Local Lexing

Steven Obua

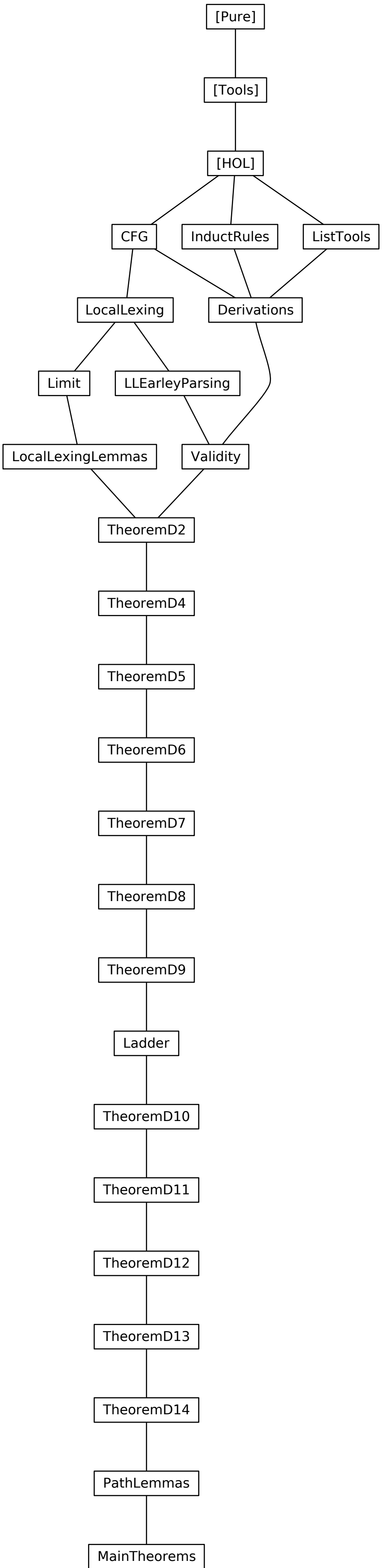
March 17, 2025

Abstract

This formalisation accompanies the paper Local Lexing[1], which introduces a novel parsing concept of the same name. The paper also gives a high-level algorithm for local lexing as an extension of Earley's algorithm. This formalisation proves the algorithm to be correct with respect to its local lexing semantics. As a special case, this formalisation thus also contains a proof of the correctness of Earley's algorithm. The paper contains a short outline of how this formalisation is organised.

Contents

[1] https://arxiv.org/abs/1702.03277


[Pure]
[Tools]
[HOL]
CFG
InductRules
ListTools
LocalLexing
Derivations
Limit
LLEarleyParsing
LocalLexingLemmas
Validity
TheoremD2
TheoremD4
TheoremD5
TheoremD6
TheoremD7
TheoremD8
TheoremD9
Ladder
TheoremD10
TheoremD11
TheoremD12
TheoremD13
TheoremD14
PathLemmas
MainTheorems

theory *CFG*
imports *Main*
begin

typedecl *symbol*

type-synonym *rule* = *symbol* × *symbol list*

type-synonym *sentence* = *symbol list*

locale *CFG* =
  fixes $\mathfrak{N}$ :: *symbol set*
  fixes $\mathfrak{T}$ :: *symbol set*
  fixes $\mathfrak{R}$ :: *rule set*
  fixes $\mathfrak{S}$ :: *symbol*
  assumes *disjunct-symbols*: $\mathfrak{N} \cap \mathfrak{T} = \{\}$
  assumes *startsymbol-dom*: $\mathfrak{S} \in \mathfrak{N}$
  assumes *validRules*: $\forall\ (N, \alpha) \in \mathfrak{R}.\ N \in \mathfrak{N} \land (\forall\ s \in set\ \alpha.\ s \in \mathfrak{N} \cup \mathfrak{T})$
begin

definition *is-terminal* :: *symbol* ⇒ *bool*
where
  is-terminal s = ($s \in \mathfrak{T}$)

definition *is-nonterminal* :: *symbol* ⇒ *bool*
where
  is-nonterminal s = ($s \in \mathfrak{N}$)

lemma *is-nonterminal-startsymbol*:*is-nonterminal* $\mathfrak{S}$
  ⟨*proof*⟩

definition *is-symbol* :: *symbol* ⇒ *bool*
where
  is-symbol s = (*is-terminal s* ∨ *is-nonterminal s*)

definition *is-sentence* :: *sentence* ⇒ *bool*
where
  is-sentence s = *list-all is-symbol s*

definition *is-word* :: *sentence* ⇒ *bool*
where
  is-word s = *list-all is-terminal s*

definition *derives1* :: *sentence* ⇒ *sentence* ⇒ *bool*
where
  derives1 u v =
    ($\exists\ x\ y\ N\ \alpha$.
        $u = x\ @\ [N]\ @\ y$
      $\land\ v = x\ @\ \alpha\ @\ y$

$\wedge$ *is-sentence* x
$\wedge$ *is-sentence* y
$\wedge$ $(N, \alpha) \in \mathfrak{R})$

definition *derivations1* :: (*sentence* $\times$ *sentence*) *set*
where
derivations1 = { (u,v) | u v. *derives1* u v }

definition *derivations* :: (*sentence* $\times$ *sentence*) *set*
where
derivations = *derivations1*^*

definition *derives* :: *sentence* $\Rightarrow$ *sentence* $\Rightarrow$ *bool*
where
derives u v = $((u, v) \in$ *derivations*)

definition *is-derivation* :: *sentence* $\Rightarrow$ *bool*
where
is-derivation u = *derives* $[\mathfrak{S}]$ u

definition $\mathcal{L}$ :: *sentence set*
where
$\mathcal{L}$ = { v | v. *is-word* v $\wedge$ *is-derivation* v}

definition $\mathcal{L}_P$ :: *sentence set*
where
$\mathcal{L}_P$ = { u | u v. *is-word* u $\wedge$ *is-derivation* $(u@v)$ }

end

end
theory *LocalLexing*
imports *CFG*
begin

typedecl *character*

type-synonym *lexer* = *character list* $\Rightarrow$ *nat* $\Rightarrow$ *nat set*

type-synonym *token* = *symbol* $\times$ *character list*

type-synonym *tokens* = *token list*

definition *terminal-of-token* :: *token* $\Rightarrow$ *symbol*
where
terminal-of-token t = *fst* t

definition *terminals* :: *tokens* $\Rightarrow$ *sentence*
where

terminals ts = *map terminal-of-token ts*

definition *chars-of-token* :: *token* ⇒ *character list*
where
chars-of-token t = *snd t*

fun *chars* :: *tokens* ⇒ *character list*
where
chars [] = []
| *chars* (*t*#*ts*) = (*chars-of-token t*) @ (*chars ts*)

fun *charslength* :: *tokens* ⇒ *nat*
where
charslength cs = *length* (*chars cs*)

definition *is-lexer* :: *lexer* ⇒ *bool*
where
is-lexer lexer =
$(\forall\ D\ p\ l.\ (p \leq \mathit{length}\ D \wedge l \in \mathit{lexer}\ D\ p \longrightarrow p + l \leq \mathit{length}\ D) \wedge$
$(p > \mathit{length}\ D \longrightarrow \mathit{lexer}\ D\ p = \{\}))$

type-synonym *selector* = *token set* ⇒ *token set* ⇒ *token set*

definition *is-selector* :: *selector* ⇒ *bool*
where
$\mathit{is\text{-}selector}\ \mathit{sel} = (\forall\ A\ B.\ A \subseteq B \longrightarrow (A \subseteq \mathit{sel}\ A\ B \wedge \mathit{sel}\ A\ B \subseteq B))$

fun *by-length* :: *nat* ⇒ *tokens set* ⇒ *tokens set*
where
$\mathit{by\text{-}length}\ l\ \mathit{tss} = \{\ \mathit{ts}\ .\ \mathit{ts} \in \mathit{tss} \wedge \mathit{length}\ (\mathit{chars}\ \mathit{ts}) = l\ \}$

fun *funpower* :: ('*a* ⇒ '*a*) ⇒ *nat* ⇒ ('*a* ⇒ '*a*)
where
funpower f 0 x = *x*
| *funpower f* (*Suc n*) *x* = *f* (*funpower f n x*)

definition *natUnion* :: (*nat* ⇒ '*a set*) ⇒ '*a set*
where
$\mathit{natUnion}\ f = \bigcup\ \{\ f\ n \mid n.\ \mathit{True}\ \}$

definition *limit* :: ('*a set* ⇒ '*a set*) ⇒ '*a set* ⇒ '*a set*
where
$\mathit{limit}\ f\ x = \mathit{natUnion}\ (\lambda\ n.\ \mathit{funpower}\ f\ n\ x)$

locale *LocalLexing* = *CFG* +
fixes *Lex* :: *symbol* ⇒ *lexer*
fixes *Sel* :: *selector*
assumes *Lex-is-lexer*: $\forall\ t \in \mathfrak{T}.\ \mathit{is\text{-}lexer}\ (\mathit{Lex}\ t)$
assumes *Sel-is-selector*: *is-selector Sel*

fixes *Doc* :: *character list*
begin

definition *admissible* :: *tokens* ⇒ *bool*
where
admissible ts = (*terminals ts* ∈ $\mathcal{L}_P$)

definition *Append* :: *token set* ⇒ *nat* ⇒ *tokens set* ⇒ *tokens set*
where
Append Z k P = *P* ∪
{ *p* @ [*t*] | *p t*. *p* ∈ *by-length k P* ∧ *t* ∈ *Z* ∧ *admissible* (*p* @ [*t*])}

definition $\mathcal{X}$ :: *nat* ⇒ *token set*
where
$\mathcal{X}$ *k* = {(*t*, ω) | *t l* ω. *t* ∈ $\mathfrak{T}$ ∧ *l* ∈ *Lex t Doc k* ∧ ω = *take l* (*drop k Doc*)}

definition $\mathcal{W}$ :: *tokens set* ⇒ *nat* ⇒ *token set*
where
$\mathcal{W}$ *P k* = { *u*. *u* ∈ $\mathcal{X}$ *k* ∧ (∃ *p* ∈ *by-length k P*. *admissible* (*p*@[*u*])) }

definition $\mathcal{Y}$ :: *token set* ⇒ *tokens set* ⇒ *nat* ⇒ *token set*
where
$\mathcal{Y}$ *T P k* = *Sel T* ($\mathcal{W}$ *P k*)

fun $\mathcal{P}$ :: *nat* ⇒ *nat* ⇒ *tokens set*
and $\mathcal{Q}$ :: *nat* ⇒ *tokens set*
and $\mathcal{Z}$ :: *nat* ⇒ *nat* ⇒ *token set*
where
$\mathcal{P}$ *0 0* = {[]}
| $\mathcal{P}$ *k* (*Suc u*) = *limit* (*Append* ($\mathcal{Z}$ *k* (*Suc u*)) *k*) ($\mathcal{P}$ *k u*)
| $\mathcal{P}$ (*Suc k*) *0* = $\mathcal{Q}$ *k*
| $\mathcal{Z}$ *k 0* = {}
| $\mathcal{Z}$ *k* (*Suc u*) = $\mathcal{Y}$ ($\mathcal{Z}$ *k u*) ($\mathcal{P}$ *k u*) *k*
| $\mathcal{Q}$ *k* = *natUnion* ($\mathcal{P}$ *k*)

definition $\mathfrak{P}$ :: *tokens set*
where
$\mathfrak{P}$ = $\mathcal{Q}$ (*length Doc*)

definition *ll* :: *tokens set*
where
ll = { *p* . *p* ∈ $\mathfrak{P}$ ∧ *charslength p* = *length Doc* ∧ *terminals p* ∈ $\mathcal{L}$ }

end

end
theory *LLEarleyParsing*
imports *LocalLexing*
begin

datatype *item* =
Item
(*item-rule*: *rule*)
(*item-dot* : *nat*)
(*item-origin* : *nat*)
(*item-end* : *nat*)

type-synonym *items* = *item set*

definition *item-nonterminal* :: *item* ⇒ *symbol*
where
item-nonterminal x = *fst* (*item-rule x*)

definition *item-rhs* :: *item* ⇒ *sentence*
where
item-rhs x = *snd* (*item-rule x*)

definition *item-α* :: *item* ⇒ *sentence*
where
item-α x = *take* (*item-dot x*) (*item-rhs x*)

definition *item-β* :: *item* ⇒ *sentence*
where
item-β x = *drop* (*item-dot x*) (*item-rhs x*)

definition *init-item* :: *rule* ⇒ *nat* ⇒ *item*
where
init-item r k = *Item r 0 k k*

definition *is-complete* :: *item* ⇒ *bool*
where
is-complete x = (*item-dot x* ≥ *length* (*item-rhs x*))

definition *next-symbol* :: *item* ⇒ *symbol option*
where
next-symbol x = (*if is-complete x then None else Some* ((*item-rhs x*) ! (*item-dot x*)))

definition *inc-item* :: *item* ⇒ *nat* ⇒ *item*
where
inc-item x k = *Item* (*item-rule x*) (*item-dot x* + *1*) (*item-origin x*) *k*

definition *bin* :: *items* ⇒ *nat* ⇒ *items*
where
bin I k = { *x* . *x* ∈ *I* ∧ *item-end x* = *k* }

context *LocalLexing* **begin**

definition $\mathit{Init} :: \mathit{items}$
where
$\mathit{Init} = \{\ \mathit{init\text{-}item}\ r\ 0 \mid r.\ r \in \mathfrak{R} \wedge \mathit{fst}\ r = \mathfrak{S}\ \}$

definition $\mathit{Predict} :: \mathit{nat} \Rightarrow \mathit{items} \Rightarrow \mathit{items}$
where
$\mathit{Predict}\ k\ I = I\ \cup$
$\{\ \mathit{init\text{-}item}\ r\ k \mid r\ x.\ r \in \mathfrak{R} \wedge x \in \mathit{bin}\ I\ k\ \wedge$
$\mathit{next\text{-}symbol}\ x = \mathit{Some}(\mathit{fst}\ r)\ \}$

definition $\mathit{Complete} :: \mathit{nat} \Rightarrow \mathit{items} \Rightarrow \mathit{items}$
where
$\mathit{Complete}\ k\ I = I \cup \{\ \mathit{inc\text{-}item}\ x\ k \mid x\ y.$
$x \in \mathit{bin}\ I\ (\mathit{item\text{-}origin}\ y) \wedge y \in \mathit{bin}\ I\ k \wedge \mathit{is\text{-}complete}\ y\ \wedge$
$\mathit{next\text{-}symbol}\ x = \mathit{Some}\ (\mathit{item\text{-}nonterminal}\ y)\ \}$

definition $\mathit{TokensAt} :: \mathit{nat} \Rightarrow \mathit{items} \Rightarrow \mathit{token\ set}$
where
$\mathit{TokensAt}\ k\ I = \{\ (t,\ s) \mid t\ s\ x\ l.\ x \in \mathit{bin}\ I\ k\ \wedge$
$\mathit{next\text{-}symbol}\ x = \mathit{Some}\ t \wedge \mathit{is\text{-}terminal}\ t\ \wedge$
$l \in \mathit{Lex}\ t\ \mathit{Doc}\ k \wedge s = \mathit{take}\ l\ (\mathit{drop}\ k\ \mathit{Doc})\ \}$

definition $\mathit{Tokens} :: \mathit{nat} \Rightarrow \mathit{token\ set} \Rightarrow \mathit{items} \Rightarrow \mathit{token\ set}$
where
$\mathit{Tokens}\ k\ T\ I = \mathit{Sel}\ T\ (\mathit{TokensAt}\ k\ I)$

definition $\mathit{Scan} :: \mathit{token\ set} \Rightarrow \mathit{nat} \Rightarrow \mathit{items} \Rightarrow \mathit{items}$
where
$\mathit{Scan}\ T\ k\ I = I\ \cup$
$\{\ \mathit{inc\text{-}item}\ x\ (k + \mathit{length}\ c) \mid x\ t\ c.\ x \in \mathit{bin}\ I\ k \wedge (t,\ c) \in T\ \wedge$
$\mathit{next\text{-}symbol}\ x = \mathit{Some}\ t\ \}$

definition $\pi :: \mathit{nat} \Rightarrow \mathit{token\ set} \Rightarrow \mathit{items} \Rightarrow \mathit{items}$
where
$\pi\ k\ T\ I =$
$\mathit{limit}\ (\lambda\ I.\ \mathit{Scan}\ T\ k\ (\mathit{Complete}\ k\ (\mathit{Predict}\ k\ I)))\ I$

fun $\mathcal{J} :: \mathit{nat} \Rightarrow \mathit{nat} \Rightarrow \mathit{items}$
and $\mathcal{I} :: \mathit{nat} \Rightarrow \mathit{items}$
and $\mathcal{T} :: \mathit{nat} \Rightarrow \mathit{nat} \Rightarrow \mathit{token\ set}$
where
$\mathcal{J}\ 0\ 0 = \pi\ 0\ \{\}\ \mathit{Init}$
$\mid \mathcal{J}\ k\ (\mathit{Suc}\ u) = \pi\ k\ (\mathcal{T}\ k\ (\mathit{Suc}\ u))\ (\mathcal{J}\ k\ u)$
$\mid \mathcal{J}\ (\mathit{Suc}\ k)\ 0 = \pi\ (\mathit{Suc}\ k)\ \{\}\ (\mathcal{I}\ k)$
$\mid \mathcal{T}\ k\ 0 = \{\}$
$\mid \mathcal{T}\ k\ (\mathit{Suc}\ u) = \mathit{Tokens}\ k\ (\mathcal{T}\ k\ u)\ (\mathcal{J}\ k\ u)$
$\mid \mathcal{I}\ k = \mathit{natUnion}\ (\mathcal{J}\ k)$

definition $\mathfrak{I} :: \mathit{items}$

where
$\mathfrak{I} = \mathcal{I}$ (*length Doc*)

definition *is-finished* :: *item* ⇒ *bool* **where**
is-finished x = (*item-nonterminal x* = $\mathfrak{S}$ ∧ *item-origin x* = *0* ∧ *item-end x* = *length Doc* ∧
is-complete x)

definition *earley-recognised* :: *bool*
where
earley-recognised = (∃ *x* ∈ $\mathfrak{I}$. *is-finished x*)

