FOV* φ *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 set-mp*[*OF set-cutSame*] *simp del*: *stream-enc.simps*)
(*auto simp*: *cutSame-def in-set-zip stream-map2-alt shift-snth*)

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

proof (*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*: ⟦*stream-enc* (*w*, *I*) = *stream-enc* (*w′*, *I′*); *wf-formula* (*length I*) *φ*;
  wf-interp-for-formula (*w*, *I*) *φ*; *wf-interp-for-formula* (*w′*, *I′*) *φ*⟧ ⟹
  (*w*, *I*) ⊨ *φ* ⟷ (*w′*, *I′*) ⊨ *φ*
proof (*induction φ 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*: *stream-map2-alt shift-snth*) (*metis snth-same*)
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*: *stream-map2-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*: *stream-map2-alt*)
next
  case (*FOr φ1 φ2*) **show** *?case* **unfolding** *satisfies.simps*(*5*)
  proof (*intro disj-cong*)
    from *FOr*(*3*−*6*) **show** (*w*, *I*) ⊨ *φ1* ⟷ (*w′*, *I′*) ⊨ *φ1*
      by (*intro FOr*(*1*)) *auto*
  next
    from *FOr*(*3*−*6*) **show** (*w*, *I*) ⊨ *φ2* ⟷ (*w′*, *I′*) ⊨ *φ2*
      by (*intro FOr*(*2*)) *auto*
  qed
next
  case (*FAnd φ1 φ2*) **show** *?case* **unfolding** *satisfies.simps*(*6*)
  proof (*intro conj-cong*)
    from *FAnd*(*3*−*6*) **show** (*w*, *I*) ⊨ *φ1* ⟷ (*w′*, *I′*) ⊨ *φ1*
      by (*intro FAnd*(*1*)) *auto*
  next
    from *FAnd*(*3*−*6*) **show** (*w*, *I*) ⊨ *φ2* ⟷ (*w′*, *I′*) ⊨ *φ2*
      by (*intro FAnd*(*2*)) *auto*
  qed
next
  case (*FExists φ*)
  hence *length*: *length I′* = *length I* **by** (*metis length-snth-enc*)
  show *?case*
  proof

assume $(w, I) \models$ *FExists* φ
with *FExists.prems(3)* **obtain** p **where** $(w, \mathit{Inl}\ p\ \#\ I) \models \varphi$ **by** *auto*
moreover
with *FExists.prems* **have** $(w', \mathit{Inl}\ p\ \#\ I') \models \varphi$
apply (*intro iffD1*[*OF FExists.IH*[*of w Inl p # I w' Inl p # I'*]])
apply (*auto simp*: *stream-map2-alt split*: *sum.splits*) []
apply (*auto split*: *sum.splits split-if-asm*) []
apply (*blast dest*!: *wf-interp-for-formula-FExists*)
apply (*blast dest*!: *wf-interp-for-formula-FExists*[*of I'*, *unfolded length*])
apply *assumption*
done
ultimately show $(w', I') \models$ *FExists* φ **by** *auto*
next
assume $(w', I') \models$ *FExists* φ
with *FExists.prems(1,2,4)* **obtain** p **where** $(w', \mathit{Inl}\ p\ \#\ I') \models \varphi$ **by** *auto*
moreover
with *FExists.prems* **have** $(w, \mathit{Inl}\ p\ \#\ I) \models \varphi$
apply (*intro iffD2*[*OF FExists.IH*[*of w Inl p # I w' Inl p # I'*]])
apply (*auto simp*: *stream-map2-alt split*: *sum.splits*) []
apply (*auto split*: *sum.splits split-if-asm*) []
apply (*blast dest*!: *wf-interp-for-formula-FExists*)
apply (*blast dest*!: *wf-interp-for-formula-FExists*[*of I'*, *unfolded length*])
apply *assumption*
done
ultimately show $(w, I) \models$ *FExists* φ **by** *auto*
qed
next
case (*FEXISTS* φ)
hence *length*: *length* I' = *length* I **by** (*metis length-snth-enc*)
show *?case*
proof
assume $(w, I) \models$ *FEXISTS* φ
with *FEXISTS.prems(3)* **obtain** P **where** *finite* P $(w, \mathit{Inr}\ P\ \#\ I) \models \varphi$ **by** *auto*
moreover
with *FEXISTS.prems* **have** $(w', \mathit{Inr}\ P\ \#\ I') \models \varphi$
apply (*intro iffD1*[*OF FEXISTS.IH*[*of w Inr P # I w' Inr P # I'*]])
apply (*auto simp*: *stream-map2-alt split*: *sum.splits*) []
apply (*auto split*: *sum.splits split-if-asm*) []
apply (*blast dest*!: *wf-interp-for-formula-FEXISTS*)
apply (*blast dest*!: *wf-interp-for-formula-FEXISTS*[*of I'*, *unfolded length*])
apply *assumption*
done
ultimately show $(w', I') \models$ *FEXISTS* φ **by** *auto*
next
assume $(w', I') \models$ *FEXISTS* φ
with *FEXISTS.prems(1,2,4)* **obtain** P **where** *finite* P $(w', \mathit{Inr}\ P\ \#\ I') \models \varphi$ **by** *auto*
moreover

with *FEXISTS.prems* **have** $(w,\ Inr\ P\ \#\ I) \models \varphi$
apply (*intro iffD2*[*OF FEXISTS.IH*[*of w Inr P # I w′ Inr P # I′*]])
apply (*auto simp*: *stream-map2-alt split*: *sum.splits*) []
apply (*auto split*: *sum.splits split-if-asm*) []
apply (*blast dest*!: *wf-interp-for-formula-FEXISTS*)
apply (*blast dest*!: *wf-interp-for-formula-FEXISTS*[*of I′*, *unfolded length*])
apply *assumption*
done
ultimately show $(w,\ I) \models FEXISTS\ \varphi$ **by** *auto*
qed
qed *auto*

lemma $lang_{WS1S}$*-FOr*:
assumes *wf-formula n* $(FOr\ \varphi_1\ \varphi_2)$
shows $lang_{WS1S}\ n\ (FOr\ \varphi_1\ \varphi_2) \subseteq$
$(lang_{WS1S}\ n\ \varphi_1 \cup lang_{WS1S}\ n\ \varphi_2) \cap \bigcup\{enc\ (w,\ I) \mid w\ I.\ length\ I = n \wedge$ *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** $x \in lang_{WS1S}\ n\ (FOr\ \varphi_1\ \varphi_2)$
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** $lang_{WS1S}$*-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** $lang_{WS1S}$*-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** $lang_{WS1S}$*-def* **by** (*fastforce simp del*: *enc.simps*)
with $*$ **show** *?thesis* **by** *auto*
qed
qed

lemma $lang_{WS1S}$*-FAnd*:
assumes *wf-formula n* $(FAnd\ \varphi_1\ \varphi_2)$
shows $lang_{WS1S}\ n\ (FAnd\ \varphi_1\ \varphi_2) \subseteq$
$lang_{WS1S}\ n\ \varphi_1 \cap lang_{WS1S}\ n\ \varphi_2 \cap \bigcup\{enc\ (w,\ I) \mid w\ I.\ length\ I = n \wedge$ *wf-interp-for-formula* $(w,\ I)\ (FAnd\ \varphi_1\ \varphi_2)\}$
using *assms* **unfolding** $lang_{WS1S}$*-def* **by** (*fastforce simp del*: *enc.simps*)

10.3 From WS1S to Regular expressions

fun *rexp-of* :: *nat* $\Rightarrow$ *′a formula* $\Rightarrow$ (*′a* $\times$ *bool list*) *rexp* **where**
rexp-of n (*FQ a m*) =
Inter (*TIMES* [*rexp.Not Zero*, *arbitrary-except n* [(*m*, *True*)] [*a*], *rexp.Not Zero*])