end

end
theory *Limit*
imports *LocalLexing*
begin

definition *setmonotone* :: (*'a set* ⇒ *'a set*) ⇒ *bool*
where
setmonotone f = (∀ *X*. *X* ⊆ *f X*)

lemma *setmonotone-funpower*: *setmonotone f* ⟹ *setmonotone* (*funpower f n*)
⟨*proof*⟩

lemma *subset-setmonotone*: *setmonotone f* ⟹ *X* ⊆ *f X*
⟨*proof*⟩

lemma *elem-setmonotone*: *setmonotone f* ⟹ *x* ∈ *X* ⟹ *x* ∈ *f X*
⟨*proof*⟩

lemma *elem-natUnion*: (∀ *n*. *x* ∈ *f n*) ⟹ *x* ∈ *natUnion f*
⟨*proof*⟩

lemma *subset-natUnion*: (∀ *n*. *X* ⊆ *f n*) ⟹ *X* ⊆ *natUnion f*
⟨*proof*⟩

lemma *setmonotone-limit*:
assumes *fmono*: *setmonotone f*
shows *setmonotone* (*limit f*)
⟨*proof*⟩

lemma[*simp*]: *funpower id n* = *id*
⟨*proof*⟩

lemma[*simp*]: *limit id* = *id*
⟨*proof*⟩

lemma *natUnion-decompose*[*consumes 1*, *case-names Decompose*]:
  assumes *p*: *p* ∈ *natUnion S*
  assumes *decompose*: ⋀ *n p*. *p* ∈ *S n* ⟹ *P p*
  shows *P p*
⟨*proof*⟩

lemma *limit-induct*[*consumes 1*, *case-names Init Iterate*]:
  assumes *p*: (*p* :: *'a*) ∈ *limit f X*
  assumes *init*: ⋀ *p*. *p* ∈ *X* ⟹ *P p*
  assumes *iterate*: ⋀ *p Y*. (⋀ *q* . *q* ∈ *Y* ⟹ *P q*) ⟹ *p* ∈ *f Y* ⟹ *P p*
  shows *P p*
⟨*proof*⟩

definition *chain* :: (*nat* ⇒ *'a set*) ⇒ *bool*
where
  chain C = (∀ *i*. *C i* ⊆ *C* (*i* + *1*))

definition *continuous* :: (*'a set* ⇒ *'b set*) ⇒ *bool*
where
  continuous f = (∀ *C*. *chain C* ⟶ (*chain* (*f o C*) ∧ *f* (*natUnion C*) = *natUnion* (*f o C*)))

lemma *continuous-apply*:
  continuous f ⟹ *chain C* ⟹ *f* (*natUnion C*) = *natUnion* (*f o C*)
⟨*proof*⟩

lemma *continuous-imp-mono*:
  assumes *continuous*: *continuous f*
  shows *mono f*
⟨*proof*⟩

lemma *mono-maps-chain-to-chain*:
  assumes *f*: *mono f*
  assumes *C*: *chain C*
  shows *chain* (*f o C*)
⟨*proof*⟩

lemma *natUnion-upperbound*:
  (⋀ *n*. *f n* ⊆ *G*) ⟹ (*natUnion f*) ⊆ *G*
⟨*proof*⟩

lemma *funpower-upperbound*:
  (⋀ *I*. *I* ⊆ *G* ⟹ *f I* ⊆ *G*) ⟹ *I* ⊆ *G* ⟹ *funpower f n I* ⊆ *G*
⟨*proof*⟩

lemma *limit-upperbound*:
  (⋀ *I*. *I* ⊆ *G* ⟹ *f I* ⊆ *G*) ⟹ *I* ⊆ *G* ⟹ *limit f I* ⊆ *G*
⟨*proof*⟩

lemma *elem-limit-simp*: $x \in limit\ f\ X = (\exists\ n.\ x \in funpower\ f\ n\ X)$
⟨*proof*⟩

definition *pointwise* :: $('a\ set \Rightarrow 'b\ set) \Rightarrow bool$ **where**
pointwise $f = (\forall\ X.\ f\ X = \bigcup\ \{\ f\ \{x\}\ |\ x.\ x \in X\})$

lemma *pointwise-simp*:
assumes f: *pointwise* f
shows $f\ X = \bigcup\ \{\ f\ \{x\}\ |\ x.\ x \in X\}$
⟨*proof*⟩

lemma *natUnion-elem*: $x \in f\ n \Longrightarrow x \in natUnion\ f$
⟨*proof*⟩

lemma *limit-elem*: $x \in funpower\ f\ n\ X \Longrightarrow x \in limit\ f\ X$
⟨*proof*⟩

lemma *limit-step-pointwise*:
assumes x: $x \in limit\ f\ X$
assumes f: *pointwise* f
assumes y: $y \in f\ \{x\}$
shows $y \in limit\ f\ X$
⟨*proof*⟩

definition *pointbase* :: $('a\ set \Rightarrow 'b\ set) \Rightarrow 'a\ set \Rightarrow 'b\ set$ **where**
pointbase $F\ I = \bigcup\ \{\ F\ X\ |\ X.\ finite\ X \land X \subseteq I\ \}$

definition *pointbased* :: $('a\ set \Rightarrow 'b\ set) \Rightarrow bool$ **where**
pointbased $f = (\exists\ F.\ f = pointbase\ F)$

lemma *pointwise-implies-pointbased*:
assumes *pointwise*: *pointwise* f
shows *pointbased* f
⟨*proof*⟩

lemma *pointbase-is-mono*:
mono (*pointbase* f)
⟨*proof*⟩

lemma *chain-implies-mono*: $chain\ C \Longrightarrow mono\ C$
⟨*proof*⟩

lemma *chain-cover-witness*: $finite\ X \Longrightarrow chain\ C \Longrightarrow X \subseteq natUnion\ C \Longrightarrow \exists\ n.\ X \subseteq C\ n$
⟨*proof*⟩

lemma *pointbase-is-continuous*:
continuous (*pointbase* f)
⟨*proof*⟩

lemma *pointbased-implies-continuous*:
pointbased f $\Longrightarrow$ *continuous f*
⟨*proof*⟩

lemma *setmonotone-implies-chain-funpower*:
assumes *setmonotone*: *setmonotone f*
shows *chain* (λ *n. funpower f n I*)
⟨*proof*⟩

lemma *natUnion-subset*: ($\bigwedge$ *n.* $\exists$ *m. f n* $\subseteq$ *g m*) $\Longrightarrow$ *natUnion f* $\subseteq$ *natUnion g*
⟨*proof*⟩

lemma *natUnion-eq*[*case-names Subset Superset*]:
($\bigwedge$ *n.* $\exists$ *m. f n* $\subseteq$ *g m*) $\Longrightarrow$ ($\bigwedge$ *n.* $\exists$ *m. g n* $\subseteq$ *f m*) $\Longrightarrow$ *natUnion f* = *natUnion g*
⟨*proof*⟩

lemma *natUnion-shift*[*symmetric*]:
assumes *chain*: *chain C*
shows *natUnion C* = *natUnion* (λ *n. C* (*n* + *m*))
⟨*proof*⟩

definition *regular* :: (*'a set* $\Rightarrow$ *'a set*) $\Rightarrow$ *bool*
where
regular f = (*setmonotone f* $\wedge$ *continuous f*)

lemma *regular-fixpoint*:
assumes *regular*: *regular f*
shows *f* (*limit f I*) = *limit f I*
⟨*proof*⟩

lemma *fix-is-fix-of-limit*:
assumes *fixpoint*: *f I* = *I*
shows *limit f I* = *I*
⟨*proof*⟩

lemma *limit-is-idempotent*: *regular f* $\Longrightarrow$ *limit f* (*limit f I*) = *limit f I*
⟨*proof*⟩

definition *mk-regular1* :: (*'b* $\Rightarrow$ *'a* $\Rightarrow$ *bool*) $\Rightarrow$ (*'b* $\Rightarrow$ *'a* $\Rightarrow$ *'a*) $\Rightarrow$ *'a set* $\Rightarrow$ *'a set*
where
mk-regular1 P F I = *I* $\cup$ { *F q x* | *q x. x* $\in$ *I* $\wedge$ *P q x* }

definition *mk-regular2* :: (*'b* $\Rightarrow$ *'a* $\Rightarrow$ *'a* $\Rightarrow$ *bool*) $\Rightarrow$ (*'b* $\Rightarrow$ *'a* $\Rightarrow$ *'a* $\Rightarrow$ *'a*) $\Rightarrow$ *'a set* $\Rightarrow$ *'a set* **where**
mk-regular2 P F I = *I* $\cup$ { *F q x y* | *q x y. x* $\in$ *I* $\wedge$ *y* $\in$ *I* $\wedge$ *P q x y* }

lemma *setmonotone-mk-regular1*: *setmonotone* (*mk-regular1 P F*)

⟨*proof*⟩

lemma *setmonotone-mk-regular2*: *setmonotone* (*mk-regular2 P F*)
⟨*proof*⟩

lemma *pointbased-mk-regular1*: *pointbased* (*mk-regular1 P F*)
⟨*proof*⟩

lemma *pointbased-mk-regular2*: *pointbased* (*mk-regular2 P F*)
⟨*proof*⟩

lemma *regular1*:*regular* (*mk-regular1 P F*)
⟨*proof*⟩

lemma *regular2*: *regular* (*mk-regular2 P F*)
⟨*proof*⟩

lemma *continuous-comp*:
assumes *f*: *continuous f*
assumes *g*: *continuous g*
shows *continuous* (*g o f*)
⟨*proof*⟩

lemma *setmonotone-comp*:
assumes *f*: *setmonotone f*
assumes *g*: *setmonotone g*
shows *setmonotone* (*g o f*)
⟨*proof*⟩

lemma *regular-comp*:
assumes *f*: *regular f*
assumes *g*: *regular g*
shows *regular* (*g o f*)
⟨*proof*⟩

lemma *setmonotone-id*[*simp*]: *setmonotone id*
⟨*proof*⟩

lemma *continuous-id*[*simp*]: *continuous id*
⟨*proof*⟩

lemma *regular-id*[*simp*]: *regular id*
⟨*proof*⟩

lemma *regular-funpower*: *regular f* $\Longrightarrow$ *regular* (*funpower f n*)
⟨*proof*⟩

lemma *mono-id*[*simp*]: *mono id*
⟨*proof*⟩

lemma *mono-funpower*:
assumes *mono*: *mono f*
shows *mono* (*funpower f n*)
⟨*proof*⟩

lemma *mono-limit*:
assumes *mono*: *mono f*
shows *mono* (*limit f*)
⟨*proof*⟩

lemma *continuous-funpower*:
assumes *continuous*: *continuous f*
shows *continuous* (*funpower f n*)
⟨*proof*⟩

lemma *natUnion-swap*:
$natUnion\ (\lambda\ i.\ natUnion\ (\lambda\ j.\ f\ i\ j)) = natUnion\ (\lambda\ j.\ natUnion\ (\lambda\ i.\ f\ i\ j))$
⟨*proof*⟩

lemma *continuous-limit*:
assumes *continuous*: *continuous f*
shows *continuous* (*limit f*)
⟨*proof*⟩

lemma *regular-limit*: *regular f* $\Longrightarrow$ *regular* (*limit f*)
⟨*proof*⟩

lemma *regular-implies-mono*: *regular f* $\Longrightarrow$ *mono f*
⟨*proof*⟩

lemma *regular-implies-setmonotone*: *regular f* $\Longrightarrow$ *setmonotone f*
⟨*proof*⟩

lemma *regular-implies-continuous*: *regular f* $\Longrightarrow$ *continuous f*
⟨*proof*⟩

end
theory *LocalLexingLemmas*
imports *LocalLexing Limit*
begin

context *LocalLexing* **begin**

lemma[*simp*]: *setmonotone* (*Append Z k*) ⟨*proof*⟩

lemma *subset-*$\mathcal{P}$*Suc*: $\mathcal{P}\ k\ u \subseteq \mathcal{P}\ k\ (Suc\ u)$
⟨*proof*⟩

lemma *subset-fSuc-strict*:
assumes *f*: $\bigwedge$ *u*. $f\ u \subseteq f\ (Suc\ u)$
shows $u < v \Longrightarrow f\ u \subseteq f\ v$
⟨*proof*⟩

lemma *subset-fSuc*:
assumes *f*: $\bigwedge$ *u*. $f\ u \subseteq f\ (Suc\ u)$
shows $u \leq v \Longrightarrow f\ u \subseteq f\ v$
⟨*proof*⟩

lemma *subset-$\mathcal{P}$k*: $u \leq v \Longrightarrow \mathcal{P}\ k\ u \subseteq \mathcal{P}\ k\ v$
⟨*proof*⟩

lemma *subset-$\mathcal{P}\mathcal{Q}$k*: $\mathcal{P}\ k\ u \subseteq \mathcal{Q}\ k$ ⟨*proof*⟩

lemma *subset-$\mathcal{Q}\mathcal{P}$Suc*: $\mathcal{Q}\ k \subseteq \mathcal{P}\ (Suc\ k)\ u$
⟨*proof*⟩

lemma *subset-$\mathcal{Q}$Suc*: $\mathcal{Q}\ k \subseteq \mathcal{Q}\ (Suc\ k)$
⟨*proof*⟩

lemma *subset-$\mathcal{Q}$*: $i \leq j \Longrightarrow \mathcal{Q}\ i \subseteq \mathcal{Q}\ j$
⟨*proof*⟩

lemma *empty-$\mathcal{X}$*[*simp*]: $k > length\ Doc \Longrightarrow \mathcal{X}\ k = \{\}$
⟨*proof*⟩

lemma *Sel-empty*[*simp*]: $Sel\ \{\}\ \{\} = \{\}$
⟨*proof*⟩

lemma *empty-$\mathcal{Z}$*[*simp*]: $k > length\ Doc \Longrightarrow \mathcal{Z}\ k\ u = \{\}$
⟨*proof*⟩

lemma[*simp*]: $Append\ \{\}\ k = id$ ⟨*proof*⟩

lemma[*simp*]: $k > length\ Doc \Longrightarrow \mathcal{P}\ k\ v = \mathcal{P}\ k\ 0$
⟨*proof*⟩

lemma *$\mathcal{Q}$SucEq*: $k \geq length\ Doc \Longrightarrow \mathcal{Q}\ (Suc\ k) = \mathcal{Q}\ k$
⟨*proof*⟩

lemma *$\mathcal{Q}$-converges*:
assumes *k*: $k \geq length\ Doc$
shows $\mathcal{Q}\ k = \mathfrak{P}$
⟨*proof*⟩

lemma *$\mathfrak{P}$-covers-$\mathcal{Q}$*: $\mathcal{Q}\ k \subseteq \mathfrak{P}$
⟨*proof*⟩

lemma *Sel-upper-bound*: $A \subseteq B \Longrightarrow Sel\ A\ B \subseteq B$
⟨*proof*⟩

lemma *Sel-lower-bound*: $A \subseteq B \Longrightarrow A \subseteq Sel\ A\ B$
⟨*proof*⟩

lemma $\mathfrak{P}$*-covers-*$\mathcal{P}$: $\mathcal{P}\ k\ u \subseteq \mathfrak{P}$
⟨*proof*⟩

lemma $\mathcal{W}$*-montone*: $P \subseteq Q \Longrightarrow \mathcal{W}\ P\ k \subseteq \mathcal{W}\ Q\ k$
⟨*proof*⟩

lemma *Sel-precondition*:
$\mathcal{Z}\ k\ u \subseteq \mathcal{W}\ (\mathcal{P}\ k\ u)\ k$
⟨*proof*⟩

lemma $\mathcal{W}$*-bounded-by-*$\mathcal{X}$: $\mathcal{W}\ P\ k \subseteq \mathcal{X}\ k$
⟨*proof*⟩

lemma $\mathcal{Z}$*-subset-*$\mathcal{X}$: $\mathcal{Z}\ k\ n \subseteq \mathcal{X}\ k$
⟨*proof*⟩

lemma $\mathcal{Z}$*-subset-Suc*: $\mathcal{Z}\ k\ n \subseteq \mathcal{Z}\ k\ (Suc\ n)$
⟨*proof*⟩

lemma $\mathcal{Y}$*-upper-bound*: $\mathcal{Y}\ (\mathcal{Z}\ k\ u)\ (\mathcal{P}\ k\ u)\ k \subseteq \mathcal{W}\ \ (\mathcal{P}\ k\ u)\ k$
⟨*proof*⟩

lemma $\mathfrak{P}$*-induct*[*consumes 1*, *case-names Base Induct*]:
assumes *p*: $p \in \mathfrak{P}$
assumes *base*: $P\ []$
assumes *induct*: $\bigwedge\ p\ k\ u.\ (\bigwedge\ q.\ q \in \mathcal{P}\ k\ u \Longrightarrow P\ q) \Longrightarrow p \in \mathcal{P}\ k\ (Suc\ u) \Longrightarrow P\ p$
shows $P\ p$
⟨*proof*⟩

lemma *Append-mono*: $U \subseteq V \Longrightarrow P \subseteq Q \Longrightarrow Append\ U\ k\ P \subseteq Append\ V\ k\ Q$
⟨*proof*⟩

lemma *pointwise-Append*: *pointwise* (*Append T k*)
⟨*proof*⟩

lemma *regular-Append*: *regular* (*Append T k*)
⟨*proof*⟩

end

end
theory *InductRules*

imports *Main*
begin

lemma *disjCases2*[*consumes 1*, *case-names 1 2*]:
assumes *AB*: $A \vee B$
and *AP*: $A \Longrightarrow P$
and *BP*: $B \Longrightarrow P$
shows P
⟨*proof*⟩

lemma *disjCases3*[*consumes 1*, *case-names 1 2 3*]:
assumes *AB*: $A \vee B \vee C$
and *AP*: $A \Longrightarrow P$
and *BP*: $B \Longrightarrow P$
and *CP*: $C \Longrightarrow P$
shows P
⟨*proof*⟩

lemma *disjCases4*[*consumes 1*, *case-names 1 2 3 4*]:
assumes *AB*: $A \vee B \vee C \vee D$
and *AP*: $A \Longrightarrow P$
and *BP*: $B \Longrightarrow P$
and *CP*: $C \Longrightarrow P$
and *DP*: $D \Longrightarrow P$
shows P
⟨*proof*⟩

lemma *disjCases5*[*consumes 1*, *case-names 1 2 3 4 5*]:
assumes *AB*: $A \vee B \vee C \vee D \vee E$
and *AP*: $A \Longrightarrow P$
and *BP*: $B \Longrightarrow P$
and *CP*: $C \Longrightarrow P$
and *DP*: $D \Longrightarrow P$
and *EP*: $E \Longrightarrow P$
shows P
⟨*proof*⟩

lemma *minimal-witness-ex*:
assumes *k*: $P\ (k::nat)$
shows $\exists\ k0.\ k0 \le k \wedge P\ k0 \wedge (\forall\ k.\ k < k0 \longrightarrow \neg\ (P\ k))$
⟨*proof*⟩

lemma *minimal-witness*[*consumes 1*, *case-names Minimal*]:
assumes $P\ (k::nat)$
and $\bigwedge K.\ K \le k \Longrightarrow P\ K \Longrightarrow (\bigwedge k.\ k < K \Longrightarrow \neg\ (P\ k)) \Longrightarrow Q$
shows Q
⟨*proof*⟩

lemma *ex-minimal-witness*[*consumes 1*, *case-names Minimal*]:

assumes $\exists\ k.\ P\ (k{::}nat)$
and $\bigwedge K.\ P\ K \Longrightarrow (\bigwedge k.\ k < K \Longrightarrow \neg\ (P\ k)) \Longrightarrow Q$
shows Q
⟨*proof*⟩

end
theory *ListTools*
imports *Main*
begin

definition *is-first* :: $'a \Rightarrow 'a\ list \Rightarrow bool$
where
is-first $x\ u = (\exists\ v.\ u = [x]@v)$

definition *is-last* :: $'a \Rightarrow 'a\ list \Rightarrow bool$
where
is-last $x\ u = (\exists\ v.\ u = v@[x])$

definition *is-prefix* :: $'a\ list \Rightarrow 'a\ list \Rightarrow bool$
where
is-prefix $u\ v = (\exists\ w.\ u@w = v)$

definition *is-proper-prefix* :: $'a\ list \Rightarrow 'a\ list \Rightarrow bool$
where
is-proper-prefix $u\ v = (\exists\ w.\ w \neq [] \wedge u@w = v)$

lemma *is-prefix-eq-proper-prefix*: *is-prefix* $a\ b = (a = b \vee$ *is-proper-prefix* $a\ b)$
⟨*proof*⟩

lemma *is-proper-prefix-eq-prefix*: *is-proper-prefix* $a\ b = (a \neq b \wedge$ *is-prefix* $a\ b)$
⟨*proof*⟩

definition *is-suffix* :: $'a\ list \Rightarrow 'a\ list \Rightarrow bool$
where
is-suffix $u\ v = (\exists\ w.\ w@u = v)$

definition *is-proper-suffix* :: $'a\ list \Rightarrow 'a\ list \Rightarrow bool$
where
is-proper-suffix $u\ v = (\exists\ w.\ w \neq [] \wedge w@u = v)$

lemma *is-suffix-eq-proper-suffix*: *is-suffix* $a\ b = (a = b \vee$ *is-proper-suffix* $a\ b)$
⟨*proof*⟩

lemma *is-proper-suffix-eq-suffix*: *is-proper-suffix* $a\ b = (a \neq b \wedge$ *is-suffix* $a\ b)$
⟨*proof*⟩

lemma *is-prefix-unsplit*: *is-prefix* $u\ a \Longrightarrow u\ @\ (drop\ (length\ u)\ a) = a$
⟨*proof*⟩

lemma *le-take-same*: $i \leq j \Longrightarrow take\ j\ a = take\ j\ b \Longrightarrow take\ i\ a = take\ i\ b$
⟨*proof*⟩

lemma *is-first-drop-length*:
assumes $k \leq length\ a$
and $k > length\ u$
and $v = X\#w$
and $take\ k\ a = take\ k\ (u@v)$
shows $is\text{-}first\ X\ (drop\ (length\ u)\ a)$
⟨*proof*⟩

lemma *is-first-cons*: $is\text{-}first\ x\ (y\#ys) = (x = y)$
⟨*proof*⟩

lemma *list-all-pos-neg-ex*: $list\text{-}all\ P\ D \Longrightarrow \neg\ (list\text{-}all\ Q\ D) \Longrightarrow$
$\exists\ k.\ k < length\ D \wedge P(D\ !\ k) \wedge \neg(Q(D\ !\ k))$
⟨*proof*⟩

lemma *split-list-at*: $k < length\ D \Longrightarrow D = (take\ k\ D)@[D\ !\ k]@(drop\ (Suc\ k)\ D)$
⟨*proof*⟩

lemma *take-eq-take-append*: $i \leq j \Longrightarrow j \leq length\ a \Longrightarrow \exists\ u.\ take\ j\ a = take\ i\ a$ $@\ u$
⟨*proof*⟩

lemma *is-proper-suffix-length-cmp*: $is\text{-}proper\text{-}suffix\ a\ b \Longrightarrow length\ a < length\ b$
⟨*proof*⟩

end
theory *Derivations*
imports *CFG ListTools InductRules*
begin

context *CFG* **begin**

lemma [*simp*]: $is\text{-}terminal\ t \Longrightarrow is\text{-}symbol\ t$
⟨*proof*⟩

lemma [*simp*]: *is-sentence* [] ⟨*proof*⟩

lemma [*simp*]: *is-word* [] ⟨*proof*⟩

lemma [*simp*]: $is\text{-}word\ u \Longrightarrow is\text{-}sentence\ u$
⟨*proof*⟩

definition *leftderives1* :: $sentence \Rightarrow sentence \Rightarrow bool$
where

leftderives1 u v =
  (∃ *x y N α*.
      u = *x* @ [*N*] @ *y*
    ∧ *v* = *x* @ *α* @ *y*
    ∧ *is-word x*
    ∧ *is-sentence y*
    ∧ (*N*, *α*) ∈ 𝔑)

lemma *leftderives1-implies-derives1* [*simp*]: *leftderives1 u v* ⟹ *derives1 u v*
⟨*proof*⟩

definition *leftderivations1* :: (*sentence* × *sentence*) *set*
where
leftderivations1 = { (*u*,*v*) | *u v*. *leftderives1 u v* }

lemma [*simp*]: *leftderivations1* ⊆ *derivations1*
⟨*proof*⟩

definition *leftderivations* :: (*sentence* × *sentence*) *set*
where
leftderivations = *leftderivations1*^*

lemma *rtrancl-subset-implies*: *a* ⊆ *b* ⟹ *a* ⊆ *b*^* ⟨*proof*⟩

lemma *leftderivations-subset-derivations*[*simp*]: *leftderivations* ⊆ *derivations*
⟨*proof*⟩

definition *leftderives* :: *sentence* ⇒ *sentence* ⇒ *bool*
where
leftderives u v = ((*u*, *v*) ∈ *leftderivations*)

lemma *leftderives-implies-derives*[*simp*]: *leftderives u v* ⟹ *derives u v*
⟨*proof*⟩

definition *is-leftderivation* :: *sentence* ⇒ *bool*
where
is-leftderivation u = *leftderives* [𝔖] *u*

lemma *leftderivation-implies-derivation*[*simp*]:
is-leftderivation u ⟹ *is-derivation u*
⟨*proof*⟩

lemma *leftderives-refl*[*simp*]: *leftderives u u*
⟨*proof*⟩

lemma *leftderives1-implies-leftderives*[*simp*]:*leftderives1 a b* ⟹ *leftderives a b*
⟨*proof*⟩

lemma *leftderives-trans*: *leftderives a b* ⟹ *leftderives b c* ⟹ *leftderives a c*

⟨*proof*⟩

lemma *leftderives1-eq-leftderivations1*: *leftderives1 x y* = ((*x*, *y*) ∈ *leftderivations1*)
⟨*proof*⟩

lemma *leftderives-induct*[*consumes 1*, *case-names Base Step*]:
assumes *derives*: *leftderives a b*
assumes *Pa*: *P a*
assumes *induct*: $\bigwedge$*y z*. *leftderives a y* ⟹ *leftderives1 y z* ⟹ *P y* ⟹ *P z*
shows *P b*
⟨*proof*⟩

end

context *CFG* **begin**

lemma *derives1-implies-derives*[*simp*]:*derives1 a b* ⟹ *derives a b*
⟨*proof*⟩

lemma *derives-trans*: *derives a b* ⟹ *derives b c* ⟹ *derives a c*
⟨*proof*⟩

lemma *derives1-eq-derivations1*: *derives1 x y* = ((*x*, *y*) ∈ *derivations1*)
⟨*proof*⟩

lemma *derives-induct*[*consumes 1*, *case-names Base Step*]:
assumes *derives*: *derives a b*
assumes *Pa*: *P a*
assumes *induct*: $\bigwedge$*y z*. *derives a y* ⟹ *derives1 y z* ⟹ *P y* ⟹ *P z*
shows *P b*
⟨*proof*⟩

end

context *CFG* **begin**

definition *Derives1* :: *sentence* ⇒ *nat* ⇒ *rule* ⇒ *sentence* ⇒ *bool*
where
Derives1 u i r v =
(∃ *x y N α*.
u = *x* @ [*N*] @ *y*
∧ *v* = *x* @ *α* @ *y*
∧ *is-sentence x*
∧ *is-sentence y*
∧ (*N*, *α*) ∈ $\mathfrak{R}$
∧ *r* = (*N*, *α*) ∧ *i* = *length x*)

lemma *Derives1-split*:
Derives1 u i r v $\Longrightarrow \exists$ *x y*. $u = x$ @ [*fst r*] @ $y \wedge v = x$ @ (*snd r*) @ $y \wedge$ *length* $x = i$
⟨*proof*⟩

lemma *Derives1-implies-derives1*: *Derives1 u i r v* $\Longrightarrow$ *derives1 u v*
⟨*proof*⟩

lemma *derives1-implies-Derives1*: *derives1 u v* $\Longrightarrow \exists$ *i r*. *Derives1 u i r v*
⟨*proof*⟩

lemma *Derives1-unique-dest*: *Derives1 u i r v* $\Longrightarrow$ *Derives1 u i r w* $\Longrightarrow v = w$
⟨*proof*⟩

lemma *Derives1-unique-src*: *Derives1 u i r w* $\Longrightarrow$ *Derives1 v i r w* $\Longrightarrow u = v$
⟨*proof*⟩

type-synonym *derivation* = (*nat* × *rule*) *list*

fun *Derivation* :: *sentence* ⇒ *derivation* ⇒ *sentence* ⇒ *bool*
where
Derivation a [] *b* = (*a* = *b*)
| *Derivation a* (*d*#*D*) *b* = ($\exists$ *x*. *Derives1 a* (*fst d*) (*snd d*) *x* $\wedge$ *Derivation x D b*)

lemma *Derivation-implies-derives*: *Derivation a D b* $\Longrightarrow$ *derives a b*
⟨*proof*⟩

lemma *Derivation-Derives1*: *Derivation a S y* $\Longrightarrow$ *Derives1 y i r z* $\Longrightarrow$ *Derivation a* (*S*@[(*i*,*r*)]) *z*
⟨*proof*⟩

lemma *derives-implies-Derivation*: *derives a b* $\Longrightarrow \exists$ *D*. *Derivation a D b*
⟨*proof*⟩

lemma *Derives1-take*: *Derives1 a i r b* $\Longrightarrow$ *take i a* = *take i b*
⟨*proof*⟩

lemma *Derives1-drop*: *Derives1 a i r b* $\Longrightarrow$ *drop* (*Suc i*) *a* = *drop* (*i* + *length* (*snd r*)) *b*
⟨*proof*⟩

lemma *Derives1-bound*: *Derives1 a i r b* $\Longrightarrow i <$ *length a*
⟨*proof*⟩

lemma *Derives1-length*: *Derives1 a i r b* $\Longrightarrow$ *length b* = *length a* + *length* (*snd r*) − *1*
⟨*proof*⟩

definition *leftmost* :: *nat* ⇒ *sentence* ⇒ *bool*
where
leftmost i s = (*i* < *length s* ∧ *is-word* (*take i s*) ∧ *is-nonterminal* (*s* ! *i*))

lemma *set-take*: *set* (*take n s*) = { *s* ! *i* | *i*. *i* < *n* ∧ *i* < *length s*}
⟨*proof*⟩

lemma *list-all-take*: *list-all P* (*take n s*) = (∀ *i*. *i* < *n* ∧ *i* < *length s* ⟶ *P* (*s* ! *i*))
⟨*proof*⟩

lemma *is-sentence-concat*: *is-sentence* (*x*@*y*) = (*is-sentence x* ∧ *is-sentence y*)
⟨*proof*⟩

lemma *is-sentence-cons*: *is-sentence* (*x*#*xs*) = (*is-symbol x* ∧ *is-sentence xs*)
⟨*proof*⟩

lemma *rule-nonterminal-type*[*simp*]: (*N*, α) ∈ $\mathfrak{R}$ ⟹ *is-nonterminal N*
⟨*proof*⟩

lemma *rule-*α*-type*[*simp*]: (*N*, α) ∈ $\mathfrak{R}$ ⟹ *is-sentence* α
⟨*proof*⟩

lemma [*simp*]: *is-nonterminal N* ⟹ *is-symbol N*
⟨*proof*⟩

lemma *Derives1-sentence1*[*elim*]: *Derives1 a i r b* ⟹ *is-sentence a*
⟨*proof*⟩

lemma *Derives1-sentence2*[*elim*]: *Derives1 a i r b* ⟹ *is-sentence b*
⟨*proof*⟩

lemma [*elim*]: *Derives1 a i r b* ⟹ *r* ∈ $\mathfrak{R}$
⟨*proof*⟩

lemma *is-sentence-symbol*: *is-sentence a* ⟹ *i* < *length a* ⟹ *is-symbol* (*a* ! *i*)
⟨*proof*⟩

lemma *is-symbol-distinct*: *is-symbol x* ⟹ *is-terminal x* ≠ *is-nonterminal x*
⟨*proof*⟩

lemma *is-terminal-nonterminal*: *is-terminal x* ⟹ *is-nonterminal x* ⟹ *False*
⟨*proof*⟩

lemma *Derives1-leftmost*:
assumes *Derives1 a i r b*
shows ∃ *j*. *leftmost j a* ∧ *j* ≤ *i*
⟨*proof*⟩

lemma *Derivation-leftmost*: $D \neq [] \Longrightarrow$ *Derivation a D b* $\Longrightarrow \exists\ i.$ *leftmost i a*
⟨*proof*⟩

lemma *nonword-has-nonterminal*:
is-sentence a $\Longrightarrow\ \neg$ (*is-word a*) $\Longrightarrow \exists\ k.\ k <$ *length a* $\wedge$ *is-nonterminal* (*a* ! *k*)
⟨*proof*⟩

lemma *leftmost-cons-nonterminal*:
is-nonterminal x $\Longrightarrow$ *leftmost 0* (*x#xs*)
⟨*proof*⟩

lemma *leftmost-cons-terminal*:
is-terminal x $\Longrightarrow$ *leftmost i* (*x#xs*) = ($i > 0 \wedge$ *leftmost* ($i - 1$) *xs*)
⟨*proof*⟩

lemma *is-nonterminal-cons-terminal*:
is-terminal x $\Longrightarrow k <$ *length* (*x* # *a*) $\Longrightarrow$ *is-nonterminal* ((*x* # *a*) ! *k*) $\Longrightarrow$
$k > 0 \wedge k - 1 <$ *length a* $\wedge$ *is-nonterminal* (*a* ! ($k - 1$))
⟨*proof*⟩

lemma *leftmost-exists*:
is-sentence a $\Longrightarrow k <$ *length a* $\Longrightarrow$ *is-nonterminal* (*a* ! *k*) $\Longrightarrow$
$\exists\ i.$ *leftmost i a* $\wedge\ i \leq k$
⟨*proof*⟩

lemma *nonword-leftmost-exists*:
is-sentence a $\Longrightarrow \neg$ (*is-word a*) $\Longrightarrow \exists\ i.$ *leftmost i a*
⟨*proof*⟩

lemma *leftmost-unaffected-Derives1*: *leftmost j a* $\Longrightarrow j < i \Longrightarrow$ *Derives1 a i r b*
$\Longrightarrow$ *leftmost j b*
⟨*proof*⟩

definition *derivation-ge* :: *derivation* $\Rightarrow$ *nat* $\Rightarrow$ *bool*
where
derivation-ge D i = ($\forall\ d \in$ *set D*. *fst d* $\geq i$)

lemma *derivation-ge-cons*: *derivation-ge* (*d#D*) *i* = (*fst d* $\geq i \wedge$ *derivation-ge D*
i)
⟨*proof*⟩

lemma *derivation-ge-append*:
derivation-ge (*D@E*) *i* = (*derivation-ge D i* $\wedge$ *derivation-ge E i*)
⟨*proof*⟩

lemma *leftmost-unaffected-Derivation*:
derivation-ge D (*Suc i*) $\Longrightarrow$ *leftmost i a* $\Longrightarrow$ *Derivation a D b* $\Longrightarrow$ *leftmost i b*
⟨*proof*⟩

lemma *le-Derives1-take*:
assumes *le*: $i \leq j$
and *D*: *Derives1 a j r b*
shows *take i a* = *take i b*
⟨*proof*⟩

lemma *Derivation-take*: *derivation-ge D i* ⟹ *Derivation a D b* ⟹ *take i a* = *take i b*
⟨*proof*⟩

lemma *leftmost-cons-less*: $i <$ *length u* ⟹ *leftmost i* (*u*@*v*) = *leftmost i u*
⟨*proof*⟩

lemma *leftmost-is-nonterminal*: *leftmost i u* ⟹ *is-nonterminal* (*u* ! *i*)
⟨*proof*⟩

lemma *is-word-is-terminal*: $i <$ *length u* ⟹ *is-word u* ⟹ *is-terminal* (*u* ! *i*)
⟨*proof*⟩

lemma *leftmost-append*:
assumes *leftmost*: *leftmost i* (*u*@*v*)
and *is-word*: *is-word u*
shows *length u* $\leq i$
⟨*proof*⟩