$(ENC\ n\ (FQ\ a\ m))$
| *rexp-of* n (*FLess m1 m2*) = (*if m1* = *m2 then Zero else*
Inter (*TIMES* [*rexp.Not Zero*, *arbitrary-except* n [(*m1*, *True*)] Σ,
rexp.Not Zero, *arbitrary-except* n [(*m2*, *True*)] Σ,
rexp.Not Zero]) (*ENC* n (*FLess m1 m2*)))
| *rexp-of* n (*FIn* m M) =
Inter (*TIMES* [*rexp.Not Zero*, *arbitrary-except* n [(*min* m M, *True*), (*max* m M, *True*)] Σ, *rexp.Not Zero*])
(*ENC* n (*FIn* m M))
| *rexp-of* n (*FNot* φ) = *Inter* (*rexp.Not* (*rexp-of* n φ)) (*ENC* n (*FNot* φ))
| *rexp-of* n (*FOr* φ_1 φ_2) = *Inter* (*Plus* (*rexp-of* n φ_1) (*rexp-of* n φ_2)) (*ENC* n (*FOr* φ_1 φ_2))
| *rexp-of* n (*FAnd* φ_1 φ_2) = *INTERSECT* [*rexp-of* n φ_1, *rexp-of* n φ_2, *ENC* n (*FAnd* φ_1 φ_2)]
| *rexp-of* n (*FExists* φ) = *samequot-exec* (*any*, *replicate* n *False*) (*Pr* (*rexp-of* (n + *1*) φ))
| *rexp-of* n (*FEXISTS* φ) = *samequot-exec* (*any*, *replicate* n *False*) (*Pr* (*rexp-of* (n + *1*) φ))

fun *rexp-of-alt* :: *nat* $\Rightarrow$ *'a formula* $\Rightarrow$ (*'a* $\times$ *bool list*) *rexp* **where**
rexp-of-alt n (*FQ* a m) =
TIMES [*rexp.Not Zero*, *arbitrary-except* n [(m, *True*)] [a], *rexp.Not Zero*]
| *rexp-of-alt* n (*FLess m1 m2*) = (*if m1* = *m2 then Zero else*
TIMES [*rexp.Not Zero*, *arbitrary-except* n [(*m1*, *True*)] Σ,
rexp.Not Zero, *arbitrary-except* n [(*m2*, *True*)] Σ,
rexp.Not Zero])
| *rexp-of-alt* n (*FIn* m M) =
TIMES [*rexp.Not Zero*, *arbitrary-except* n [(*min* m M, *True*), (*max* m M, *True*)] Σ, *rexp.Not Zero*]
| *rexp-of-alt* n (*FNot* φ) = *rexp.Not* (*rexp-of-alt* n φ)
| *rexp-of-alt* n (*FOr* φ_1 φ_2) = *Plus* (*rexp-of-alt* n φ_1) (*rexp-of-alt* n φ_2)
| *rexp-of-alt* n (*FAnd* φ_1 φ_2) = *Inter* (*rexp-of-alt* n φ_1) (*rexp-of-alt* n φ_2)
| *rexp-of-alt* n (*FExists* φ) = *samequot-exec* (*any*, *replicate* n *False*) (*Pr* (*Inter* (*rexp-of-alt* (n + *1*) φ) (*ENC* (*Suc* n) φ)))
| *rexp-of-alt* n (*FEXISTS* φ) = *samequot-exec* (*any*, *replicate* n *False*) (*Pr* (*Inter* (*rexp-of-alt* (n + *1*) φ) (*ENC* (*Suc* n) φ)))

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

lemma *enc-eqI*:
assumes $x \in$ *enc* (w, I) $x \in$ *enc* (w', I') *wf-interp-for-formula* (w, I) φ *wf-interp-for-formula* (w', I') φ
length I = *length* I'
shows *enc* (w, I) = *enc* (w', I')
proof −
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) φ; *wf-interp-for-formula* (w, I) φ ;*wf-interp-for-formula* (w', I') $\varphi\rrbracket \Longrightarrow$
$(w, I) \models \varphi \longleftrightarrow (w', I') \models \varphi$
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) φ;
wf-interp-for-formula (w, I) φ ;*wf-interp-for-formula* (w', I') $\varphi\rrbracket \Longrightarrow$
$(w, I) \models \varphi \longleftrightarrow (w', I') \models \varphi$
by (*intro enc-eq-welldef*[*OF enc-eqI*])

lemma *wf-rexp-of*: *wf-formula* n $\varphi \Longrightarrow$ *wf* n (*rexp-of* n φ)
by (*induct* φ *arbitrary*: n)
(*auto simp*: *wf-rexp-ENC intro*: *wf-rexp-arbitrary-except intro*!: *wf-samequot-exec*,
auto simp: σ*-def set-n-lists image-iff*)

theorem *lang*$_{WS1S}$*-rexp-of*: *wf-formula* n $\varphi \Longrightarrow$ *lang*$_{WS1S}$ n φ = *lang* n (*rexp-of* n φ)
(**is** - $\Longrightarrow$ - = *?L* n φ)
proof (*induct* φ *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*$_{WS1S}$*-def* **by** *blast*
hence *x-alt*: x = *map* (*split* (*enc-atom* I)) (*zip* [*0* ..< *length* x] (*stake* (*length* x) (w @– *same 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 enc-def 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-cutSame-Inl*])
hence *enc-atom*: x ! p = *enc-atom* I p ((w @– *same 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-atom* I p *?p*] $\in$ *lang* n (*arbitrary-except* n [(m, *True*)] [a])
by (*intro enc-atom-lang-arbitrary-except-True*) *auto*
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-*σ *dest*!:

in-set-takeD)
moreover from *(2,3) **have** 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 (FQ a m) **using** *(1,2,3)
unfolding rexp-of.simps lang.simps(5,8) rexp-of-list.simps lang-ENC[OF FQ]
by (auto elim: ssubst simp del: o-apply append.simps lang.simps enc.simps)
next
fix x **let** ?x = x @− same (any, replicate n False)
assume x: x ∈ ?L n (FQ a m)
with FQ **obtain** w I **where**
I: x ∈ enc (w, I) length I = n wf-interp-for-formula (w, I) (FQ a m)
unfolding rexp-of.simps lang.simps lang-ENC[OF FQ] **by** fastforce
hence stream-enc: stream-enc (w, I) = ?x **using** stream-enc-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-interp-for-formula (dec-word ?x, stream-dec n {m} ?x) (FQ a m) **unfolding** I(1)
using enc-wf-interp[OF FQ(1)[folded I(2)]] **by** auto
moreover
from x **obtain** u1 u u2 **where** x = u1 @ u @ u2
u ∈ lang n (arbitrary-except n [(m, True)] [a])
unfolding rexp-of.simps lang.simps rexp-of-list.simps **using** concE **by** fast
with FQ(1) **obtain** v **where** v: x = u1 @ [v] @ u2 snd v ! m fst v = a
using arbitrary-except[of u n m True [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-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
} note * = this
from v **have** v = x ! length u1 **by** simp
with u **have** ?x !! length u1 = v **by** (auto simp: shift-snth)
with * m I v(3) **have** (dec-word ?x, stream-dec n {m} ?x) ⊨ FQ a m
using stream-enc-enc[OF - I(1), symmetric] less-length-cutSame[of w any length u1 a]
by (auto simp del: enc.simps stream-enc.simps simp: dec-word-stream-enc dest!:
stream-dec-enc-Inl stream-dec-not-Inr split: sum.splits)
(auto simp: stream-map2-alt cutSame-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*: *stream-map2-alt sdrop-snth shift-snth intro*: *stream-enc-unique*,
auto simp: *stream-map2-szip stream.set-natural'*)
moreover from *I* **have** *wf-word n x* **unfolding** *wf-word* **by** (*auto elim*: *enc-set-σ simp del*: *enc.simps*)
ultimately show $x \in lang_{WS1S}$ *n* (*FQ a m*) **unfolding** $lang_{WS1S}$*-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** $x \in lang_{WS1S}$ *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 $lang_{WS1S}$*-def* **by** *blast*
hence *x-alt*: *x* = *map* (*split* (*enc-atom I*)) (*zip* [*0* ..< *length x*] (*stake* (*length x*) (*w* @− *same 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 enc-def 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-cutSame-Inl*])
hence *enc-atom*: *x* ! *p* = *enc-atom I p* ((*w* @− *same any*) !! *p*) (**is** - = *enc-atom* - - *?p*)
x ! *q* = *enc-atom I q* ((*w* @− *same 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* (*arbitrary-except n* [(*m*, *True*)] Σ)
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* (*arbitrary-except n* [(*m'*, *True*)] Σ)
by (*intro enc-atom-lang-arbitrary-except-True*) (*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-*σ *dest*!: *in-set-takeD*)
moreover from $*(2,3)$ **have** *take* $(q - p - 1)$ (*drop* (*Suc p*) *x*) $\in$ *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*) $\in$ *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 $x \in$ *?L n* (*FLess m m'*) **using** $*(1,2,3)$
unfolding *rexp-of.simps lang.simps*(*5*,*8*) *rexp-of-list.simps Int-Diff lang-ENC*[*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* @− *same* (*any*, *replicate n False*)
assume *x*: $x \in$ *?L n* (*FLess m m'*)
with *FLess* **obtain** *w I* **where**
I: $x \in$ *enc* (*w*, *I*) *length I* = *n wf-interp-for-formula* (*w*, *I*) (*FLess m m'*)
unfolding *rexp-of.simps lang.simps lang-ENC*[*OF FLess*] *if-not-P*[*OF False*] **by** *fastforce*
hence *stream-enc*: *stream-enc* (*w*, *I*) = *x* @− *same* (*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 u u2 u' u3* **where** *x* = *u1* @ *u* @ *u2* @ *u'* @ *u3*
u $\in$ *lang n* (*arbitrary-except n* [(*m*, *True*)] Σ)
u' $\in$ *lang n* (*arbitrary-except n* [(*m'*, *True*)] Σ)
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* $\in$ *set* Σ *fst v'* $\in$ *set* Σ
using *arbitrary-except*[*of u n m True* Σ] *arbitrary-except*[*of u' n m' True* Σ] **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 I* **have** *p* < *length x snd* (*x* ! *p*) ! *m* **by** (*auto dest*: *enc-Inl simp del*: *enc.simps*)
moreover
from *m I* **have** *ex1*: $\exists!p.$ *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*: $\exists!p$. *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*) $\models$ *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*: *stream-map2-alt sdrop-snth shift-snth intro*: *stream-enc-unique*,
auto simp: *stream-map2-szip stream.set-natural'*)
moreover from I **have** *wf-word n x* **unfolding** *wf-word* **by** (*auto elim*: *enc-set-*σ *simp del*: *enc.simps*)
ultimately show $x \in lang_{WS1S}$ *n* (*FLess m m'*) **unfolding** $lang_{WS1S}$*-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*: $lang_{WS1S}$*-def del*: *o-apply*)
next
case (*FIn m M*)
show *?case*
proof (*intro equalityI subsetI*)
fix x **assume** $x \in lang_{WS1S}$ *n* (*FIn m M*)
then obtain *w I* **where**
$*$: $x \in$ *enc* (w, I) *wf-interp-for-formula* (w, I) (*FIn m M*) *length I* = *n* (w, I) $\models$ *FIn m M*
unfolding $lang_{WS1S}$*-def* **by** *blast*
hence *x-alt*: x = *map* (*split* (*enc-atom I*)) (*zip* [*0* ..< *length x*] (*stake* (*length x*) (w @− *same 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 enc-def split*: *sum.splits*)
with *FIn(1)* $*$(*1,2,3*) **have** *p-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-cutSame-Inl*]

trans-less-add1 [*OF bspec*[*OF less-length-cutSame-Inr*]])
hence *enc-atom*: $x ! p = enc\text{-}atom\ I\ p\ ((w\ @-\ same\ any)\ !!\ p)$ (**is** - = *enc-atom* - - *?p*)
$\forall p \in P.\ x ! p = enc\text{-}atom\ I\ p\ ((w\ @-\ same\ any)\ !!\ p)$ (**is** *Ball* - ($\lambda p.$ - = *enc-atom* - - (*?P p*)))
by (*subst x-alt*, *simp*)+
with ∗(*1*) *p-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
have [*enc-atom I p ?p*] ∈ *lang n* (*arbitrary-except n* [(*min m M*, *True*), (*max m M*, *True*)] Σ)
proof (*cases* $m < M$)
case *True* **with** ∗(*2*,*3*) *FIn*(*1*) *p* **show** *?thesis*
by (*intro enc-atom-lang-arbitrary-except-True2*) (*auto simp*: *min-absorb1 max-absorb2 shift-snth*)
next
case *False* **with** ∗(*2*,*3*) *FIn*(*1*) *p* **show** *?thesis*
by (*intro enc-atom-lang-arbitrary-except-True2*) (*auto simp*: *min-absorb2 max-absorb1 shift-snth*)
qed
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** *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* (*FIn m M*) **using** ∗(*1*,*2*,*3*)
unfolding *rexp-of.simps lang.simps*(*5*,*8*) *rexp-of-list.simps Int-Diff lang-ENC*[*OF FIn*]
by (*auto elim*: *ssubst simp del*: *o-apply append.simps lang.simps enc.simps*)
next
fix *x* **let** *?x* = *x* @− *same* (*any*, *replicate n False*)
assume *x*: *x* ∈ *?L n* (*FIn m M*)
with *FIn* **obtain** *w I* **where**
I: *x* ∈ *enc* (*w*, *I*) *length I* = *n wf-interp-for-formula* (*w*, *I*) (*FIn m M*)
unfolding *rexp-of.simps lang.simps lang-ENC*[*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-word ?x*, *stream-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 ∈ *lang n* (*arbitrary-except n* [(*min m M*, *True*), (*max m M*, *True*)] Σ)
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* **and** *snd v* ! *min m M snd v* ! *max m M fst v* ∈ *set* Σ

using *arbitrary-except2*[*of u n min m M True max m M True* Σ] **by** *fastforce*
hence v': *snd v* ! *m snd v* ! *M*
by (*induct m* < *M*) (*auto simp*: *min-absorb1 min-absorb2 max-absorb1 max-absorb2*)
from *v* **have** *u*: *length u1* < *length x* **by** *auto*
{ from *v* 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*
} note ∗ = *this*
from *v* v' **have** *v* = *x* ! *length u1* **by** *simp*
with v'(*2*) *m*(*3*,*4*) *u I*(*1*,*3*) **have** *length u1* ∈ *P* **by** (*auto dest*!: *enc-Inr simp del*: *enc.simps*)
with ∗ *m I* **have** (*dec-word ?x*, *stream-dec n* {*m*} *?x*) ⊨ *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*: *stream-map2-alt sdrop-snth shift-snth intro*: *stream-enc-unique*, *auto simp*: *stream-map2-szip stream.set-natural*′)
moreover from *I* **have** *wf-word n x* **unfolding** *wf-word* **by** (*auto elim*: *enc-set-σ simp del*: *enc.simps*)
ultimately show *x* ∈ $lang_{WS1S}$ *n* (*FIn m M*) **unfolding** $lang_{WS1S}$*-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* φ_1 φ_2)
from *FOr*(*3*) **have** *IH1*: $lang_{WS1S}$ *n* φ_1 = *lang n* (*rexp-of n* φ_1)
by (*intro FOr*(*1*)) *auto*
from *FOr*(*3*) **have** *IH2*: $lang_{WS1S}$ *n* φ_2 = *lang n* (*rexp-of n* φ_2)
by (*intro FOr*(*2*)) *auto*
show *?case*
proof (*intro equalityI subsetI*)
fix *x* **assume** *x* ∈ $lang_{WS1S}$ *n* (*FOr* φ_1 φ_2) **thus** *x* ∈ *lang n* (*rexp-of n* (*FOr* φ_1 φ_2))
using $lang_{WS1S}$*-FOr*[*OF FOr*(*3*)] **unfolding** *lang-ENC*[*OF FOr*(*3*)] *rexp-of.simps lang.simps IH1 IH2* **by** *blast*
next