lemma *derivation-ge-empty*[*simp*]: *derivation-ge* [] *i*
⟨*proof*⟩

lemma *leftmost-notword*: *leftmost i a* ⟹ $j > i$ ⟹ ¬ (*is-word* (*take j a*))
⟨*proof*⟩

lemma *leftmost-unique*: *leftmost i a* ⟹ *leftmost j a* ⟹ $i = j$
⟨*proof*⟩

lemma *leftmost-Derives1*: *leftmost i a* ⟹ *Derives1 a j r b* ⟹ $i \leq j$
⟨*proof*⟩

lemma *leftmost-Derives1-propagate*:
assumes *leftmost*: *leftmost i a*
and *Derives1*: *Derives1 a j r b*
shows (*is-word b* ∧ $i = j$) ∨ (∃ *k*. *leftmost k b* ∧ $i \leq k$)
⟨*proof*⟩

lemma *is-word-Derives1*[*elim*]: *is-word a* ⟹ *Derives1 a i r b* ⟹ *False*
⟨*proof*⟩

lemma *is-word-Derivation*[*elim*]: *is-word a* ⟹ *Derivation a D b* ⟹ *D* = []
⟨*proof*⟩

lemma *leftmost-Derivation*:
leftmost i a $\Longrightarrow$ *Derivation a D b* $\Longrightarrow$ $j \leq i$ $\Longrightarrow$ *derivation-ge D j*
⟨*proof*⟩

lemma *derivation-ge-list-all*: *derivation-ge D i* = *list-all* (λ *d*. *fst d* ≥ *i*) *D*
⟨*proof*⟩

lemma *split-derivation-leftmost*:
assumes *derivation-ge D i*
and ¬ (*derivation-ge D* (*Suc i*))
shows ∃ *E F r*. *D* = *E*@[(*i*, *r*)]@*F* ∧ *derivation-ge E* (*Suc i*)
⟨*proof*⟩

lemma *Derives1-Derives1-swap*:
assumes $i < j$
and *Derives1 a j p b*
and *Derives1 b i q c*
shows ∃ *b′*. *Derives1 a i q b′* ∧ *Derives1 b′* (*j* − *1* + *length* (*snd q*)) *p c*
⟨*proof*⟩

definition *derivation-shift* :: *derivation* ⇒ *nat* ⇒ *nat* ⇒ *derivation*
where
derivation-shift D left right = *map* (λ *d*. (*fst d* − *left* + *right*, *snd d*)) *D*

lemma *derivation-shift-empty*[*simp*]: *derivation-shift* [] *left right* = []
⟨*proof*⟩

lemma *derivation-shift-cons*[*simp*]:
derivation-shift (*d*#*D*) *left right* = ((*fst d* − *left* + *right*, *snd d*)#(*derivation-shift D left right*))
⟨*proof*⟩

lemma *Derivation-append*: *Derivation a* (*D*@*E*) *c* = (∃ *b*. *Derivation a D b* ∧ *Derivation b E c*)
⟨*proof*⟩

lemma *Derivation-implies-append*:
Derivation a D b $\Longrightarrow$ *Derivation b E c* $\Longrightarrow$ *Derivation a* (*D*@*E*) *c*
⟨*proof*⟩

lemma *Derivation-swap-single-end-to-front*:
$i < j$ $\Longrightarrow$ *derivation-ge D j* $\Longrightarrow$ *Derivation a* (*D*@[(*i*,*r*)]) *b* $\Longrightarrow$
Derivation a ((*i*,*r*)#(*derivation-shift D 1* (*length* (*snd r*)))) *b*
⟨*proof*⟩

lemma *Derivation-swap-single-mid-to-front*:
assumes $i < j$
and *derivation-ge D j*
and *Derivation a* (*D*@[(*i*,*r*)]@*E*) *b*

shows *Derivation a ((i,r)#((derivation-shift D 1 (length (snd r)))@E)) b*
⟨*proof*⟩

lemma *length-derivation-shift*[*simp*]:
length(*derivation-shift D left right*) = *length D*
⟨*proof*⟩

definition *LeftDerives1* :: *sentence* ⇒ *nat* ⇒ *rule* ⇒ *sentence* ⇒ *bool*
where
LeftDerives1 u i r v = (*leftmost i u* ∧ *Derives1 u i r v*)

lemma *LeftDerives1-implies-leftderives1*: *LeftDerives1 u i r v* ⟹ *leftderives1 u v*
⟨*proof*⟩

lemma *leftmost-Derives1-leftderives*:
leftmost i a ⟹ *Derives1 a i r b* ⟹ *leftderives b c* ⟹ *leftderives a c*
⟨*proof*⟩

theorem *Derivation-implies-leftderives-gen*:
Derivation a D (*u*@*v*) ⟹ *is-word u* ⟹ (∃ *w*.
leftderives a (*u*@*w*) ∧
(*v* = [] ⟶ *w* = []) ∧
(∀ *X*. *is-first X v* ⟶ *is-first X w*))
⟨*proof*⟩

lemma *derives-implies-leftderives-gen*: *derives a* (*u*@*v*) ⟹ *is-word u* ⟹ (∃ *w*.
leftderives a (*u*@*w*) ∧
(*v* = [] ⟶ *w* = []) ∧
(∀ *X*. *is-first X v* ⟶ *is-first X w*))
⟨*proof*⟩

lemma *derives-implies-leftderives*: *derives a b* ⟹ *is-word b* ⟹ *leftderives a b*
⟨*proof*⟩

fun *LeftDerivation* :: *sentence* ⇒ *derivation* ⇒ *sentence* ⇒ *bool*
where
LeftDerivation a [] *b* = (*a* = *b*)
| *LeftDerivation a* (*d*#*D*) *b* = (∃ *x*. *LeftDerives1 a* (*fst d*) (*snd d*) *x* ∧ *LeftDerivation x D b*)

lemma *LeftDerives1-implies-Derives1*: *LeftDerives1 a i r b* ⟹ *Derives1 a i r b*
⟨*proof*⟩

lemma *LeftDerivation-implies-Derivation*:
LeftDerivation a D b ⟹ *Derivation a D b*
⟨*proof*⟩

lemma *LeftDerivation-implies-leftderives*: *LeftDerivation a D b* ⟹ *leftderives a b*
⟨*proof*⟩

lemma *leftmost-witness*[*simp*]: *leftmost* (*length* *x*) (*x*@(*N*#*y*)) = (*is-word* *x* ∧ *is-nonterminal* *N*)
⟨*proof*⟩

lemma *leftderives1-implies-LeftDerives1*:
assumes *leftderives1*: *leftderives1* *u* *v*
shows ∃ *i* *r*. *LeftDerives1* *u* *i* *r* *v*
⟨*proof*⟩

lemma *LeftDerivation-LeftDerives1*:
LeftDerivation *a* *S* *y* ⟹ *LeftDerives1* *y* *i* *r* *z* ⟹ *LeftDerivation* *a* (*S*@[(*i*,*r*)]) *z*
⟨*proof*⟩

lemma *leftderives-implies-LeftDerivation*: *leftderives* *a* *b* ⟹ ∃ *D*. *LeftDerivation* *a* *D* *b*
⟨*proof*⟩

lemma *LeftDerivation-append*:
LeftDerivation *a* (*D*@*E*) *c* = (∃ *b*. *LeftDerivation* *a* *D* *b* ∧ *LeftDerivation* *b* *E* *c*)
⟨*proof*⟩

lemma *LeftDerivation-implies-append*:
LeftDerivation *a* *D* *b* ⟹ *LeftDerivation* *b* *E* *c* ⟹ *LeftDerivation* *a* (*D*@*E*) *c*
⟨*proof*⟩

lemma *Derivation-unique-dest*: *Derivation* *a* *D* *b* ⟹ *Derivation* *a* *D* *c* ⟹ *b* = *c*
⟨*proof*⟩

lemma *Derivation-unique-src*: *Derivation* *a* *D* *c* ⟹ *Derivation* *b* *D* *c* ⟹ *a* = *b*
⟨*proof*⟩

lemma *LeftDerives1-unique*: *LeftDerives1* *a* *i* *r* *b* ⟹ *LeftDerives1* *a* *j* *s* *b* ⟹ *i* = *j* ∧ *r* = *s*
⟨*proof*⟩

lemma *leftlang*: $\mathcal{L}$ = { *v* | *v*. *is-word* *v* ∧ *is-leftderivation* *v* }
⟨*proof*⟩

lemma *leftprefixlang*: $\mathcal{L}_P$ = { *u* | *u* *v*. *is-word* *u* ∧ *is-leftderivation* (*u*@*v*) }
⟨*proof*⟩

lemma *derives-implies-leftderives-cons*:
is-word *a* ⟹ *derives* *u* (*a*@*X*#*b*) ⟹ ∃ *c*. *leftderives* *u* (*a*@*X*#*c*)
⟨*proof*⟩

lemma *is-word-append*[*simp*]: *is-word* (*a*@*b*) = (*is-word* *a* ∧ *is-word* *b*)
⟨*proof*⟩

lemma $\mathcal{L}_P$*-split*: $a@b \in \mathcal{L}_P \Longrightarrow a \in \mathcal{L}_P$
⟨*proof*⟩

lemma $\mathcal{L}_P$*-is-word*: $a \in \mathcal{L}_P \Longrightarrow$ *is-word a*
⟨*proof*⟩

definition *Derive* :: *sentence* ⇒ *derivation* ⇒ *sentence*
where
Derive a D = (*THE b*. *Derivation a D b*)

lemma *Derivation-dest-ex-unique*: *Derivation a D b* ⟹ ∃! *x*. *Derivation a D x*
⟨*proof*⟩

lemma *Derive*:
assumes *ab*: *Derivation a D b*
shows *Derive a D* = *b*
⟨*proof*⟩

end

end
theory *Validity*
imports *LLEarleyParsing Derivations*
begin

context *LocalLexing* **begin**

definition *wellformed-token* :: *token* ⇒ *bool*
where
wellformed-token t = *is-terminal* (*terminal-of-token t*)

definition *wellformed-tokens* :: *tokens* ⇒ *bool*
where
wellformed-tokens ts = *list-all wellformed-token ts*

definition *doc-tokens* :: *tokens* ⇒ *bool*
where
doc-tokens p = (*wellformed-tokens p* ∧ *is-prefix* (*chars p*) *Doc*)

definition *wellformed-item* :: *item* ⇒ *bool*
where
wellformed-item x = (
item-rule x ∈ $\mathfrak{R}$ ∧
item-origin x ≤ *item-end x* ∧
item-end x ≤ *length Doc* ∧
item-dot x ≤ *length* (*item-rhs x*))

definition *wellformed-items* :: *items* ⇒ *bool*
where

wellformed-items X = (∀ *x* ∈ *X*. *wellformed-item x*)

lemma *is-word-terminals*: *wellformed-tokens p* ⟹ *is-word* (*terminals p*)
⟨*proof*⟩

lemma *is-word-subset*: *is-word x* ⟹ *set y* ⊆ *set x* ⟹ *is-word y*
⟨*proof*⟩

lemma *is-word-terminals-take*: *wellformed-tokens p* ⟹ *is-word*(*terminals* (*take n p*))
⟨*proof*⟩

lemma *is-word-terminals-drop*: *wellformed-tokens p* ⟹ *is-word*(*terminals* (*drop n p*))
⟨*proof*⟩

definition *pvalid* :: *tokens* ⇒ *item* ⇒ *bool*
where
pvalid p x = (∃ *u* γ.
wellformed-tokens p ∧
wellformed-item x ∧
u ≤ *length p* ∧
charslength p = *item-end x* ∧
charslength (*take u p*) = *item-origin x* ∧
is-derivation (*terminals* (*take u p*) @ [*item-nonterminal x*] @ γ) ∧
derives (*item-α x*) (*terminals* (*drop u p*)))

definition *Gen* :: *tokens set* ⇒ *items*
where
Gen P = { *x* | *x p*. *p* ∈ *P* ∧ *pvalid p x* }

lemma *wellformed-items* (*Gen P*)
⟨*proof*⟩

lemma *wellformed-items* (*Init*)
⟨*proof*⟩

definition *pvalid-left* :: *tokens* ⇒ *item* ⇒ *bool*
where
pvalid-left p x = (∃ *u* γ.
wellformed-tokens p ∧
wellformed-item x ∧
u ≤ *length p* ∧
charslength p = *item-end x* ∧
charslength (*take u p*) = *item-origin x* ∧
is-leftderivation (*terminals* (*take u p*) @ [*item-nonterminal x*] @ γ) ∧
leftderives (*item-α x*) (*terminals* (*drop u p*)))

lemma *pvalid-left*: *pvalid p x* = *pvalid-left p x*

⟨*proof*⟩

lemma $\mathcal{L}_P$*-wellformed-tokens*: *terminals p* $\in \mathcal{L}_P \Longrightarrow$ *wellformed-tokens p*
⟨*proof*⟩

end

end
theory *TheoremD2*
imports *LocalLexingLemmas Validity Derivations*
begin

context *LocalLexing* **begin**

definition *splits-at* :: *sentence* ⇒ *nat* ⇒ *sentence* ⇒ *symbol* ⇒ *sentence* ⇒ *bool*
where
splits-at δ *i* α *N* β = (*i* < *length* δ ∧ α = *take i* δ ∧ *N* = δ ! *i* ∧ β = *drop* (*Suc i*) δ)

lemma *splits-at-combine*: *splits-at* δ *i* α *N* $\beta \Longrightarrow \delta = \alpha$ @ [*N*] @ β
⟨*proof*⟩

lemma *splits-at-combine-dest*: *Derives1 a i r b* ⟹ *splits-at a i* α *N* β ⟹ *b* = α @ (*snd r*) @ β
⟨*proof*⟩

lemma *Derives1-nonterminal*:
assumes *Derives1 a i r b*
assumes *splits-at a i* α *N* β
shows *fst r* = *N* ∧ *is-nonterminal N*
⟨*proof*⟩

lemma *splits-at-ex*: *Derives1* δ *i r s* ⟹ ∃ α *N* β. *splits-at* δ *i* α *N* β
⟨*proof*⟩

lemma *splits-at-*α: *Derives1* δ *i r s* ⟹ *splits-at* δ *i* α *N* β ⟹
α = *take i* δ ∧ α = *take i s* ∧ *length* α = *i*
⟨*proof*⟩

lemma *LeftDerives1-splits-at-is-word*: *LeftDerives1* δ *i r s* ⟹ *splits-at* δ *i* α *N* β ⟹ *is-word* α
⟨*proof*⟩

lemma *splits-at-*β: *Derives1* δ *i r s* ⟹ *splits-at* δ *i* α *N* β ⟹
β = *drop* (*Suc i*) δ ∧ β = *drop* (*i* + *length* (*snd r*)) *s* ∧ *length* β = *length* δ − *i* − *1*
⟨*proof*⟩

lemma *Derives1-prefix*:
assumes *ab*: $Derives1\ \delta\ i\ r\ (a@b)$
assumes *split*: $splits\text{-}at\ \delta\ i\ \alpha\ N\ \beta$
shows $is\text{-}prefix\ \alpha\ a \lor is\text{-}prefix\ a\ \alpha$
⟨*proof*⟩

lemma *Derives1-suffix*:
assumes *ab*: $Derives1\ \delta\ i\ r\ (a@b)$
assumes *split*: $splits\text{-}at\ \delta\ i\ \alpha\ N\ \beta$
shows $is\text{-}suffix\ \beta\ b \lor is\text{-}suffix\ b\ \beta$
⟨*proof*⟩

lemma *Derives1-skip-prefix*:
$length\ a \leq i \Longrightarrow Derives1\ (a@b)\ i\ r\ (a@c) \Longrightarrow Derives1\ b\ (i - length\ a)\ r\ c$
⟨*proof*⟩

lemma *cancel-suffix*:
assumes $a\ @\ c = b\ @\ d$
assumes $length\ c \leq length\ d$
shows $a = b\ @\ (take\ (length\ d - length\ c)\ d)$
⟨*proof*⟩

lemma *is-sentence-take*:
$is\text{-}sentence\ y \Longrightarrow is\text{-}sentence\ (take\ n\ y)$
⟨*proof*⟩

lemma *Derives1-skip-suffix*:
assumes *i*: $i < length\ a$
assumes *D*: $Derives1\ (a@c)\ i\ r\ (b@c)$
shows $Derives1\ a\ i\ r\ b$
⟨*proof*⟩

lemma *drop-cancel-suffix*: $a@c = drop\ n\ (b@c) \Longrightarrow a = drop\ n\ b$
⟨*proof*⟩

lemma *drop-keep-last*: $u \neq [] \Longrightarrow u = drop\ n\ (a@[X]) \Longrightarrow u = drop\ n\ a\ @\ [X]$
⟨*proof*⟩

lemma *Derives1-X-is-part-of-rule*[*consumes 2, case-names Suffix Prefix*]:
assumes *aXb*: $Derives1\ \delta\ i\ r\ (a@[X]@b)$
assumes *split*: $splits\text{-}at\ \delta\ i\ \alpha\ N\ \beta$
assumes *prefix*: $\bigwedge \beta.\ \delta = a\ @\ [X]\ @\ \beta \Longrightarrow length\ a < i \Longrightarrow$
$Derives1\ \beta\ (i - length\ a - 1)\ r\ b \Longrightarrow False$
assumes *suffix*: $\bigwedge \alpha.\ \delta = \alpha\ @\ [X]\ @\ b \Longrightarrow Derives1\ \alpha\ i\ r\ a \Longrightarrow False$
shows $\exists\ u\ v.\ a = \alpha\ @\ u \land b = v\ @\ \beta \land (snd\ r) = u@[X]@v$
⟨*proof*⟩

lemma $\mathcal{L}_P$*-derives*: $a \in \mathcal{L}_P \Longrightarrow \exists\ b.\ derives\ [\mathfrak{S}]\ (a@b)$
⟨*proof*⟩

lemma $\mathcal{L}_P$*-leftderives*: $a \in \mathcal{L}_P \Longrightarrow \exists\ b.$ *leftderives* $[\mathfrak{S}]$ $(a@b)$
⟨*proof*⟩