fix x **assume** $x \in$ *lang n* (*rexp-of n* (*FOr* φ_1 φ_2))
then obtain *w I* **where** *or*: $x \in lang_{WS1S}$ *n* $\varphi_1 \lor x \in lang_{WS1S}$ *n* φ_2 **and** *I*: $x \in$ *enc* (*w*, *I*) *length I* = *n*
wf-interp-for-formula (*w*, *I*) (*FOr* φ_1 φ_2)
unfolding *lang-ENC*[*OF FOr*(*3*)] *rexp-of.simps lang.simps IH1 IH2 Int-Diff* **by** *auto*
have (*w*, *I*) $\models \varphi_1 \lor$ (*w*, *I*) $\models \varphi_2$
proof (*intro mp*[*OF disj-mono*[*OF impI impI*] *or*])
assume $x \in lang_{WS1S}$ *n* φ_1
with *I FOr*(*3*) **show** (*w*, *I*) $\models \varphi_1$
unfolding $lang_{WS1S}$*-def I*(*1*) *wf-interp-for-formula-FOr*
by (*auto dest*!: *enc-welldef*[*of x w I* - - φ_1] *simp del*: *enc.simps*)
next
assume $x \in lang_{WS1S}$ *n* φ_2
with *I FOr*(*3*) **show** (*w*, *I*) $\models \varphi_2$
unfolding $lang_{WS1S}$*-def I*(*1*) *wf-interp-for-formula-FOr*
by (*auto dest*!: *enc-welldef*[*of x w I* - - φ_2] *simp del*: *enc.simps*)
qed
with *I* **show** $x \in lang_{WS1S}$ *n* (*FOr* φ_1 φ_2) **unfolding** $lang_{WS1S}$*-def* **by** *auto*
qed
next
case (*FAnd* φ_1 φ_2)
from *FAnd*(*3*) **have** *IH1*: $lang_{WS1S}$ *n* φ_1 = *lang n* (*rexp-of n* φ_1)
by (*intro FAnd*(*1*)) *auto*
from *FAnd*(*3*) **have** *IH2*: $lang_{WS1S}$ *n* φ_2 = *lang n* (*rexp-of n* φ_2)
by (*intro FAnd*(*2*)) *auto*
show *?case*
proof (*intro equalityI subsetI*)
fix x **assume** $x \in lang_{WS1S}$ *n* (*FAnd* φ_1 φ_2) **thus** $x \in$ *lang n* (*rexp-of n* (*FAnd* φ_1 φ_2))
using $lang_{WS1S}$*-FAnd*[*OF FAnd*(*3*)]
unfolding *lang-ENC*[*OF FAnd*(*3*)] *rexp-of.simps rexp-of-list.simps lang.simps IH1 IH2 Int-assoc*
by *blast*
next
fix x **assume** $x \in$ *lang n* (*rexp-of n* (*FAnd* φ_1 φ_2))
then obtain *w I* **where** *and*: $x \in lang_{WS1S}$ *n* $\varphi_1 \land x \in lang_{WS1S}$ *n* φ_2 **and** *I*: $x \in$ *enc* (*w*, *I*) *length I* = *n*
wf-interp-for-formula (*w*, *I*) (*FAnd* φ_1 φ_2)
unfolding *lang-ENC*[*OF FAnd*(*3*)] *rexp-of.simps rexp-of-list.simps lang.simps IH1 IH2 Int-Diff* **by** *auto*
have (*w*, *I*) $\models \varphi_1 \land$ (*w*, *I*) $\models \varphi_2$
proof (*intro mp*[*OF conj-mono*[*OF impI impI*] *and*])
assume $x \in lang_{WS1S}$ *n* φ_1
with *I FAnd*(*3*) **show** (*w*, *I*) $\models \varphi_1$
unfolding $lang_{WS1S}$*-def I*(*1*) *wf-interp-for-formula-FAnd*
by (*auto dest*!: *enc-welldef*[*of x w I* - - φ_1] *simp del*: *enc.simps*)
next
assume $x \in lang_{WS1S}$ *n* φ_2

with *I FAnd*(*3*) **show** $(w, I) \models \varphi_2$
unfolding *lang*$_{WS1S}$*-def I*(*1*) *wf-interp-for-formula-FAnd*
by (*auto dest*!: *enc-welldef*[*of x w I - -* φ_2] *simp del*: *enc.simps*)
qed
with *I* **show** $x \in$ *lang*$_{WS1S}$ *n* (*FAnd* φ_1 φ_2) **unfolding** *lang*$_{WS1S}$*-def* **by** *auto*
qed
next
case (*FNot* φ)
hence *IH*: *?L n* φ = *lang*$_{WS1S}$ *n* φ **by** *simp*
show *?case*
proof (*intro equalityI subsetI*)
fix *x* **assume** $x \in$ *lang*$_{WS1S}$ *n* (*FNot* φ)
then obtain *w I* **where**
$*$: $x \in$ *enc* (*w*, *I*) *wf-interp-for-formula* (*w*, *I*) φ *length I* = *n* **and** *unsat*: $\neg$ $(w, I) \models \varphi$
unfolding *lang*$_{WS1S}$*-def* **by** *auto*
{ **assume** $x \in$ *?L n* φ
hence $(w, I) \models \varphi$ **using** *enc-welldef*[*of x w I - -* φ, *OF* $*$(*1*) *- - -* $*$(*2*)] *FNot*(*2*)
unfolding $*$(*3*) *lang*$_{WS1S}$*-def IH* **by** *auto*
}
with *unsat* **have** $x \notin$ *?L n* φ **by** *blast*
with $*$ **show** $x \in$ *?L n* (*FNot* φ) **unfolding** *rexp-of.simps lang.simps* **using** *lang-ENC*[*OF FNot*(*2*)]
by (*auto simp del*: *enc.simps simp*: *comp-def intro*: *enc-set-*σ)
next
fix *x* **assume** $x \in$ *?L n* (*FNot* φ)
with *IH* **have** $x \in$ *lang n* (*ENC n* (*FNot* φ)) **and** *x*: $x \notin$ *lang*$_{WS1S}$ *n* φ **by** (*auto simp del*: *o-apply*)
then obtain *w I* **where** $*$: $x \in$ *enc* (*w*, *I*) *wf-interp-for-formula* (*w*, *I*) (*FNot* φ) *length I* = *n*
unfolding *lang-ENC*[*OF FNot*(*2*)] **by** *blast*
{ **assume** $\neg$ $(w, I) \models$ *FNot* φ
with $*$ **have** $x \in$ *lang*$_{WS1S}$ *n* φ **unfolding** *lang*$_{WS1S}$*-def* **by** *auto*
}
with *x* $*$ **show** $x \in$ *lang*$_{WS1S}$ *n* (*FNot* φ) **unfolding** *lang*$_{WS1S}$*-def* **by** *blast*
qed
next
case (*FExists* φ)
have σ: (*any*, *replicate n False*) $\in$ (*set o* σ Σ) *n* **by** (*auto simp*: σ*-def set-n-lists image-iff*)
from *FExists*(*2*) **have** *wf*: *wf n* (*Pr* (*rexp-of* (*Suc n*) φ)) **by** (*fastforce intro*: *wf-rexp-of*)
note *lang-quot* = *lang-samequot-exec*[*OF wf* σ]
show *?case*
proof (*intro equalityI subsetI*)
fix *x* **assume** $x \in$ *lang*$_{WS1S}$ *n* (*FExists* φ)
then obtain *w I p* **where**

$*$: $x \in$ *enc* (w, I) *wf-interp-for-formula* (w, I) (*FExists* φ) *length* $I = n$ $(w,$ *Inl* p # $I) \models \varphi$
unfolding $lang_{WS1S}$-*def* **by** *auto*
with *FExists*(*2*) **have** *enc* $(w,$ *Inl* p # $I) \subseteq$ *?L* (*Suc* n) φ
by (*subst FExists*(*1*)[*of Suc* n, *symmetric*])
(*fastforce simp del*: *enc.simps simp*: $lang_{WS1S}$-*def nth-Cons′ intro*!: *exI*[*of* - *enc* $(w,$ *Inl* p # $I)$])+
thus $x \in$ *?L* n (*FExists* φ) **using** $*$(*1*,*2*,*3*)
by (*auto simp*: *lang-quot simp del*: *o-apply enc.simps elim*: *set-mp*[*OF SAMEQUOT-mono*[*OF image-mono*]])
next
fix x **assume** $x \in$ *?L* n (*FExists* φ)
then obtain x' m **where** $x' \in$ *?L* (*Suc* n) φ **and**
x: $x =$ *fin-cutSame* (*any*, *replicate* n *False*) (*map* π 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) φ
by (*intro subsetD*[*OF equalityD2*[*OF FExists*(*1*)], *of Suc* n x'])
(*auto split*: *split-if-asm sum.splits*)
then obtain w I' **where**
$*$: $x' \in$ *enc* (w, I') *wf-interp-for-formula* (w, I') φ *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 split-if-asm*)
ultimately have $x \in$ *enc* (w, I) *wf-interp-for-formula* (w, I) (*FExists* φ) *length* $I = n$
$(w, I) \models$ *FExists* φ **using** *FExists*(*2*) *fin-cutSame-tl*[*OF ex-Loop-stream-enc*, *of Inl* p # I w]
unfolding x **by** (*auto simp add*: π-*def nth-Cons′ split*: *split-if-asm*)
thus $x \in lang_{WS1S}$ n (*FExists* φ) **unfolding** $lang_{WS1S}$-*def* **by** (*auto intro*!: *exI*[*of* - I])
qed
next
case (*FEXISTS* φ)
have σ: (*any*, *replicate* n *False*) $\in$ (*set o* σ Σ) n **by** (*auto simp*: σ-*def set-n-lists image-iff*)
from *FEXISTS*(*2*) **have** *wf*: *wf* n (*Pr* (*rexp-of* (*Suc* n) φ)) **by** (*fastforce intro*: *wf-rexp-of*)
note *lang-quot* = *lang-samequot-exec*[*OF wf* σ]
show *?case*
proof (*intro equalityI subsetI*)
fix x **assume** $x \in lang_{WS1S}$ n (*FEXISTS* φ)
then obtain w I P **where**
$*$: $x \in$ *enc* (w, I) *wf-interp-for-formula* (w, I) (*FEXISTS* φ) *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) φ

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* φ) **using** $*$(*1*,*2*,*3*,*4*)
by (*auto simp*: *lang-quot simp del*: *o-apply enc.simps elim*: *set-mp*[*OF SAMEQUOT-mono*[*OF image-mono*]])
next
fix x **assume** $x \in$ *?L n* (*FEXISTS* φ)
then obtain x' m **where** $x' \in$ *?L* (*Suc n*) φ **and**
x: x = *fin-cutSame* (*any*, *replicate n False*) (*map* π 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*) φ
by (*intro subsetD*[*OF equalityD2*[*OF FEXISTS*(*1*)], *of Suc n x′*])
(*auto split*: *split-if-asm sum.splits*)
then obtain w I' **where**
$*$: $x' \in$ *enc* (w, I') *wf-interp-for-formula* (w, I') φ *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 = *Inr P finite P*
by (*auto simp*: *nth-Cons′ split*: *sum.splits split-if-asm*)
ultimately have $x \in$ *enc* (w, I) *wf-interp-for-formula* (w, I) (*FEXISTS* φ) *length* $I = n$
(w, I) $\models$ *FEXISTS* φ **using** *FEXISTS*(*2*) *fin-cutSame-tl*[*OF ex-Loop-stream-enc*, *of Inr P* # *I*]
unfolding x **by** (*auto simp*: *nth-Cons′* π-*def split*: *split-if-asm*)
thus $x \in$ *lang*$_{WS1S}$ *n* (*FEXISTS* φ) **unfolding** *lang*$_{WS1S}$-*def* **by** (*auto intro*!: *exI*[*of* - *I*])
qed
qed

lemma *wf-rexp-of-alt*: *wf-formula n* φ $\Longrightarrow$ *wf n* (*rexp-of-alt n* φ)
by (*induct* φ *arbitrary*: *n*)
(*auto simp*: *wf-rexp-ENC intro*: *wf-rexp-arbitrary-except intro*!: *wf-samequot-exec*,
auto simp: σ-*def set-n-lists image-iff*)

lemma *wf-rexp-of′*: *wf-formula n* φ $\Longrightarrow$ *wf n* (*rexp-of′ n* φ)
unfolding *rexp-of′-def* **by** (*auto intro*: *wf-rexp-of-alt wf-rexp-ENC*)

lemma *ENC-FNot*: *ENC n* (*FNot* φ) = *ENC n* φ
unfolding *ENC-def* **by** *auto*

lemma *ENC-FAnd*:
wf-formula n (*FAnd* φ ψ) $\Longrightarrow$ *lang n* (*ENC n* (*FAnd* φ ψ)) $\subseteq$ *lang n* (*ENC n* φ) $\cap$ *lang n* (*ENC n* ψ)
proof
fix x **assume** *wf*: *wf-formula n* (*FAnd* φ ψ) **and** x: $x \in$ *lang n* (*ENC n* (*FAnd* φ ψ))

hence *wf1*: *wf-formula n* φ **and** *wf2*: *wf-formula n* ψ **by** *auto*
from *x* **obtain** *w I* **where** *I*: $x \in$ *enc* (*w*, *I*) *wf-interp-for-formula* (*w*, *I*) (*FAnd* φ ψ) *length* $I = n$
using *lang-ENC*[*OF wf*] **by** *auto*
hence *wf-interp-for-formula* (*w*, *I*) φ *wf-interp-for-formula* (*w*, *I*) ψ
using *wf-interp-for-formula-FAnd* **by** *auto*
thus $x \in$ *lang n* (*ENC n* φ) $\cap$ *lang n* (*ENC n* ψ)
unfolding *lang-ENC*[*OF wf1*] *lang-ENC*[*OF wf2*] **using** *I* **by** *blast*
qed

lemma *ENC-FOr*:
wf-formula n (*FOr* φ ψ) $\Longrightarrow$ *lang n* (*ENC n* (*FOr* φ ψ)) $\subseteq$ *lang n* (*ENC n* φ) $\cap$ *lang n* (*ENC n* ψ)
proof
fix *x* **assume** *wf*: *wf-formula n* (*FOr* φ ψ) **and** *x*: $x \in$ *lang n* (*ENC n* (*FOr* φ ψ))
hence *wf1*: *wf-formula n* φ **and** *wf2*: *wf-formula n* ψ **by** *auto*
from *x* **obtain** *w I* **where** *I*: $x \in$ *enc* (*w*, *I*) *wf-interp-for-formula* (*w*, *I*) (*FOr* φ ψ) *length* $I = n$
using *lang-ENC*[*OF wf*] **by** *auto*
hence *wf-interp-for-formula* (*w*, *I*) φ *wf-interp-for-formula* (*w*, *I*) ψ
using *wf-interp-for-formula-FOr* **by** *auto*
thus $x \in$ *lang n* (*ENC n* φ) $\cap$ *lang n* (*ENC n* ψ)
unfolding *lang-ENC*[*OF wf1*] *lang-ENC*[*OF wf2*] **using** *I* **by** *blast*
qed