lemma *Derives1-rule*: *Derives1* $a\ i\ r\ b \Longrightarrow r \in \mathfrak{R}$
⟨*proof*⟩

lemma *is-prefix-empty*[*simp*]: *is-prefix* [] a
⟨*proof*⟩

lemma *is-prefix-cons*: *is-prefix* $(x \# a)$ $b = (\exists\ c.\ b = x \# c \wedge$ *is-prefix* $a\ c)$
⟨*proof*⟩

lemma *is-prefix-cancel*[*simp*]: *is-prefix* $(a@b)$ $(a@c) =$ *is-prefix* $b\ c$
⟨*proof*⟩

lemma *is-prefix-chars*: *is-prefix* $a\ b \Longrightarrow$ *is-prefix* (*chars* a) (*chars* b)
⟨*proof*⟩

lemma *is-prefix-length*: *is-prefix* $a\ b \Longrightarrow$ *length* $a \leq$ *length* b
⟨*proof*⟩

lemma *is-prefix-take*[*simp*]: *is-prefix* (*take* n a) a
⟨*proof*⟩

lemma *doc-tokens-length*: *doc-tokens* $p \Longrightarrow$ *length* (*chars* p) $\leq$ *length Doc*
⟨*proof*⟩

fun *count-terminals* :: *sentence* ⇒ *nat* **where**
count-terminals [] = *0*
| *count-terminals* (*x*#*xs*) = (*if* (*is-terminal x*) *then Suc* (*count-terminals xs*) *else* (*count-terminals xs*))

lemma *count-terminals-upper-bound*: *count-terminals* $p \leq$ *length* p
⟨*proof*⟩

lemma *count-terminals-append*[*simp*]: *count-terminals* $(a@b)$ = *count-terminals* a + *count-terminals* b
⟨*proof*⟩

lemma *Derives1-count-terminals*:
assumes *D*: *Derives1* $a\ i\ r\ b$
shows *count-terminals* b = *count-terminals* a + *count-terminals* (*snd* r)
⟨*proof*⟩

lemma *Derives1-count-terminals-leq*:
assumes *D*: *Derives1* $a\ i\ r\ b$
shows *count-terminals* $a \leq$ *count-terminals* b
⟨*proof*⟩

lemma *Derivation-count-terminals-leq*:
Derivation a E b $\Longrightarrow$ *count-terminals a* $\leq$ *count-terminals b*
⟨*proof*⟩

lemma *derives-count-terminals-leq*: *derives a b* $\Longrightarrow$ *count-terminals a* $\leq$ *count-terminals b*
⟨*proof*⟩

lemma *is-word-cons*[*simp*]: *is-word* (*x*#*xs*) = (*is-terminal x* $\wedge$ *is-word xs*)
⟨*proof*⟩

lemma *count-terminals-of-word*: *is-word w* $\Longrightarrow$ *count-terminals w* = *length w*
⟨*proof*⟩

lemma *length-terminals*[*simp*]: *length* (*terminals p*) = *length p*
⟨*proof*⟩

lemma *path-length-is-upper-bound*:
assumes *p*: *wellformed-tokens p*
assumes α: *is-word* α
assumes *derives*: *derives* (α@*u*) (*terminals p*)
shows *length* α $\leq$ *length p*
⟨*proof*⟩

lemma *is-word-Derives1-index*:
assumes *w*: *is-word w*
assumes *derives1*: *Derives1* (*w*@*a*) *i r b*
shows *i* $\geq$ *length w*
⟨*proof*⟩

lemma *is-word-Derivation-derivation-ge*:
assumes *w*: *is-word w*
assumes *D*: *Derivation* (*w*@*a*) *D b*
shows *derivation-ge D* (*length w*)
⟨*proof*⟩

lemma *derives-word-is-prefix*:
assumes *w*: *is-word w*
assumes *derives*: *derives* (*w*@*a*) *b*
shows *is-prefix w b*
⟨*proof*⟩

lemma *terminals-take*[*simp*]: *terminals* (*take n p*) = *take n* (*terminals p*)
⟨*proof*⟩

lemma *terminals-drop*[*simp*]: *terminals* (*drop n p*) = *drop n* (*terminals p*)
⟨*proof*⟩

lemma *take-prefix*[*simp*]: *is-prefix a b* $\Longrightarrow$ *take* (*length a*) *b* = *a*
⟨*proof*⟩

lemma *Derives1-drop-prefixword*:
assumes *w*: *is-word w*
assumes *wa-b*: *Derives1* (*w*@*a*) *i r b*
shows *Derives1 a* (*i* − *length w*) *r* (*drop* (*length w*) *b*)
⟨*proof*⟩

lemma *derives1-drop-prefixword*:
assumes *w*: *is-word w*
assumes *wa-b*: *derives1* (*w*@*a*) *b*
shows *derives1 a* (*drop* (*length w*) *b*)
⟨*proof*⟩

lemma *derives1-is-word-is-prefix-drop*:
assumes *w*: *is-word w*
assumes *w-a*: *is-prefix w a*
assumes *ab*: *derives1 a b*
shows *derives1* (*drop* (*length w*) *a*) (*drop* (*length w*) *b*)
⟨*proof*⟩

lemma *derives-drop-prefixword-helper*:
derives a b $\Longrightarrow$ *is-word w* $\Longrightarrow$ *is-prefix w a* $\Longrightarrow$ *derives* (*drop* (*length w*) *a*) (*drop* (*length w*) *b*)
⟨*proof*⟩

lemma *derive-drop-prefixword*:
is-word w $\Longrightarrow$ *derives* (*w*@*a*) *b* $\Longrightarrow$ *derives a* (*drop* (*length w*) *b*)
⟨*proof*⟩

lemma *thmD2′*:
assumes *X*: *is-terminal X*
assumes *p*: *doc-tokens p*
assumes *pX*: (*terminals p*)@[*X*] $\in \mathcal{L}_P$
shows $\exists$ *x*. *pvalid p x* $\wedge$ *next-symbol x* = *Some X*
⟨*proof*⟩

lemma *admissible-wellformed-tokens*: *admissible p* $\Longrightarrow$ *wellformed-tokens p*
⟨*proof*⟩

lemma *chars-append*[*simp*]: *chars* (*a*@*b*) = (*chars a*)@(*chars b*)
⟨*proof*⟩

lemma *chars-of-token-simp*[*simp*]: *chars-of-token* (*a*, *b*) = *b*
⟨*proof*⟩

lemma $\mathcal{X}$*-is-prefix*: *t* $\in \mathcal{X}$ *k* $\Longrightarrow$ *is-prefix* (*snd t*) (*drop k Doc*)
⟨*proof*⟩

lemma *is-prefix-append*: *is-prefix* $(a@b)$ D = (*is-prefix* a D ∧ *is-prefix* b (*drop* (*length* a) D))
⟨*proof*⟩

lemma $\mathfrak{P}$-*are-doc-tokens*: $p \in \mathfrak{P} \Longrightarrow$ *doc-tokens* p
⟨*proof*⟩

theorem *thmD2*:
assumes X: *is-terminal* X
assumes p: $p \in \mathfrak{P}$
assumes pX: (*terminals* p)@$[X] \in \mathcal{L}_P$
shows ∃ x. *pvalid* p x ∧ *next-symbol* x = *Some* X
⟨*proof*⟩

end

end
theory *TheoremD4*
imports *TheoremD2*
begin

context *LocalLexing* **begin**

lemma $\mathcal{X}$-*are-terminals*: $u \in \mathcal{X}\ k \Longrightarrow$ *is-terminal* (*terminal-of-token* u)
⟨*proof*⟩

lemma *terminals-append*[*simp*]: *terminals* $(a@b)$ = ((*terminals* a) @ (*terminals* b))
⟨*proof*⟩

lemma *terminals-singleton*[*simp*]: *terminals* $[u]$ = [*terminal-of-token* u]
⟨*proof*⟩

lemma *terminal-of-token-simp*[*simp*]: *terminal-of-token* $(a, b) = a$
⟨*proof*⟩

lemma *pvalid-item-end*: *pvalid* p $x \Longrightarrow$ *item-end* x = *charslength* p
⟨*proof*⟩

lemma $\mathcal{W}$-*elem-in-TokensAt*:
assumes P: $P \subseteq \mathfrak{P}$
assumes *u-in-*$\mathcal{W}$: $u \in \mathcal{W}\ P\ k$
shows $u \in$ *TokensAt* k (*Gen* P)
⟨*proof*⟩

lemma *is-derivation-is-sentence*: *is-derivation* $s \Longrightarrow$ *is-sentence* s
⟨*proof*⟩

lemma *is-sentence-cons*: *is-sentence* $(N\#s) = (\textit{is-symbol } N \wedge \textit{is-sentence } s)$
⟨*proof*⟩

lemma *is-derivation-step*:
assumes *uNv*: *is-derivation* $(u@[N]@v)$
assumes $N\alpha$: $(N, \alpha) \in \mathfrak{R}$
shows *is-derivation* $(u@\alpha@v)$
⟨*proof*⟩

lemma *is-derivation-derives*:
derives $\alpha\ \beta \Longrightarrow$ *is-derivation* $(u@\alpha@v) \Longrightarrow$ *is-derivation* $(u@\beta@v)$
⟨*proof*⟩

lemma *item-rhs-split*: *item-rhs* $x = (\textit{item-}\alpha\ x)@(\textit{item-}\beta\ x)$
⟨*proof*⟩

lemma *pvalid-is-derivation-terminals-item-*β:
assumes *pvalid*: *pvalid* $p\ x$
shows $\exists\ \delta.$ *is-derivation* $((\textit{terminals } p)@(\textit{item-}\beta\ x)@\delta)$
⟨*proof*⟩

lemma *next-symbol-not-complete*: *next-symbol* $x = \textit{Some } t \Longrightarrow \neg\ (\textit{is-complete } x)$
⟨*proof*⟩

lemma *next-symbol-starts-item-*β:
assumes *wf*: *wellformed-item* x
assumes *next-symbol*: *next-symbol* $x = \textit{Some } t$
shows $\exists\ \delta.$ *item-*$\beta\ x = t\#\delta$
⟨*proof*⟩

lemma *pvalid-prefixlang*:
assumes *pvalid*: *pvalid* $p\ x$
assumes *is-terminal*: *is-terminal* t
assumes *next-symbol*: *next-symbol* $x = \textit{Some } t$
shows $(\textit{terminals } p)\ @\ [t] \in \mathcal{L}_P$
⟨*proof*⟩

lemma *TokensAt-elem-in-*$\mathcal{W}$:
assumes P: $P \subseteq \mathfrak{P}$
assumes *u-in-Tokens-at*: $u \in \textit{TokensAt } k\ (\textit{Gen } P)$
shows $u \in \mathcal{W}\ P\ k$
⟨*proof*⟩

theorem *thmD4*:
assumes P: $P \subseteq \mathfrak{P}$
shows $\mathcal{W}\ P\ k = \textit{TokensAt } k\ (\textit{Gen } P)$
⟨*proof*⟩

end

end
theory *TheoremD5*
imports *TheoremD4*
begin

context *LocalLexing* **begin**

lemma *Scan-empty*: *Scan* {} *k I* = *I*
⟨*proof*⟩

lemma *π-no-tokens*: *π k* {} *I* = *limit* (*λ I*. *Complete k* (*Predict k I*)) *I*
⟨*proof*⟩

lemma *bin-elem*: *x* ∈ *bin I k* ⟹ *x* ∈ *I*
⟨*proof*⟩

lemma *Gen-implies-pvalid*: *x* ∈ *Gen P* ⟹ ∃ *p* ∈ *P*. *pvalid p x*
⟨*proof*⟩

lemma *wellformed-init-item*[*simp*]: *r* ∈ 𝔑 ⟹ *k* ≤ *length Doc* ⟹ *wellformed-item* (*init-item r k*)
⟨*proof*⟩

lemma *init-item-origin*[*simp*]: *item-origin* (*init-item r k*) = *k*
⟨*proof*⟩

lemma *init-item-end*[*simp*]: *item-end* (*init-item r k*) = *k*
⟨*proof*⟩

lemma *init-item-nonterminal*[*simp*]: *item-nonterminal* (*init-item r k*) = *fst r*
⟨*proof*⟩

lemma *init-item-α*[*simp*]: *item-α* (*init-item r k*) = []
⟨*proof*⟩

lemma *Predict-elem-in-Gen*:
assumes *I-in-Gen-P*: *I* ⊆ *Gen P*
assumes *k*: *k* ≤ *length Doc*
assumes *x-in-Predict*: *x* ∈ *Predict k I*
shows *x* ∈ *Gen P*
⟨*proof*⟩

lemma *Predict-subset-Gen*:
assumes *I* ⊆ *Gen P*
assumes *k* ≤ *length Doc*
shows *Predict k I* ⊆ *Gen P*
⟨*proof*⟩

lemma *nth-superfluous-append*[*simp*]: $i < length\ a \Longrightarrow (a@b)!i = a!i$
⟨*proof*⟩

lemma *tokens-nth-in-$\mathcal{Z}$*:
$p \in \mathfrak{P} \Longrightarrow \forall\ i.\ i < length\ p \longrightarrow (\exists\ u.\ p\ !\ i \in \mathcal{Z}\ (charslength\ (take\ i\ p))\ u)$
⟨*proof*⟩

lemma *path-append-token*:
assumes *p*: $p \in \mathcal{P}\ k\ u$
assumes *t*: $t \in \mathcal{Z}\ k\ (Suc\ u)$
assumes *pt*: $admissible\ (p@[t])$
assumes *k*: $charslength\ p = k$
shows $p@[t] \in \mathcal{P}\ k\ (Suc\ u)$
⟨*proof*⟩

definition *indexlt-rel* :: $((nat \times nat) \times (nat \times nat))\ set$ **where**
indexlt-rel = *less-than* <∗*lex*∗> *less-than*

definition *indexlt* :: $nat \Rightarrow nat \Rightarrow nat \Rightarrow nat \Rightarrow bool$ **where**
$indexlt\ k'\ u'\ k\ u = (((k', u'), (k, u)) \in indexlt\text{-}rel)$

lemma *indexlt-simp*: $indexlt\ k'\ u'\ k\ u = (k' < k \lor (k' = k \land u' < u))$
⟨*proof*⟩

lemma *wf-indexlt-rel*: *wf indexlt-rel*
⟨*proof*⟩

lemma *$\mathcal{P}$-induct*[*consumes 1*, *case-names Induct*]:
assumes $p \in \mathcal{P}\ k\ u$
assumes *induct*: $\bigwedge\ p\ k\ u\ .\ (\bigwedge\ p'\ k'\ u'.\ p' \in \mathcal{P}\ k'\ u' \Longrightarrow indexlt\ k'\ u'\ k\ u \Longrightarrow P\ p'\ k'\ u')$
$\Longrightarrow p \in \mathcal{P}\ k\ u \Longrightarrow P\ p\ k\ u$
shows $P\ p\ k\ u$
⟨*proof*⟩

lemma *nonempty-path-indices*:
assumes *p*: $p \in \mathcal{P}\ k\ u$
assumes *nonempty*: $p \neq []$
shows $k > 0 \lor u > 0$
⟨*proof*⟩

lemma *base-paths*:
assumes *p*: $p \in \mathcal{P}\ k\ 0$
assumes *k*: $k > 0$
shows $\exists\ u.\ p \in \mathcal{P}\ (k - 1)\ u$
⟨*proof*⟩

lemma *indexlt-trans*: $indexlt\ k''\ u''\ k'\ u' \Longrightarrow indexlt\ k'\ u'\ k\ u \Longrightarrow indexlt\ k''\ u''\ k\ u$

⟨*proof*⟩

definition *is-continuation* :: *nat* ⇒ *nat* ⇒ *tokens* ⇒ *tokens* ⇒ *bool* **where**
is-continuation k u q ts = ($q \in \mathcal{P}$ *k u* ∧ *charslength q* = *k* ∧ *admissible* (*q*@*ts*) ∧
(∀ *t* ∈ *set ts*. *t* ∈ $\mathcal{Z}$ *k* (*Suc u*)) ∧ (∀ *t* ∈ *set* (*butlast ts*). *chars-of-token t* = []))

lemma *limit-Append-path-nonelem-split*: *p* ∈ *limit* (*Append T k*) ($\mathcal{P}$ *k u*) ⟹ *p* ∉ $\mathcal{P}$ *k u* ⟹
∃ *q ts*. *p* = *q*@*ts* ∧ *q* ∈ $\mathcal{P}$ *k u* ∧ *charslength q* = *k* ∧ *admissible* (*q*@*ts*) ∧ (∀ *t* ∈ *set ts*. *t* ∈ *T*) ∧
(∀ *t* ∈ *set* (*butlast ts*). *chars-of-token t* = [])
⟨*proof*⟩

lemma *limit-Append-path-nonelem-split′*:
p ∈ *limit* (*Append* ($\mathcal{Z}$ *k* (*Suc u*)) *k*) ($\mathcal{P}$ *k u*) ⟹ *p* ∉ $\mathcal{P}$ *k u* ⟹
∃ *q ts*. *p* = *q*@*ts* ∧ *is-continuation k u q ts*
⟨*proof*⟩

lemma *final-step-of-path*: *p* ∈ $\mathcal{P}$ *k u* ⟹ *p* ≠ [] ⟹ (∃ *q ts k′ u′*. *p* = *q*@*ts* ∧ *indexlt k′ u′ k u*
∧ *is-continuation k′ u′ q ts*)
⟨*proof*⟩

lemma *terminals-empty*[*simp*]: *terminals* [] = []
⟨*proof*⟩

lemma *empty-in-*$\mathcal{L}_P$[*simp*]: [] ∈ $\mathcal{L}_P$
⟨*proof*⟩

lemma *admissible-empty*[*simp*]: *admissible* []
⟨*proof*⟩

lemma $\mathfrak{P}$*-are-admissible*: *p* ∈ $\mathfrak{P}$ ⟹ *admissible p*
⟨*proof*⟩

lemma *prefix-of-empty-is-empty*: *is-prefix q* [] ⟹ *q* = []
⟨*proof*⟩

lemma *subset-*$\mathcal{P}$:
assumes *leq*: $k' < k \lor (k' = k \land u' \leq u)$
shows $\mathcal{P}\ k'\ u' \subseteq \mathcal{P}\ k\ u$
⟨*proof*⟩

lemma *empty-path-is-elem*[*simp*]: [] ∈ $\mathcal{P}$ *k u*
⟨*proof*⟩

lemma *is-prefix-of-append*:
assumes *is-prefix p* (*a*@*b*)

shows *is-prefix p a* $\vee$ ($\exists$ b'. $b' \neq []$ $\wedge$ *is-prefix* b' *b* $\wedge$ $p = a@b'$)
⟨*proof*⟩

lemma *prefix-is-continuation*: *is-continuation k u p ts* $\Longrightarrow$ *is-prefix* ts' *ts* $\Longrightarrow$
is-continuation k u p ts'
⟨*proof*⟩

lemma *charslength-0*: ($\forall$ $t \in$ *set ts*. *chars-of-token* $t = []$) = (*charslength ts* = *0*)
⟨*proof*⟩

lemma *is-continuation-in-*$\mathcal{P}$: *is-continuation k u p ts* $\Longrightarrow$ $p@ts \in \mathcal{P}$ k (*Suc u*)
⟨*proof*⟩

lemma *indexlt-subset-*$\mathcal{P}$: *indexlt* k' u' k u $\Longrightarrow$ $\mathcal{P}$ k' (*Suc* u') $\subseteq$ $\mathcal{P}$ k u
⟨*proof*⟩

lemma *prefixes-are-paths*: $p \in \mathcal{P}$ k u $\Longrightarrow$ *is-prefix x p* $\Longrightarrow$ $x \in \mathcal{P}$ k u
⟨*proof*⟩

lemma *empty-or-last-of-suffix*:
assumes $q = q'$ @ $[t]$
assumes $q = p$ @ ts
shows $ts = []$ $\vee$ ($\exists$ ts'. $q' = p$ @ ts' $\wedge$ $ts'@[t] = ts$)
⟨*proof*⟩

lemma *is-prefix-butlast*: *is-prefix q* (*butlast p*) $\Longrightarrow$ *is-prefix q p*
⟨*proof*⟩

lemma *last-step-of-path*:
$q \in \mathcal{P}$ k u $\Longrightarrow$ $q = q'@[t]$ $\Longrightarrow$
$\exists$ k' u'. *indexlt* k' u' k u $\wedge$ $q \in \mathcal{P}$ k' (*Suc* u') $\wedge$ *charslength* $q' = k'$ $\wedge$ $t \in \mathcal{Z}$ k'
(*Suc* u')
⟨*proof*⟩

lemma *charslength-of-butlast-0*: $p \in \mathcal{P}$ k *0* $\Longrightarrow$ $p = q@[t]$ $\Longrightarrow$ *charslength* $q < k$
⟨*proof*⟩

lemma *charslength-of-butlast*: $p \in \mathcal{P}$ k u $\Longrightarrow$ $p = q@[t]$ $\Longrightarrow$ *charslength* $q \leq k$
⟨*proof*⟩

lemma *last-token-of-path*:
assumes $q \in \mathcal{P}$ k u
assumes $q = q'@[t]$
assumes *charslength* $q' = k$
shows $t \in \mathcal{Z}$ k u
⟨*proof*⟩

lemma *final-step-of-path*$'$: $p \in \mathcal{P}$ k u $\Longrightarrow$ $p \notin \mathcal{P}$ k $(u - 1)$ $\Longrightarrow$
$\exists$ q ts. $u > 0$ $\wedge$ $p = q@ts$ $\wedge$ *is-continuation k* $(u - 1)$ *q ts*

⟨*proof*⟩

lemma *is-continuation-continue*:
assumes *is-continuation k u q ts*
assumes *charslength ts = 0*
assumes $t \in \mathcal{Z}\ k\ (Suc\ u)$
assumes *admissible* (*q* @ *ts* @ [*t*])
shows *is-continuation k u q* (*ts*@[*t*])
⟨*proof*⟩

theorem *compatibility-def*:
assumes *p-in-dom*: $p \in \mathcal{P}\ k\ u$
assumes *q-in-dom*: $q \in \mathcal{P}\ k\ u$
assumes *p-charslength*: *charslength p = k*
assumes *q-split*: $q = q'@[t]$
assumes *q'len*: *charslength* $q' = k$
assumes *admissible*: *admissible* (*p* @ [*t*])
shows $p\ @\ [t] \in \mathcal{P}\ k\ u$
⟨*proof*⟩

lemma *is-prefix-admissible*:
assumes *is-prefix a b*
assumes *admissible b*
shows *admissible a*
⟨*proof*⟩

lemma *butlast-split*: $n < length\ q \Longrightarrow butlast\ q = (take\ n\ q)@(drop\ n\ (butlast\ q))$
⟨*proof*⟩

lemma *in-*$\mathcal{P}$*-charslength*:
assumes *p-dom*: $p \in \mathcal{P}\ k\ u$
shows $\exists\ v.\ p \in \mathcal{P}\ (charslength\ p)\ v$
⟨*proof*⟩

theorem *general-compatibility*:
$p \in \mathcal{P}\ k\ u \Longrightarrow q \in \mathcal{P}\ k\ u \Longrightarrow charslength\ p = charslength\ (take\ n\ q)$
$\Longrightarrow charslength\ p \leq k \Longrightarrow admissible\ (p\ @\ (drop\ n\ q)) \Longrightarrow p\ @\ (drop\ n\ q) \in$
$\mathcal{P}\ k\ u$
⟨*proof*⟩

lemma *wellformed-item-derives*:
assumes *wellformed*: *wellformed-item x*
shows *derives* [*item-nonterminal x*] (*item-rhs x*)
⟨*proof*⟩

lemma *wellformed-complete-item-*β:
assumes *wellformed*: *wellformed-item x*

assumes *complete*: *is-complete x*
shows *item-β x* = []
⟨*proof*⟩

lemma *wellformed-complete-item-derives*:
assumes *wellformed*: *wellformed-item x*
assumes *complete*: *is-complete x*
shows *derives* [*item-nonterminal x*] (*item-α x*)
⟨*proof*⟩

lemma *is-derivation-implies-admissible*:
is-derivation (*terminals p* @ δ) ⟹ *is-word* (*terminals p*) ⟹ *admissible p*
⟨*proof*⟩

lemma *item-rhs-of-inc-item*[*simp*]: *item-rhs* (*inc-item x k*) = *item-rhs x*
⟨*proof*⟩

lemma *item-rule-of-inc-item*[*simp*]: *item-rule* (*inc-item x k*) = *item-rule x*
⟨*proof*⟩

lemma *item-origin-of-inc-item*[*simp*]: *item-origin* (*inc-item x k*) = *item-origin x*
⟨*proof*⟩

lemma *item-end-of-inc-item*[*simp*]: *item-end* (*inc-item x k*) = *k*
⟨*proof*⟩

lemma *item-dot-of-inc-item*[*simp*]: *item-dot* (*inc-item x k*) = (*item-dot x*) + *1*
⟨*proof*⟩

lemma *item-nonterminal-of-inc-item*[*simp*]: *item-nonterminal* (*inc-item x k*) = *item-nonterminal x*
⟨*proof*⟩

lemma *wellformed-inc-item*:
assumes *wellformed*: *wellformed-item x*
assumes *next-symbol*: *next-symbol x* = *Some s*
assumes *k-upper-bound*: *k* ≤ *length Doc*
assumes *k-lower-bound*: *k* ≥ *item-end x*
shows *wellformed-item* (*inc-item x k*)
⟨*proof*⟩

lemma *item-α-of-inc-item*:
assumes *wellformed*: *wellformed-item x*
assumes *next-symbol*: *next-symbol x* = *Some s*
shows *item-α* (*inc-item x k*) = *item-α x* @ [*s*]
⟨*proof*⟩

lemma *derives1-pad*:
assumes *derives1*: *derives1 α β*

assumes u: *is-sentence* u
assumes v: *is-sentence* v
shows *derives1* $(u@\alpha@v)$ $(u@\beta@v)$
⟨*proof*⟩

lemma *derives-pad*:
derives α $\beta \Longrightarrow$ *is-sentence* $u \Longrightarrow$ *is-sentence* $v \Longrightarrow$ *derives* $(u@\alpha@v)$ $(u@\beta@v)$
⟨*proof*⟩

lemma *derives1-is-sentence*: *derives1* α $\beta \Longrightarrow$ *is-sentence* $\alpha \wedge$ *is-sentence* β
⟨*proof*⟩

lemma *derives-is-sentence*: *derives* α $\beta \Longrightarrow (\alpha = \beta) \vee$ (*is-sentence* $\alpha \wedge$ *is-sentence* β)
⟨*proof*⟩

lemma *derives-append*:
assumes *au*: *derives* a u
assumes *bv*: *derives* b v
assumes *is-sentence-a*: *is-sentence* a
assumes *is-sentence-b*: *is-sentence* b
shows *derives* $(a@b)$ $(u@v)$
⟨*proof*⟩

lemma *is-sentence-item-α*: *wellformed-item* $x \Longrightarrow$ *is-sentence* (*item-α* x)
⟨*proof*⟩

lemma *is-nonterminal-item-nonterminal*: *wellformed-item* $x \Longrightarrow$ *is-nonterminal* (*item-nonterminal* x)
⟨*proof*⟩

lemma *Complete-elem-in-Gen*:
assumes *I-in-Gen*: $I \subseteq$ *Gen* $(\mathcal{P}\ k\ u)$
assumes k: $k \leq$ *length Doc*
assumes *x-in-Complete*: $x \in$ *Complete* k I
shows $x \in$ *Gen* $(\mathcal{P}\ k\ u)$
⟨*proof*⟩

lemma *Complete-subset-Gen*:
assumes *I-in-Gen-P*: $I \subseteq$ *Gen* $(\mathcal{P}\ k\ u)$
assumes k: $k \leq$ *length Doc*
shows *Complete* k $I \subseteq$ *Gen* $(\mathcal{P}\ k\ u)$
⟨*proof*⟩

lemma *$\mathcal{P}$-are-admissible*: $p \in \mathcal{P}\ k\ u \Longrightarrow$ *admissible* p
⟨*proof*⟩

lemma *is-continuation-base*:
assumes *p-dom*: $p \in \mathcal{P}\ k\ u$

assumes *charslength-p*: $charslength\ p = k$
shows *is-continuation k u p* []
⟨*proof*⟩

lemma *is-continuation-empty-chars*:
$is\text{-}continuation\ k\ u\ q\ ts \Longrightarrow charslength\ (q@ts) = k \Longrightarrow chars\ ts = []$
⟨*proof*⟩

lemma *𝒵-subset*: $u \leq v \Longrightarrow \mathcal{Z}\ k\ u \subseteq \mathcal{Z}\ k\ v$
⟨*proof*⟩

lemma *is-continuation-increase-u*:
assumes *cont*: *is-continuation k u q ts*
assumes *uv*: $u \leq v$
shows *is-continuation k v q ts*
⟨*proof*⟩

lemma *pvalid-next-symbol-derivable*:
assumes *pvalid*: *pvalid p x*
assumes *next-symbol*: $next\text{-}symbol\ x = Some\ s$
shows $\exists\ \delta.\ is\text{-}derivation((terminals\ p)@[s]@\delta)$
⟨*proof*⟩

lemma *pvalid-admissible*:
assumes *pvalid*: *pvalid p x*
shows *admissible p*
⟨*proof*⟩

lemma *pvalid-next-terminal-admissible*:
assumes *pvalid*: *pvalid p x*
assumes *next-symbol*: $next\text{-}symbol\ x = Some\ t$
assumes *terminal*: *is-terminal t*
shows $admissible\ (p@[(t,\ c)])$
⟨*proof*⟩

lemma *𝒳-wellformed*: $t \in \mathcal{X}\ k \Longrightarrow wellformed\text{-}token\ t$
⟨*proof*⟩

lemma *𝒵-wellformed*: $t \in \mathcal{Z}\ k\ u \Longrightarrow wellformed\text{-}token\ t$
⟨*proof*⟩

lemma *Scan-elem-in-Gen*:
assumes *I-in-Gen*: $I \subseteq Gen\ (\mathcal{P}\ k\ u)$
assumes *k*: $k \leq length\ Doc$
assumes *T*: $T \subseteq \mathcal{Z}\ k\ u$
assumes *x-in-Scan*: $x \in Scan\ T\ k\ I$
shows $x \in Gen\ (\mathcal{P}\ k\ u)$
⟨*proof*⟩

lemma *Scan-subset-Gen*:
  assumes *I-in-Gen*: $I \subseteq Gen\ (\mathcal{P}\ k\ u)$
  assumes *k*: $k \leq length\ Doc$
  assumes *T*: $T \subseteq \mathcal{Z}\ k\ u$
  shows $Scan\ T\ k\ I \subseteq Gen\ (\mathcal{P}\ k\ u)$
⟨*proof*⟩

theorem *thmD5*:
  assumes *I*: $I \subseteq Gen\ (\mathcal{P}\ k\ u)$
  assumes *k*: $k \leq length\ Doc$
  assumes *T*: $T \subseteq \mathcal{Z}\ k\ u$
  shows $\pi\ k\ T\ I \subseteq Gen\ (\mathcal{P}\ k\ u)$
⟨*proof*⟩

end

end
theory *TheoremD6*
imports *TheoremD5*
begin

context *LocalLexing* **begin**

definition *inc-dot* :: *nat* ⇒ *item* ⇒ *item*
where
  inc-dot d x = *Item* (*item-rule x*) (*item-dot x* + *d*) (*item-origin x*) (*item-end x*)

lemma *inc-dot-0*[*simp*]: *inc-dot 0 x* = *x*
  ⟨*proof*⟩

lemma *Predict-mk-regular1*:
  ∃ (*P* :: *rule* ⇒ *item* ⇒ *bool*) *F*. *Predict k* = *mk-regular1 P F*
⟨*proof*⟩

lemma *Complete-mk-regular2*:
  ∃ (*P* :: *dummy* ⇒ *item* ⇒ *item* ⇒ *bool*) *F*. *Complete k* = *mk-regular2 P F*
⟨*proof*⟩

lemma *Scan-mk-regular1*:
  ∃ (*P* :: *token* ⇒ *item* ⇒ *bool*) *F*. *Scan T k* = *mk-regular1 P F*
⟨*proof*⟩

lemma *Predict-regular*: *regular* (*Predict k*)
  ⟨*proof*⟩

lemma *Complete-regular*: *regular* (*Complete k*)
  ⟨*proof*⟩

lemma *Scan-regular*: *regular* (*Scan T k*)

⟨proof⟩

lemma *π-functional*: $\pi\ k\ T = limit\ ((Scan\ T\ k)\ o\ (Complete\ k)\ o\ (Predict\ k))$
⟨proof⟩

lemma *π-step-regular*: $regular\ ((Scan\ T\ k)\ o\ (Complete\ k)\ o\ (Predict\ k))$
⟨proof⟩

lemma *π-regular*: $regular\ (\pi\ k\ T)$
⟨proof⟩

lemma *π-fix*: $Scan\ T\ k\ (Complete\ k\ (Predict\ k\ (\pi\ k\ T\ I))) = \pi\ k\ T\ I$
⟨proof⟩

lemma *π-fix′*: $((Scan\ T\ k)\ o\ (Complete\ k)\ o\ (Predict\ k))\ (\pi\ k\ T\ I) = \pi\ k\ T\ I$
⟨proof⟩

lemma *setmonotone-cases*:
assumes *setmonotone f*
shows $f\ X = X \vee X \subset f\ X$
⟨proof⟩

lemma *distribute-fixpoint-over-setmonotone-comp*:
assumes *f*: *setmonotone f*
assumes *g*: *setmonotone g*
assumes *fixpoint*: $(f\ o\ g)\ I = I$
shows $f\ I = I \wedge g\ I = I$
⟨proof⟩

lemma *distribute-fixpoint-over-setmonotone-comp-3*:
assumes *f*: *setmonotone f*
assumes *g*: *setmonotone g*
assumes *h*: *setmonotone h*
assumes *fixpoint*: $(f\ o\ g\ o\ h)\ I = I$
shows $f\ I = I \wedge g\ I = I \wedge h\ I = I$
⟨proof⟩

lemma *Predict-π-fix*: $Predict\ k\ (\pi\ k\ T\ I) = \pi\ k\ T\ I$
⟨proof⟩

lemma *Scan-π-fix*: $Scan\ T\ k\ (\pi\ k\ T\ I) = \pi\ k\ T\ I$
⟨proof⟩

lemma *Complete-π-fix*: $Complete\ k\ (\pi\ k\ T\ I) = \pi\ k\ T\ I$
⟨proof⟩

lemma *π-idempotent*: $\pi\ k\ T\ (\pi\ k\ T\ I) = \pi\ k\ T\ I$
⟨proof⟩

lemma *derivation-shift-identity*[*simp*]: *derivation-shift D 0 0* = *D*
⟨*proof*⟩

lemma *Derivation-skip-prefix*: *Derivation* (*u*@*v*) *D w* ⟹ *derivation-ge D* (*length u*) ⟹
Derivation v (*derivation-shift D* (*length u*) *0*) (*drop* (*length u*) *w*)
⟨*proof*⟩

lemma *leftmost-skip-prefix*: *leftmost i* (*u*@*v*) ⟹ *i* ≥ *length u* ⟹ *leftmost* (*i* − *length u*) *v*
⟨*proof*⟩

lemma *LeftDerivation-skip-prefix*: *LeftDerivation* (*u*@*v*) *D w* ⟹ *derivation-ge D* (*length u*) ⟹
LeftDerivation v (*derivation-shift D* (*length u*) *0*) (*drop* (*length u*) *w*)
⟨*proof*⟩

lemma *splits-at-append*: *splits-at u i u1 N u2* ⟹ *splits-at* (*u*@*v*) *i u1 N* (*u2*@*v*)
⟨*proof*⟩

lemma *LeftDerives1-append-leftmost-unique*: *LeftDerives1* (*a*@*b*) *i r c* ⟹ *leftmost j a* ⟹ *i* = *j*
⟨*proof*⟩

lemma *drop-derivation-shift*:
drop n (*derivation-shift D left right*) = *derivation-shift* (*drop n D*) *left right*
⟨*proof*⟩