lemma *ENC-FExists*:
wf-formula n (*FExists* φ) $\Longrightarrow$ *lang n* (*ENC n* (*FExists* φ)) =
SAMEQUOT (*any*, *replicate n False*) (*map* π ‘ *lang* (*Suc n*) (*ENC* (*Suc n*) φ))
(**is** - $\Longrightarrow$ *?L* = *?R*)
proof (*intro equalityI subsetI*)
fix *x* **assume** *wf*: *wf-formula n* (*FExists* φ) **and** *x*: $x \in$ *?L*
hence *wf1*: *wf-formula* (*Suc n*) φ **by** *auto*
from *x* **obtain** *w I* **where** *I*: $x \in$ *enc* (*w*, *I*) *wf-interp-for-formula* (*w*, *I*) (*FExists* φ) *length* $I = n$
using *lang-ENC*[*OF wf*] **by** *auto*
from *I*(*2*) **obtain** *p* **where** *wf-interp-for-formula* (*w*, *Inl p* # *I*) φ
using *wf-interp-for-formula-FExists*[*OF wf*[*folded I*(*3*)]] **by** *blast*
with *I*(*3*) **show** $x \in$ *?R*
unfolding *lang-ENC*[*OF wf1*] **using** *I*(*1*) *tl-enc*[*of Inl p I*, *symmetric*]
by (*simp del*: *enc.simps*)
(*fastforce simp del*: *enc.simps elim*!: *set-rev-mp*[*OF* - *SAMEQUOT-mono*[*OF image-mono*]]
intro: *exI*[*of* - *enc* (*w*, *Inl p* # *I*)]])
next
fix *x* **assume** *wf*: *wf-formula n* (*FExists* φ) **and** *x*: $x \in$ *?R*
hence *wf1*: *wf-formula* (*Suc n*) φ **and** $0 \in$ *FOV* φ **by** *auto*
from *x* **obtain** *w I* **where** *I*: $x \in$ *SAMEQUOT* (*any*, *replicate n False*) (*map* π ‘ *enc* (*w*, *I*))

wf-interp-for-formula (*w*, *I*) *φ* *length* *I* = *Suc* *n*
using *lang-ENC*[*OF* *wf1*] **unfolding** *SAMEQUOT-def* **by** *fast*
with ‹*0* ∈ *FOV* *φ*› **obtain** *p* *I′* **where** *I′*: *I* = *Inl* *p* # *I′* **by** (*cases* *I*) (*fastforce* *split*: *sum.splits*)+
with *I* **have** *wtlI*: *x* ∈ *enc* (*w*, *I′*) *length* *I′* = *n* **using** *tl-enc*[*of* *Inl* *p* *I′* *w*] **by** *auto*
have *wf-interp-for-formula* (*w*, *I′*) (*FExists* *φ*)
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* ∈ *?L* **unfolding** *lang-ENC*[*OF* *wf*] **by** *blast*
qed

lemma *ENC-FEXISTS*:
wf-formula *n* (*FEXISTS* *φ*) ⟹ *lang* *n* (*ENC* *n* (*FEXISTS* *φ*)) =
SAMEQUOT (*any*, *replicate* *n* *False*) (*map* *π* ‘ *lang* (*Suc* *n*) (*ENC* (*Suc* *n*) *φ*))
(**is** - ⟹ *?L* = *?R*)
proof (*intro* *equalityI* *subsetI*)
fix *x* **assume** *wf*: *wf-formula* *n* (*FEXISTS* *φ*) **and** *x*: *x* ∈ *?L*
hence *wf1*: *wf-formula* (*Suc* *n*) *φ* **by** *auto*
from *x* **obtain** *w* *I* **where** *I*: *x* ∈ *enc* (*w*, *I*) *wf-interp-for-formula* (*w*, *I*) (*FEXISTS* *φ*) *length* *I* = *n*
using *lang-ENC*[*OF* *wf*] **by** *auto*
from *I*(*2*) **obtain** *P* **where** *wf-interp-for-formula* (*w*, *Inr* *P* # *I*) *φ*
using *wf-interp-for-formula-FEXISTS*[*OF* *wf*[*folded* *I*(*3*)]] **by** *blast*
with *I*(*3*) **show** *x* ∈ *?R*
unfolding *lang-ENC*[*OF* *wf1*] **using** *I*(*1*) *tl-enc*[*of* *Inr* *P* *I*, *symmetric*]
by (*simp* *del*: *enc.simps*)
(*fastforce* *simp* *del*: *enc.simps* *elim*!: *set-rev-mp*[*OF* - *SAMEQUOT-mono*[*OF* *image-mono*]]
intro: *exI*[*of* - *enc* (*w*, *Inr* *P* # *I*)])
next
fix *x* **assume** *wf*: *wf-formula* *n* (*FEXISTS* *φ*) **and** *x*: *x* ∈ *?R*
hence *wf1*: *wf-formula* (*Suc* *n*) *φ* **and** *0* ∈ *SOV* *φ* **by** *auto*
from *x* **obtain** *w* *I* **where** *I*: *x* ∈ *SAMEQUOT* (*any*, *replicate* *n* *False*) (*map* *π* ‘ *enc* (*w*, *I*))
wf-interp-for-formula (*w*, *I*) *φ* *length* *I* = *Suc* *n*
using *lang-ENC*[*OF* *wf1*] **unfolding** *SAMEQUOT-def* **by** *fast*
with ‹*0* ∈ *SOV* *φ*› **obtain** *P* *I′* **where** *I′*: *I* = *Inr* *P* # *I′* **by** (*cases* *I*) (*fastforce* *split*: *sum.splits*)+
with *I* **have** *wtlI*: *x* ∈ *enc* (*w*, *I′*) *length* *I′* = *n* **using** *tl-enc*[*of* *Inr* *P* *I′* *w*] **by** *auto*
have *wf-interp-for-formula* (*w*, *I′*) (*FEXISTS* *φ*)
using *wf-interp-for-formula-FEXISTS*[*OF* *wf*[*folded* *wtlI*(*2*)]]
wf-interp-for-formula-any-Inr[*OF* *I*(*2*)[*unfolded* *I′*]] **..**
with *wtlI* **show** *x* ∈ *?L* **unfolding** *lang-ENC*[*OF* *wf*] **by** *blast*
qed

lemma *lang*$_{WS1S}$*-rexp-of-rexp-of′*:
wf-formula *n* *φ* ⟹ *lang* *n* (*rexp-of* *n* *φ*) = *lang* *n* (*rexp-of′* *n* *φ*)

unfolding *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.simps rexp-of-alt.simps lang.simps ENC-FNot* **by** *blast*
next
case (*FAnd* φ_1 φ_2)
hence *wf1*: *wf-formula n* φ_1 **and** *wf2*: *wf-formula n* φ_2 **by** *force+*
from *FAnd.IH*(*1*)[*OF wf1*] *FAnd.IH*(*2*)[*OF wf2*] **show** *?case* **using** *ENC-FAnd*[*OF FAnd.prems*]
unfolding *rexp-of.simps rexp-of-alt.simps lang.simps rexp-of-list.simps* **by** *blast*
next
case (*FOr* φ_1 φ_2)
hence *wf1*: *wf-formula n* φ_1 **and** *wf2*: *wf-formula n* φ_2 **by** *force+*
from *FOr.IH*(*1*)[*OF wf1*] *FOr.IH*(*2*)[*OF wf2*] **show** *?case* **using** *ENC-FOr*[*OF FOr.prems*]
unfolding *rexp-of.simps rexp-of-alt.simps lang.simps* **by** *blast*
next
case (*FExists* φ)
from *FExists*(*2*) **have** *IH*: *lang* (*n* + *1*) (*rexp-of* (*n* + *1*) φ) =
lang (*n* + *1*) (*Inter* (*rexp-of-alt* (*n* + *1*) φ) (*ENC* (*n* + *1*) φ)) **by** (*intro FExists.IH*) *auto*
have σ: (*any*, *replicate n False*) $\in$ (*set o* σ Σ) *n* **by** (*auto simp*: σ*-def set-n-lists image-iff*)
from *FExists*(*2*) **have** *wf*: *wf n* (*Pr* (*rexp.Inter* (*rexp-of-alt* (*n* + *1*) φ) (*ENC* (*n* + *1*) φ)))
wf n (*Pr* (*rexp-of* (*n* + *1*) φ)) **by** (*fastforce intro*!: *wf-rexp-of wf-rexp-of-alt wf-rexp-ENC*)+
note *lang-quot* = *lang-samequot-exec*[*OF wf*(*1*) σ] *lang-samequot-exec*[*OF wf*(*2*) σ]
show *?case* **unfolding** *rexp-of.simps rexp-of-alt.simps lang.simps IH lang-quot Suc-eq-plus1*
ENC-FExists[*OF FExists.prems*, *unfolded Suc-eq-plus1*] **by** (*auto simp add*: *SAMEQUOT-def*)
next
case (*FEXISTS* φ)
from *FEXISTS*(*2*) **have** *IH*: *lang* (*n* + *1*) (*rexp-of* (*n* + *1*) φ) =
lang (*n* + *1*) (*Inter* (*rexp-of-alt* (*n* + *1*) φ) (*ENC* (*n* + *1*) φ)) **by** (*intro FEXISTS.IH*) *auto*
have σ: (*any*, *replicate n False*) $\in$ (*set o* σ Σ) *n* **by** (*auto simp*: σ*-def set-n-lists image-iff*)
from *FEXISTS*(*2*) **have** *wf*: *wf n* (*Pr* (*rexp.Inter* (*rexp-of-alt* (*n* + *1*) φ) (*ENC* (*n* + *1*) φ)))
wf n (*Pr* (*rexp-of* (*n* + *1*) φ)) **by** (*fastforce intro*: *wf-rexp-of wf-rexp-of-alt wf-rexp-ENC*)+
note *lang-quot* = *lang-samequot-exec*[*OF wf*(*1*) σ] *lang-samequot-exec*[*OF wf*(*2*) σ]
show *?case* **unfolding** *rexp-of.simps rexp-of-alt.simps lang.simps IH lang-quot Suc-eq-plus1*

ENC-FEXISTS[*OF FEXISTS.prems*, *unfolded Suc-eq-plus1*] **by** (*auto simp add*: *SAMEQUOT-def*)
qed *auto*

theorem *lang*$_{WS1S}$*-rexp-of* ′: *wf-formula n* $\varphi \Longrightarrow$ *lang*$_{WS1S}$ *n* φ = *lang n* (*rexp-of* ′ *n* φ)
unfolding *lang*$_{WS1S}$*-rexp-of-rexp-of* ′[*symmetric*] **by** (*rule lang*$_{WS1S}$*-rexp-of*)

end

end

11 Normalization of WS1S Formulas

fun *nNot* **where**
nNot (*FNot* φ) = φ
| *nNot* (*FAnd* φ*1* φ*2*) = *FOr* (*nNot* φ*1*) (*nNot* φ*2*)
| *nNot* (*FOr* φ*1* φ*2*) = *FAnd* (*nNot* φ*1*) (*nNot* φ*2*)
| *nNot* φ = *FNot* φ

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* φ ψ) = *FOr* (*norm* φ) (*norm* ψ)
| *norm* (*FAnd* φ ψ) = *FAnd* (*norm* φ) (*norm* ψ)
| *norm* (*FNot* φ) = *nNot* (*norm* φ)
| *norm* (*FExists* φ) = *FExists* (*norm* φ)
| *norm* (*FEXISTS* φ) = *FEXISTS* (*norm* φ)

context *formula*
begin

lemma *satisfies-nNot*[*simp*]: (*w*, *I*) $\models$ *nNot* $\varphi \longleftrightarrow$ (*w*, *I*) $\models$ *FNot* φ
by (*induct* φ *rule*: *nNot.induct*) *auto*

lemma *FOV-nNot*[*simp*]: *FOV* (*nNot* φ) = *FOV* (*FNot* φ)
by (*induct* φ *rule*: *nNot.induct*) *auto*

lemma *SOV-nNot*[*simp*]: *SOV* (*nNot* φ) = *SOV* (*FNot* φ)
by (*induct* φ *rule*: *nNot.induct*) *auto*

lemma *pre-wf-formula-nNot*[*simp*]: *pre-wf-formula n* (*nNot* φ) = *pre-wf-formula n* (*FNot* φ)
by (*induct* φ *rule*: *nNot.induct*) *auto*

lemma *FOV-norm*[*simp*]: *FOV* (*norm* φ) = *FOV* φ

by (*induct* φ) *auto*

lemma *SOV-norm*[*simp*]: *SOV* (*norm* φ) = *SOV* φ
by (*induct* φ) *auto*

lemma *pre-wf-formula-norm*[*simp*]: *pre-wf-formula* n (*norm* φ) = *pre-wf-formula* n φ
by (*induct* φ *arbitrary*: n) *auto*

lemma *satisfies-norm*[*simp*]: $wI \models norm\ \varphi \longleftrightarrow wI \models \varphi$
by (*induct* φ *arbitrary*: *wI*) *auto*

lemma $lang_{WS1S}$*-norm*[*simp*]: $lang_{WS1S}$ n (*norm* φ) = $lang_{WS1S}$ n φ
unfolding $lang_{WS1S}$*-def* **by** *auto*

end

end

12 Deciding Equivalence of WS1S Formulas

type-synonym $'a\ T = 'a \times bool\ list$
abbreviation $\mathfrak{L} \equiv \lambda\Sigma.$ *project.lang* (*set o* (σ Σ)) π

definition *wf-rexp* **where** [*code del*]:
wf-rexp Σ = *alphabet.wf* (*set o* σ Σ)

interpretation *project set o* σ Σ π
where *alphabet.wf* (*set o* σ Σ) = *wf-rexp* Σ
by (*unfold-locales*) (*auto simp*: σ*-def* π*-def* *wf-rexp-def*)

definition *norm-lderiv* **where** [*code del*]: *norm-lderiv* $\equiv \lambda\Sigma.$ *embed.lderiv* (ε Σ)
definition *norm-rderiv* **where** [*code del*]: *norm-rderiv* $\equiv \lambda\Sigma.$ *embed.rderiv* (ε Σ)
definition *norm-rderiv-and-add* **where** [*code del*]: *norm-rderiv-and-add* $\equiv \lambda\Sigma.$ *embed.rderiv-and-add* (ε Σ)
definition *quot* **where** [*code del*]: *quot* $\equiv \lambda\Sigma.$ *embed.samequot-exec* (ε Σ)

interpretation *embed set* $\circ$ (σ (Σ :: $'a$ :: *linorder list*)) π ε Σ
where *embed.lderiv* (ε Σ) = *norm-lderiv* Σ
and *embed.rderiv* (ε Σ) = *norm-rderiv* Σ
and *embed.rderiv-and-add* (ε Σ) = *norm-rderiv-and-add* Σ
and *embed.samequot-exec* (ε Σ) = *quot* Σ
by (*unfold-locales*) (*auto simp*: *norm-lderiv-def norm-rderiv-def norm-rderiv-and-add-def quot-def* σ*-def* π*-def* ε*-def*)

definition *norm-step'* **where** [*code del*]:
norm-step' $\equiv \lambda\Sigma.$ *equivalence-checker.step'* (σ Σ) (ε Σ) (*Smart-Constructors-Normalization.norm*