lemma *take-derivation-shift*:
take n (*derivation-shift D left right*) = *derivation-shift* (*take n D*) *left right*
⟨*proof*⟩

lemma *derivation-shift-0-shift*: *derivation-shift* (*derivation-shift D left1 0*) *left2 right2* =
derivation-shift D (*left1* + *left2*) *right2*
⟨*proof*⟩

lemma *splits-at-append-prefix*:
splits-at v i α *N* β ⟹ *splits-at* (*u*@*v*) (*i* + *length u*) (*u*@α) *N* β
⟨*proof*⟩

lemma *splits-at-implies-Derives1*: *splits-at* δ *i* α *N* β ⟹ *is-sentence* δ ⟹ *r*∈ $\mathfrak{R}$ ⟹ *fst r* = *N*
⟹ *Derives1* δ *i r* (α@(*snd r*)@β)
⟨*proof*⟩

lemma *Derives1-append-prefix*:
assumes *Derives1*: *Derives1 v i r w*
assumes *u*: *is-sentence u*

shows *Derives1* (*u*@*v*) (*i* + *length u*) *r* (*u*@*w*)
⟨*proof*⟩

lemma *leftmost-prepend-word*: *leftmost i v* ⟹ *is-word u* ⟹ *leftmost* (*i* + *length u*) (*u*@*v*)
⟨*proof*⟩

lemma *LeftDerives1-append-prefix*:
assumes *Derives1*: *LeftDerives1 v i r w*
assumes *u*: *is-word u*
shows *LeftDerives1* (*u*@*v*) (*i* + *length u*) *r* (*u*@*w*)
⟨*proof*⟩

lemma *Derivation-append-prefix*: *Derivation v D w* ⟹ *is-sentence u* ⟹
Derivation (*u*@*v*) (*derivation-shift D 0* (*length u*)) (*u*@*w*)
⟨*proof*⟩

lemma *LeftDerivation-append-prefix*: *LeftDerivation v D w* ⟹ *is-word u* ⟹
LeftDerivation (*u*@*v*) (*derivation-shift D 0* (*length u*)) (*u*@*w*)
⟨*proof*⟩

lemma *derivation-ge-shift-simp*: *derivation-ge D i* ⟹ *i* ≥ *l* ⟹ *r* ≥ *l* ⟹
derivation-shift D l r = *derivation-shift D 0* (*r* − *l*)
⟨*proof*⟩

lemma *append-dropped-prefix*: *is-prefix u v* ⟹ *drop* (*length u*) *v* = *w* ⟹ *u*@*w* = *v*
⟨*proof*⟩

lemma *derivation-ge-shift-plus*:
assumes *derivation-ge D u*
assumes *derivation-ge* (*derivation-shift D u 0*) *v*
shows *derivation-ge D* (*u* + *v*)
⟨*proof*⟩

lemma *LeftDerivation-breakdown*:
LeftDerivation (*u*@*v*) *D w* ⟹ ∃ *n w1 w2*. *w* = *w1* @ *w2* ∧
LeftDerivation u (*take n D*) *w1* ∧
derivation-ge (*drop n D*) (*length w1*) ∧
LeftDerivation v (*derivation-shift* (*drop n D*) (*length w1*) *0*) *w2*
⟨*proof*⟩

lemma *Derives1-terminals-stay*:
assumes *Derives1*: *Derives1 u i r v*
assumes *t-dom*: *t* ∈ *set u*
assumes *terminal*: *is-terminal t*
shows *t* ∈ *set v*
⟨*proof*⟩

lemma *Derivation-terminals-stay*: *Derivation u D v* $\Longrightarrow$ *t* $\in$ *set u* $\Longrightarrow$ *is-terminal t* $\Longrightarrow$ *t* $\in$ *set v*
⟨*proof*⟩

lemma *Derivation-empty-no-terminals*: *Derivation u D* [] $\Longrightarrow$ *t* $\in$ *set u* $\Longrightarrow$ *is-nonterminal t*
⟨*proof*⟩

lemma *mono-subset-elem*: *mono f* $\Longrightarrow A \subseteq B \Longrightarrow x \in f\ A \Longrightarrow x \in f\ B$ ⟨*proof*⟩

lemma *wellformed-inc-dot*: *wellformed-item x* $\Longrightarrow$ *item-dot x* + *d* $\leq$ *length* (*item-rhs x*) $\Longrightarrow$
wellformed-item(*inc-dot d x*)
⟨*proof*⟩

lemma *init-item-dot*[*simp*]: *item-dot* (*init-item r k*) = *0*
⟨*proof*⟩

lemma *init-item-rhs*[*simp*]: *item-rhs* (*init-item r k*) = *snd r*
⟨*proof*⟩

lemma *init-item-β*[*simp*]: *item-β* (*init-item r k*) = *snd r*
⟨*proof*⟩

lemma *mono-π*: *mono* (π *k T*)
⟨*proof*⟩

lemma *π-subset-elem-trans*:
assumes *Y*: $Y \subseteq \pi\ k\ T\ X$
assumes *z*: $z \in \pi\ k\ T\ Y$
shows $z \in \pi\ k\ T\ X$
⟨*proof*⟩

lemma *inc-dot-origin*[*simp*]: *item-origin* (*inc-dot d x*) = *item-origin x*
⟨*proof*⟩

lemma *inc-dot-end*[*simp*]: *item-end* (*inc-dot d x*) = *item-end x*
⟨*proof*⟩

lemma *inc-dot-rhs*[*simp*]: *item-rhs* (*inc-dot d x*) = *item-rhs x*
⟨*proof*⟩

lemma *inc-dot-dot*[*simp*]: *item-dot* (*inc-dot d x*) = *item-dot x* + *d*
⟨*proof*⟩

lemma *inc-dot-nonterminal*[*simp*]: *item-nonterminal* (*inc-dot d x*) = *item-nonterminal x*
⟨*proof*⟩

lemma *Predict-subset-π*: *Predict k X* ⊆ *π k T X*
⟨*proof*⟩

lemma *Complete-subset-π*: *Complete k X* ⊆ *π k T X*
⟨*proof*⟩

lemma *inc-inc-dot*[*simp*]: *inc-dot a* (*inc-dot b x*) = *inc-dot* (*a* + *b*) *x*
⟨*proof*⟩

lemma *thmD6-Left*: *wellformed-item x* ⟹ *item-β x* = *δ* @ *ω* ⟹ *item-end x* = *k* ⟹
LeftDerivation δ D [] ⟹ *inc-dot* (*length δ*) *x* ∈ *π k* {} {*x*}
⟨*proof*⟩

lemma *derives-empty-implies-LeftDerivation*: *derives δ* [] ⟹ ∃ *D*. *LeftDerivation δ D* []
⟨*proof*⟩

lemma *thmD6*: *wellformed-item x* ⟹ *item-β x* = *δ* @ *ω* ⟹ *item-end x* = *k* ⟹

derives δ [] ⟹ *inc-dot* (*length δ*) *x* ∈ *π k* {} {*x*}
⟨*proof*⟩

end

end
theory *TheoremD7*
imports *TheoremD6*
begin

context *LocalLexing* **begin**

lemma *Derives1-keep-first-terminal*: *Derives1* (*x*#*u*) *i r* (*y*#*v*) ⟹ *is-terminal x* ⟹ *x* = *y*
⟨*proof*⟩

lemma *Derives1-nonterminal-head*:
assumes *Derives1 u i r* (*N*#*v*)
assumes *is-nonterminal N*
shows ∃ *u*′ *M*. *u* = *M*#*u*′ ∧ *is-nonterminal M*
⟨*proof*⟩

lemma *sentence-starts-with-nonterminal*:
assumes *is-nonterminal N*
assumes *derives u* []
shows ∃ *X r*. *u*@[*N*] = *X*#*r* ∧ *is-nonterminal X*
⟨*proof*⟩

lemma *Derives1-nonterminal-head*′:

assumes *Derives1 u i r* ($v1$@[N]@$v2$)
assumes *is-nonterminal N*
assumes *derives v1* []
shows $\exists$ u' M. $u = M\#u' \wedge$ *is-nonterminal M*
⟨*proof*⟩

lemma *thmD7-helper*:
assumes *LeftDerivation* [$\mathfrak{S}$] *D* ($N\#v$)
assumes *is-nonterminal N*
assumes $\mathfrak{S} \neq N$
shows $\exists$ *n M a a1 a2 w*. $n <$ *length D* $\wedge$ (M, a) $\in \mathfrak{R} \wedge$ *LeftDerivation* [$\mathfrak{S}$] (*take n D*) ($M\#w$) $\wedge$
$a = a1$ @ [N] @ $a2 \wedge$ *derives a1* []
⟨*proof*⟩

lemma *head-of-item-β-is-next-symbol*:
wellformed-item x $\Longrightarrow$ *item-β x* $= t\#\delta \Longrightarrow$ *next-symbol x* = *Some t*
⟨*proof*⟩

lemma *next-symbol-predicts*: *next-symbol x* = *Some N* $\Longrightarrow$ (N, a) $\in \mathfrak{R} \Longrightarrow k =$ *item-end x* $\Longrightarrow$
init-item (N, a) $k \in$ *Predict k* $\{x\}$
⟨*proof*⟩

lemma *thmD7-LeftDerivation*: *LeftDerivation* [$\mathfrak{S}$] *D* ($N\#\gamma$) $\Longrightarrow$ *is-nonterminal N* $\Longrightarrow$ (N, α) $\in \mathfrak{R} \Longrightarrow$
init-item (N, α) $0 \in \pi\ 0\ \{\}$ *Init*
⟨*proof*⟩

theorem *thmD7*: *is-derivation* ($N\#\gamma$) $\Longrightarrow$ *is-nonterminal N* $\Longrightarrow$ (N, α) $\in \mathfrak{R} \Longrightarrow$

init-item (N, α) $0 \in \pi\ 0\ \{\}$ *Init*
⟨*proof*⟩

end

end
theory *TheoremD8*
imports *TheoremD7*
begin

context *LocalLexing* **begin**

lemma *wellformed-tokens-empty-path*[*simp*]: *wellformed-tokens* []
⟨*proof*⟩

lemma *$\mathcal{P}$-0-0-Gen*: *Gen* ($\mathcal{P}$ 0 0) = { *x* . *wellformed-item x* $\wedge$ *item-origin x* $= 0 \wedge$ *item-end x* $= 0 \wedge$
derives (*item-α x*) [] $\wedge$ ($\exists$ γ. *is-derivation* ([*item-nonterminal x*] @ γ)) }

⟨*proof*⟩

lemma *Init-subset-Gen*: *Init* ⊆ *Gen* (𝒫 *0 0*)
⟨*proof*⟩

lemma *𝒥-0-0-subset-Gen*: 𝒥 *0 0* ⊆ *Gen* (𝒫 *0 0*)
⟨*proof*⟩

lemma *inc-dot-rule*[*simp*]: *item-rule* (*inc-dot d x*) = *item-rule x*
⟨*proof*⟩

lemma *init-item-rule*[*simp*]: *item-rule* (*init-item r k*) = *r*
⟨*proof*⟩

lemma *item-dot-is-α-length*: *wellformed-item x* ⟹ *item-dot x* = *length* (*item-α x*)
⟨*proof*⟩

lemma *Gen-subset-𝒥-0-0-helper*:
assumes *wellformed-item x*
assumes *item-origin x* = *0*
assumes *item-end x* = *0*
assumes *derives* (*item-α x*) []
assumes *is-derivation* (*item-nonterminal x* # γ)
shows *x* ∈ π *0* {} *Init*
⟨*proof*⟩

lemma *Gen-subset-𝒥-0-0*: *Gen* (𝒫 *0 0*) ⊆ 𝒥 *0 0*
⟨*proof*⟩

theorem *thmD8*: 𝒥 *0 0* = *Gen* (𝒫 *0 0*)
⟨*proof*⟩

end

end
theory *TheoremD9*
imports *TheoremD8*
begin

context *LocalLexing* **begin**

definition *items-le* :: *nat* ⇒ *items* ⇒ *items*
where
items-le k I = { *x* . *x* ∈ *I* ∧ *item-end x* ≤ *k* }

definition *items-eq* :: *nat* ⇒ *items* ⇒ *items*
where
items-eq k I = { *x* . *x* ∈ *I* ∧ *item-end x* = *k* }

definition *paths-le* :: *nat* ⇒ *tokens set* ⇒ *tokens set*
where
paths-le k P = { *p* . *p* ∈ *P* ∧ *charslength p* ≤ *k* }

definition *paths-eq* :: *nat* ⇒ *tokens set* ⇒ *tokens set*
where
paths-eq k P = { *p* . *p* ∈ *P* ∧ *charslength p* = *k* }

lemma *items-le-pointwise*: *pointwise* (*items-le k*)
⟨*proof*⟩

lemma *items-le-is-filter*: *items-le k I* ⊆ *I*
⟨*proof*⟩

lemma *items-eq-pointwise*: *pointwise* (*items-eq k*)
⟨*proof*⟩

lemma *items-eq-is-filter*: *items-eq k I* ⊆ *I*
⟨*proof*⟩

lemma *paths-le-pointwise*: *pointwise* (*paths-le k*)
⟨*proof*⟩

lemma *paths-le-continuous*: *continuous* (*paths-le k*)
⟨*proof*⟩

lemma *paths-le-mono*: *mono* (*paths-le k*)
⟨*proof*⟩

lemma *paths-le-is-filter*: *paths-le k P* ⊆ *P*
⟨*proof*⟩

lemma *paths-eq-pointwise*: *pointwise* (*paths-eq k*)
⟨*proof*⟩

lemma *paths-eq-is-filter*: *paths-eq k P* ⊆ *P*
⟨*proof*⟩

lemma *Predict-item-end*: *x* ∈ *Predict k Y* ⟹ *item-end x* = *k* ∨ *x* ∈ *Y*
⟨*proof*⟩

lemma *Complete-item-end*: *x* ∈ *Complete k Y* ⟹ *item-end x* = *k* ∨ *x* ∈ *Y*
⟨*proof*⟩

lemma *𝒥-0-0-item-end*: *x* ∈ *𝒥 0 0* ⟹ *item-end x* = *0*
⟨*proof*⟩

lemma *items-le-𝒥-0-0*: *items-le 0* (*𝒥 0 0*) = *𝒥 0 0*

⟨*proof*⟩

lemma *paths-le-*$\mathcal{P}$*-0-0*: *paths-le 0* ($\mathcal{P}$ *0 0*) = $\mathcal{P}$ *0 0*
⟨*proof*⟩

definition *empty-tokens* :: *token set* ⇒ *token set*
where
empty-tokens T = { *t* . *t* ∈ *T* ∧ *chars-of-token t* = [] }

lemma *items-le-Predict*: *items-le k* (*Predict k I*) = *Predict k* (*items-le k I*)
⟨*proof*⟩

lemma *items-le-Complete*:
wellformed-items I ⟹ *items-le k* (*Complete k I*) = *Complete k* (*items-le k I*)
⟨*proof*⟩

lemma *items-le-Scan*:
items-le k (*Scan T k I*) = *Scan* (*empty-tokens T*) *k* (*items-le k I*)
⟨*proof*⟩

lemma *wellformed-items-Gen*: *wellformed-items* (*Gen P*)
⟨*proof*⟩

lemma *wellformed-*$\mathcal{J}$*-0-0*: *wellformed-items* ($\mathcal{J}$ *0 0*)
⟨*proof*⟩

lemma *wellformed-items-Predict*:
wellformed-items I ⟹ *wellformed-items* (*Predict k I*)
⟨*proof*⟩

lemma *wellformed-items-Complete*:
wellformed-items I ⟹ *wellformed-items* (*Complete k I*)
⟨*proof*⟩

lemma $\mathcal{X}$*-length-bound*: (*t*, *c*) ∈ $\mathcal{X}$ *k* ⟹ *k* + *length c* ≤ *length Doc*
⟨*proof*⟩

lemma *wellformed-items-Scan*:
wellformed-items I ⟹ *T* ⊆ $\mathcal{X}$ *k* ⟹ *wellformed-items* (*Scan T k I*)
⟨*proof*⟩

lemma *wellformed-items-π*:
assumes *wellformed-items I*
assumes *T* ⊆ $\mathcal{X}$ *k*
shows *wellformed-items* (π *k T I*)
⟨*proof*⟩

lemma $\mathcal{J}$*-subset-Suc-u*: $\mathcal{J}$ *k u* ⊆ $\mathcal{J}$ *k* (*Suc u*)
⟨*proof*⟩

lemma *mono-TokensAt*: *mono* (*TokensAt k*)
⟨*proof*⟩

lemma $\mathcal{T}$*-subset-TokensAt*: $\mathcal{T}$ *k u* ⊆ *TokensAt k* ($\mathcal{J}$ *k u*)
⟨*proof*⟩

lemma *TokensAt-subset-*$\mathcal{X}$: *TokensAt k I* ⊆ $\mathcal{X}$ *k*
⟨*proof*⟩

lemma *wellformed-items-*$\mathcal{J}$*-induct-u*:
assumes *wellformed-items* ($\mathcal{J}$ *k u*)
shows *wellformed-items* ($\mathcal{J}$ *k* (*Suc u*))
⟨*proof*⟩

lemma *wellformed-items-*$\mathcal{J}$*-k-u-if-0*: *wellformed-items* ($\mathcal{J}$ *k 0*) ⟹ *wellformed-items* ($\mathcal{J}$ *k u*)
⟨*proof*⟩

lemma *wellformed-items-natUnion*: (⋀ *k*. *wellformed-items* (*I k*)) ⟹ *wellformed-items* (*natUnion I*)
⟨*proof*⟩

lemma *wellformed-items-*$\mathcal{I}$*-k-if-0*: *wellformed-items* ($\mathcal{J}$ *k 0*) ⟹ *wellformed-items* ($\mathcal{I}$ *k*)
⟨*proof*⟩

lemma *wellformed-items-*$\mathcal{J}$*-*$\mathcal{I}$: *wellformed-items* ($\mathcal{J}$ *k u*) ∧ *wellformed-items* ($\mathcal{I}$ *k*)
⟨*proof*⟩

lemma *wellformed-items-*$\mathcal{J}$: *wellformed-items* ($\mathcal{J}$ *k u*)
⟨*proof*⟩