:: $'a$::*linorder T rexp* ⇒ $'a$ *T rexp*)
definition *norm-closure'* **where** [*code del*]:
norm-closure' ≡ λΣ. *equivalence-checker.closure'* (σ Σ) (ε Σ) (*Smart-Constructors-Normalization.norm* :: $'a$::*linorder T rexp* ⇒ $'a$ *T rexp*)
definition *norm-check-eqv'* **where** [*code del*]:
norm-check-eqv' ≡ λΣ. *equivalence-checker.check-eqv'* (σ Σ) (ε Σ) (*Smart-Constructors-Normalization.norm* :: $'a$::*linorder T rexp* ⇒ $'a$ *T rexp*)
definition *norm-step* **where** [*code del*]:
norm-step ≡ λΣ. *equivalence-checker.step* (σ Σ) (ε Σ) (*Smart-Constructors-Normalization.norm* :: $'a$::*linorder T rexp* ⇒ $'a$ *T rexp*)
definition *norm-closure* **where** [*code del*]:
norm-closure ≡ λΣ. *equivalence-checker.closure* (σ Σ) (ε Σ) (*Smart-Constructors-Normalization.norm* :: $'a$::*linorder T rexp* ⇒ $'a$ *T rexp*)
definition *norm-check-eqv* **where** [*code del*]:
norm-check-eqv ≡ λΣ. *equivalence-checker.check-eqv* (σ Σ) (ε Σ) (*Smart-Constructors-Normalization.norm* :: $'a$::*linorder T rexp* ⇒ $'a$ *T rexp*)
definition *norm-check-eqv-counterexample* **where** [*code del*]:
norm-check-eqv-counterexample ≡ λΣ. *equivalence-checker.check-eqv-counterexample* (σ Σ) (ε Σ) (*Smart-Constructors-Normalization.norm* :: $'a$::*linorder T rexp* ⇒ $'a$ *T rexp*)

lemmas *norm-defs* = *wf-rexp-def*
norm-check-eqv-def norm-closure-def norm-step-def norm-check-eqv-counterexample-def
norm-check-eqv'-def norm-closure'-def norm-step'-def

interpretation *norm*: *equivalence-checker* σ Σ π ε Σ *Smart-Constructors-Normalization.norm* 𝔏 Σ
where *norm.check-eqv'* = *norm-check-eqv'* Σ
and *norm.check-eqv* = *norm-check-eqv* Σ
and *norm.check-eqv-counterexample* = *norm-check-eqv-counterexample* Σ
and *norm.closure'* = *norm-closure'* Σ
and *norm.closure* = *norm-closure* Σ
and *norm.step'* = *norm-step'* Σ
and *norm.step* = *norm-step* Σ
by *unfold-locales* (*auto simp*: *norm-defs trans*[*OF lang-norm*[*OF iffD2*[*OF ACI-norm-wf*]] *ACI-norm-lang*])

abbreviation *ext* Σ ≡ *None* # *map Some* (Σ :: $'a$ :: *linorder list*)

definition *any* **where** [*code del*]:
any ≡ λΣ. *formula.any* (*ext* Σ)
definition *pre-wf-formula* **where** [*code del*]:
pre-wf-formula ≡ λΣ. *formula.pre-wf-formula* (*ext* Σ)
definition *wf-formula* **where** [*code del*]:
wf-formula ≡ λΣ. *formula.wf-formula* (*ext* Σ)
definition *valid-ENC* **where** [*code del*]: *valid-ENC* ≡ λΣ. *formula.valid-ENC* (*ext* Σ)
definition *ENC* **where** [*code del*]: *ENC* ≡ λΣ. *formula.ENC* (*ext* Σ)

definition *rexp-of* **where** [*code del*]: *rexp-of* ≡ λΣ. *formula.rexp-of* (*ext* Σ)
definition *rexp-of-alt* **where** [*code del*]: *rexp-of-alt* ≡ λΣ. *formula.rexp-of-alt* (*ext* Σ)
definition *rexp-of'* **where** [*code del*]: *rexp-of'* ≡ λΣ. *formula.rexp-of'* (*ext* Σ)

lemmas *formula-defs* = *pre-wf-formula-def wf-formula-def any-def rexp-of-def rexp-of'-def rexp-of-alt-def ENC-def valid-ENC-def FOV-def SOV-def*

interpretation Φ: *formula ext* (Σ :: *'a* :: *linorder list*)
  where *alphabet.wf* (*set o σ* Σ) = *wf-rexp* Σ
  and *embed.rderiv* (ε Σ) = *norm-rderiv* Σ
  and *embed.rderiv-and-add* (ε Σ) = *norm-rderiv-and-add* Σ
  and *embed.samequot-exec* (ε Σ) = *quot* Σ
  and Φ.*any* = *any* Σ
  and Φ.*pre-wf-formula* = *pre-wf-formula* Σ
  and Φ.*wf-formula* = *wf-formula* Σ
  and Φ.*rexp-of* = *rexp-of* Σ
  and Φ.*rexp-of-alt* = *rexp-of-alt* Σ
  and Φ.*rexp-of'* = *rexp-of'* Σ
  and Φ.*valid-ENC* = *valid-ENC* Σ
  and Φ.*ENC* = *ENC* Σ
  by *unfold-locales*
    (*auto simp*: σ*-def* π*-def wf-rexp-def norm-rderiv-def norm-rderiv-and-add-def quot-def formula-defs*)

definition *check-eqv* **where**
check-eqv Σ *n* φ ψ ⟷ *wf-formula* Σ *n* (*FOr* φ ψ) ∧
  norm-check-eqv' (*ext* Σ) *n* (*rexp-of* Σ *n* (*norm* φ)) (*rexp-of* Σ *n* (*norm* ψ))

definition *check-eqv-counterexample* **where**
check-eqv-counterexample Σ *n* φ ψ =
  norm-check-eqv-counterexample (*ext* Σ) *n* (*rexp-of* Σ *n* (*norm* φ)) (*rexp-of* Σ *n* (*norm* ψ))

definition *check-eqv'* **where**
check-eqv' Σ *n* φ ψ ⟷ *wf-formula* Σ *n* (*FOr* φ ψ) ∧
  norm-check-eqv' (*ext* Σ) *n* (*rexp-of'* Σ *n* (*norm* φ)) (*rexp-of'* Σ *n* (*norm* ψ))

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

lemma *soundness*: *check-eqv* Σ *n* φ ψ ⟹ Φ.$lang_{WS1S}$ Σ *n* φ = Φ.$lang_{WS1S}$ Σ *n* ψ
  by (*rule box-equals*[*OF norm.soundness'*
  sym[*OF trans*[*OF* $lang_{WS1S}$*-rexp-of-norm*]] *sym*[*OF trans*[*OF* $lang_{WS1S}$*-rexp-of-norm*]]])
  (*auto simp*: *check-eqv-def split*: *sum.splits option.splits*)

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

lemma *soundness'*: *check-eqv'* Σ n φ $\psi \Longrightarrow \Phi.lang_{WS1S}$ Σ n $\varphi = \Phi.lang_{WS1S}$ Σ n ψ
  by (*rule box-equals*[*OF norm.soundness'*
  sym[*OF trans*[*OF* $lang_{WS1S}$*-rexp-of'-norm*]] *sym*[*OF trans*[*OF* $lang_{WS1S}$*-rexp-of'-norm*]]])
  (*auto simp*: *check-eqv'-def split*: *sum.splits option.splits*)

lemma *completeness*:
assumes $\Phi.lang_{WS1S}$ Σ n $\varphi = \Phi.lang_{WS1S}$ Σ n ψ *wf-formula* Σ n (*FOr* φ ψ)
shows *check-eqv* Σ n φ ψ
  using *assms*(*2*) **unfolding** *check-eqv-def*
  by (*intro conjI*[*OF assms*(*2*) *norm.completeness'*,
      OF box-equals[*OF assms*(*1*) $lang_{WS1S}$*-rexp-of-norm* $lang_{WS1S}$*-rexp-of-norm*]])
  (*auto split*: *sum.splits option.splits intro*!: Φ.*wf-rexp-of*)

lemma *completeness'*:
assumes $\Phi.lang_{WS1S}$ Σ n $\varphi = \Phi.lang_{WS1S}$ Σ n ψ *wf-formula* Σ n (*FOr* φ ψ)
shows *check-eqv'* Σ n φ ψ
  using *assms*(*2*) **unfolding** *check-eqv'-def*
  by (*intro conjI*[*OF assms*(*2*) *norm.completeness'*,
      OF box-equals[*OF assms*(*1*) $lang_{WS1S}$*-rexp-of'-norm* $lang_{WS1S}$*-rexp-of'-norm*]])
  (*auto split*: *sum.splits option.splits intro*!: Φ.*wf-rexp-of'*)

end