lemma *wellformed-items-*$\mathcal{I}$: *wellformed-items* ($\mathcal{I}$ *k*)
⟨*proof*⟩

lemma *funpower-consume-function*:
assumes *law*: ⋀ *X*. *P X* ⟹ *f* (*g X*) = *h* (*f X*) ∧ *P* (*g X*)
shows *P I* ⟹ *P* (*funpower g n I*) ∧ *f* (*funpower g n I*) = *funpower h n* (*f I*)
⟨*proof*⟩

lemma *limit-consume-function*:
assumes *continuous*: *continuous f*
assumes *law*: ⋀ *X*. *P X* ⟹ *f* (*g X*) = *h* (*f X*) ∧ *P* (*g X*)
assumes *setmonotone*: *setmonotone g*
shows *P I* ⟹ *f* (*limit g I*) = *limit h* (*f I*)
⟨*proof*⟩

lemma *items-le-*π*-swap*:

assumes *wellformed-I*: *wellformed-items I*
assumes *T*: $T \subseteq \mathcal{X}$ *k*
shows *items-le k* (π *k T I*) = π *k* (*empty-tokens T*) (*items-le k I*)
⟨*proof*⟩

lemma *items-le-idempotent*: *items-le k* (*items-le k I*) = *items-le k I*
⟨*proof*⟩

lemma *paths-le-idempotent*: *paths-le k* (*paths-le k P*) = *paths-le k P*
⟨*proof*⟩

lemma *items-le-fix-D*:
assumes *items-le-fix*: *items-le k I* = *I*
assumes *x-dom*: $x \in I$
shows *item-end* $x \leq k$
⟨*proof*⟩

lemma *remove-paths-le-in-subset-Gen*:
assumes *items-le k I* = *I*
assumes $I \subseteq$ *Gen P*
shows $I \subseteq$ *Gen* (*paths-le k P*)
⟨*proof*⟩

lemma *mono-Gen*: *mono Gen*
⟨*proof*⟩

lemma *empty-tokens-idempotent*: *empty-tokens* (*empty-tokens T*) = *empty-tokens T*
⟨*proof*⟩

lemma *empty-tokens-is-filter*: *empty-tokens* $T \subseteq T$
⟨*proof*⟩

lemma *items-le-paths-le*: *items-le k* (*Gen P*) = *Gen* (*paths-le k P*)
⟨*proof*⟩

lemma *bin-items-le*[*symmetric*]: *bin I k* = *bin* (*items-le k I*) *k*
⟨*proof*⟩

lemma *TokensAt-items-le*[*symmetric*]: *TokensAt k I* = *TokensAt k* (*items-le k I*)
⟨*proof*⟩

lemma *by-length-paths-le*[*symmetric*]: *by-length k P* = *by-length k* (*paths-le k P*)
⟨*proof*⟩

lemma $\mathcal{W}$*-paths-le*[*symmetric*]: $\mathcal{W}$ *P k* = $\mathcal{W}$ (*paths-le k P*) *k*
⟨*proof*⟩

theorem $\mathcal{T}$*-equals-*$\mathcal{Z}$*-induct-step*:

assumes *induct*: $\textit{items-le}\ k\ (\mathcal{J}\ k\ u) = \textit{Gen}\ (\textit{paths-le}\ k\ (\mathcal{P}\ k\ u))$
assumes *induct-tokens*: $\mathcal{T}\ k\ u = \mathcal{Z}\ k\ u$
shows $\mathcal{T}\ k\ (\textit{Suc}\ u) = \mathcal{Z}\ k\ (\textit{Suc}\ u)$
⟨*proof*⟩

theorem *thmD9*:
assumes *induct*: $\textit{items-le}\ k\ (\mathcal{J}\ k\ u) = \textit{Gen}\ (\textit{paths-le}\ k\ (\mathcal{P}\ k\ u))$
assumes *induct-tokens*: $\mathcal{T}\ k\ u = \mathcal{Z}\ k\ u$
assumes *k*: $k \leq \textit{length Doc}$
shows $\textit{items-le}\ k\ (\mathcal{J}\ k\ (\textit{Suc}\ u)) \subseteq \textit{Gen}\ (\textit{paths-le}\ k\ (\mathcal{P}\ k\ (\textit{Suc}\ u)))$
⟨*proof*⟩

end

end
theory *Ladder*
imports *TheoremD9*
begin

context *LocalLexing* **begin**

definition *LeftDerivationFix* :: *sentence* ⇒ *nat* ⇒ *derivation* ⇒ *nat* ⇒ *sentence* ⇒ *bool*
where
LeftDerivationFix $\alpha\ i\ D\ j\ \beta$ = (*is-sentence* α ∧ *is-sentence* β
∧ *LeftDerivation* $\alpha\ D\ \beta$ ∧ $i <$ *length* α ∧ $j <$ *length* β
∧ $\alpha\ !\ i = \beta\ !\ j$ ∧ (∃ *E F*. $D = E$@(*derivation-shift F 0* (*Suc j*)) ∧
LeftDerivation (*take i* α) *E* (*take j* β) ∧
LeftDerivation (*drop* (*Suc i*) α) *F* (*drop* (*Suc j*) β)))

definition *LeftDerivationIntro* ::
sentence ⇒ *nat* ⇒ *rule* ⇒ *nat* ⇒ *derivation* ⇒ *nat* ⇒ *sentence* ⇒ *bool*
where
LeftDerivationIntro α *i r ix D j* γ = (∃ β. *LeftDerives1* α *i r* β ∧
ix < *length* (*snd r*) ∧ (*snd r*) ! *ix* = γ ! *j* ∧
LeftDerivationFix β (*i* + *ix*) *D j* γ)

lemma *LeftDerivationFix-empty*[*simp*]: *is-sentence* α ⟹ $i <$ *length* α ⟹ *Left-DerivationFix* α *i* [] *i* α
⟨*proof*⟩

lemma *Derive-empty*[*simp*]: *Derive a* [] = *a*
⟨*proof*⟩

lemma *LeftDerivation-append1*: *LeftDerivation a* (*D*@[(*i*, *r*)]) *c* ⟹ ∃ *b*. *Left-Derivation a D b*
∧ *LeftDerives1 b i r c*
⟨*proof*⟩

lemma *Derivation-append1*: *Derivation a* $(D@[(i, r)])$ *c* $\Longrightarrow \exists$ *b*. *Derivation a D b*
$\wedge$ *Derives1 b i r c*
⟨*proof*⟩

lemma *Derivation-take-derive*:
assumes *Derivation a D b*
shows *Derivation a* (*take n D*) (*Derive a* (*take n D*))
⟨*proof*⟩

lemma *LeftDerivation-take-derive*:
assumes *LeftDerivation a D b*
shows *LeftDerivation a* (*take n D*) (*Derive a* (*take n D*))
⟨*proof*⟩

lemma *Derivation-Derive-take-Derives1*:
assumes $N \neq 0$
assumes $N \leq$ *length D*
assumes *Derivation a D b*
assumes α: α = *Derive a* (*take* $(N - 1)$ *D*)
assumes β = *Derive a* (*take N D*)
shows *Derives1* α (*fst* (*D* ! $(N - 1)$)) (*snd* (*D* ! $(N - 1)$)) β
⟨*proof*⟩

lemma *LeftDerivation-Derive-take-LeftDerives1*:
assumes $N \neq 0$
assumes $N \leq$ *length D*
assumes *LeftDerivation a D b*
assumes α: α = *Derive a* (*take* $(N - 1)$ *D*)
assumes β = *Derive a* (*take N D*)
shows *LeftDerives1* α (*fst* (*D* ! $(N - 1)$)) (*snd* (*D* ! $(N - 1)$)) β
⟨*proof*⟩

lemma *LeftDerives1-skip-prefix*:
length a $\leq i \Longrightarrow$ *LeftDerives1* $(a@b)$ *i r* $(a@c) \Longrightarrow$ *LeftDerives1 b* (*i* − *length a*) *r c*
⟨*proof*⟩

lemma *LeftDerives1-skip-suffix*:
assumes *i*: *i* < *length a*
assumes *D*: *LeftDerives1* $(a@c)$ *i r* $(b@c)$
shows *LeftDerives1 a i r b*
⟨*proof*⟩

lemma *LeftDerives1-X-is-part-of-rule*[*consumes 2*, *case-names Suffix Prefix*]:
assumes *aXb*: *LeftDerives1* δ *i r* $(a@[X]@b)$
assumes *split*: *splits-at* δ *i* α *N* β
assumes *prefix*: $\bigwedge \beta.\ \delta = a\ @\ [X]\ @\ \beta \Longrightarrow$ *length a* $< i \Longrightarrow$ *is-word* $(a\ @\ [X])$
$\Longrightarrow$

LeftDerives1 β $(i - \mathit{length}\ a - 1)$ *r b* $\Longrightarrow$ *False*
assumes *suffix*: $\bigwedge$ α. $\delta = \alpha$ @ [*X*] @ *b* $\Longrightarrow$ *LeftDerives1* α *i r a* $\Longrightarrow$ *False*
shows $\exists$ *u v*. *a* = α @ *u* $\wedge$ *b* = *v* @ β $\wedge$ (*snd r*) = *u*@[*X*]@*v*
⟨*proof*⟩

lemma *LeftDerivationFix-grow-suffix*:
assumes *LDF*: *LeftDerivationFix* (*b1*@[*X*]@*b2*) (*length b1*) *D j c*
assumes *suffix-b2*: *LeftDerives1 suffix e r b2*
assumes *is-word-b1X*: *is-word* (*b1*@[*X*])
shows *LeftDerivationFix* (*b1*@[*X*]@*suffix*) (*length b1*) ((*e* + *length* (*b1*@[*X*]), *r*)#*D*) *j c*
⟨*proof*⟩

lemma *Derives1-append-suffix*:
assumes *Derives1*: *Derives1 v i r w*
assumes *u*: *is-sentence u*
shows *Derives1* (*v*@*u*) *i r* (*w*@*u*)
⟨*proof*⟩

lemma *leftmost-append-suffix*: *leftmost i v* $\Longrightarrow$ *leftmost i* (*v*@*u*)
⟨*proof*⟩

lemma *LeftDerives1-append-suffix*:
assumes *Derives1*: *LeftDerives1 v i r w*
assumes *u*: *is-sentence u*
shows *LeftDerives1* (*v*@*u*) *i r* (*w*@*u*)
⟨*proof*⟩

lemma *LeftDerivationFix-is-sentence*:
LeftDerivationFix a i D j b $\Longrightarrow$ *is-sentence a* $\wedge$ *is-sentence b*
⟨*proof*⟩

lemma *LeftDerivationIntro-is-sentence*:
LeftDerivationIntro α *i r ix D j* γ $\Longrightarrow$ *is-sentence* α $\wedge$ *is-sentence* γ
⟨*proof*⟩

lemma *LeftDerivationFix-grow-prefix*:
assumes *LDF*: *LeftDerivationFix* (*b1*@[*X*]@*b2*) (*length b1*) *D j c*
assumes *prefix-b1*: *LeftDerives1 prefix e r b1*
shows *LeftDerivationFix* (*prefix*@[*X*]@*b2*) (*length prefix*) ((*e*, *r*)#*D*) *j c*
⟨*proof*⟩

lemma *LeftDerivationFixOrIntro*:
LeftDerivation a D γ $\Longrightarrow$ *is-sentence* γ $\Longrightarrow$ $j <$ *length* γ $\Longrightarrow$
($\exists$ *i*. *LeftDerivationFix a i D j* γ) $\vee$
($\exists$ *d* α *ix*. *d* < *length D* $\wedge$ *LeftDerivation a* (*take d D*) α $\wedge$
LeftDerivationIntro α (*fst* (*D* ! *d*)) (*snd* (*D* ! *d*)) *ix* (*drop* (*Suc d*) *D*) *j* γ)
⟨*proof*⟩

type-synonym *deriv* = *nat* × *nat* × *nat*
type-synonym *ladder* = *deriv list*

definition *deriv-n* :: *deriv* ⇒ *nat* **where**
deriv-n d = *fst d*

definition *deriv-j* :: *deriv* ⇒ *nat* **where**
deriv-j d = *fst* (*snd d*)

definition *deriv-ix* :: *deriv* ⇒ *nat* **where**
deriv-ix d = *snd* (*snd d*)

definition *deriv-i* :: *deriv* ⇒ *nat* **where**
deriv-i d = *snd* (*snd d*)

definition *ladder-j* :: *ladder* ⇒ *nat* ⇒ *nat* **where**
ladder-j L index = *deriv-j* (*L* ! *index*)

definition *ladder-i* :: *ladder* ⇒ *nat* ⇒ *nat* **where**
ladder-i L index = (*if index* = *0 then deriv-i* (*hd L*) *else ladder-j L* (*index* − *1*))

definition *ladder-n* :: *ladder* ⇒ *nat* ⇒ *nat* **where**
ladder-n L index = *deriv-n* (*L* ! *index*)

definition *ladder-prev-n* :: *ladder* ⇒ *nat* ⇒ *nat* **where**
ladder-prev-n L index = (*if index* = *0 then 0 else* (*ladder-n L* (*index* − *1*)))

definition *ladder-ix* :: *ladder* ⇒ *nat* ⇒ *nat* **where**
ladder-ix L index = (*if index* = *0 then undefined else deriv-ix* (*L* ! *index*))

definition *ladder-last-j* :: *ladder* ⇒ *nat* **where**
ladder-last-j L = *ladder-j L* (*length L* − *1*)

definition *ladder-last-n* :: *ladder* ⇒ *nat* **where**
ladder-last-n L = *ladder-n L* (*length L* − *1*)

definition *is-ladder* :: *derivation* ⇒ *ladder* ⇒ *bool* **where**
is-ladder D L = (*L* ≠ [] ∧
(∀ *u*. *u* < *length L* ⟶ *ladder-n L u* ≤ *length D*) ∧
(∀ *u v*. *u* < *v* ∧ *v* < *length L* ⟶ *ladder-n L u* < *ladder-n L v*) ∧
ladder-last-n L = *length D*)

definition *ladder-γ* :: *sentence* ⇒ *derivation* ⇒ *ladder* ⇒ *nat* ⇒ *sentence* **where**
ladder-γ a D L index = *Derive a* (*take* (*ladder-n L index*) *D*)

definition *ladder-α* :: *sentence* ⇒ *derivation* ⇒ *ladder* ⇒ *nat* ⇒ *sentence* **where**
ladder-α a D L index = (*if index* = *0 then a else ladder-γ a D L* (*index* − *1*))

definition *LeftDerivationIntrosAt* :: *sentence* ⇒ *derivation* ⇒ *ladder* ⇒ *nat* ⇒

bool **where**
LeftDerivationIntrosAt a D L index = (
let α = *ladder-*α *a D L index in*
let i = *ladder-i L index in*
let j = *ladder-j L index in*
let ix = *ladder-ix L index in*
let γ = *ladder-*γ *a D L index in*
let n = *ladder-n L* (*index* − *1*) *in*
let m = *ladder-n L index in*
let e = *D* ! *n in*
let E = *drop* (*Suc n*) (*take m D*) *in*
i = *fst e* ∧
LeftDerivationIntro α *i* (*snd e*) *ix E j* γ)

definition *LeftDerivationIntros* :: *sentence* ⇒ *derivation* ⇒ *ladder* ⇒ *bool* **where**
LeftDerivationIntros a D L = (
∀ *index*. *1* ≤ *index* ∧ *index* < *length L* ⟶ *LeftDerivationIntrosAt a D L index*)

definition *LeftDerivationLadder* :: *sentence* ⇒ *derivation* ⇒ *ladder* ⇒ *sentence* ⇒ *bool* **where**
LeftDerivationLadder a D L b = (
LeftDerivation a D b ∧
is-ladder D L ∧
LeftDerivationFix a (*ladder-i L 0*) (*take* (*ladder-n L 0*) *D*) (*ladder-j L 0*) (*ladder-*γ *a D L 0*) ∧
LeftDerivationIntros a D L)

definition *mk-deriv-fix* :: *nat* ⇒ *nat* ⇒ *nat* ⇒ *deriv* **where**
mk-deriv-fix i n j = (*n*, *j*, *i*)

definition *mk-deriv-intro* :: *nat* ⇒ *nat* ⇒ *nat* ⇒ *deriv* **where**
mk-deriv-intro ix n j = (*n*, *j*, *ix*)

lemma *mk-deriv-fix-i*[*simp*]: *deriv-i* (*mk-deriv-fix i n j*) = *i*
⟨*proof*⟩

lemma *mk-deriv-fix-j*[*simp*]: *deriv-j* (*mk-deriv-fix i n j*) = *j*
⟨*proof*⟩

lemma *mk-deriv-fix-n*[*simp*]: *deriv-n* (*mk-deriv-fix i n j*) = *n*
⟨*proof*⟩

lemma *mk-deriv-intro-i*[*simp*]: *deriv-i* (*mk-deriv-intro i n j*) = *i*
⟨*proof*⟩

lemma *mk-deriv-intro-ix*[*simp*]: *deriv-ix* (*mk-deriv-intro ix n j*) = *ix*
⟨*proof*⟩

lemma *mk-deriv-intro-j*[*simp*]: *deriv-j* (*mk-deriv-intro i n j*) = *j*
⟨*proof*⟩

lemma *mk-deriv-intro-n*[*simp*]: *deriv-n* (*mk-deriv-intro i n j*) = *n*
⟨*proof*⟩

lemma *LeftDerivationFix-implies-ex-ladder*:
LeftDerivationFix a i D j $\gamma \Longrightarrow \exists$ *L*. *LeftDerivationLadder a D L* $\gamma \wedge$
ladder-last-j L = *j* $\wedge$ *ladder-last-n L* = *length D*
⟨*proof*⟩

lemma *trivP*[*case-names prems*]: *P* $\Longrightarrow$ *P* ⟨*proof*⟩

lemma *LeftDerivationLadder-ladder-n-bound*:
assumes *LeftDerivationLadder a D L b*
assumes *index* < *length L*
shows *ladder-n L index* $\leq$ *length D*
⟨*proof*⟩

lemma *LeftDerivationLadder-deriv-n-bound*:
assumes *LeftDerivationLadder a D L b*
assumes *index* < *length L*
shows *deriv-n* (*L* ! *index*) $\leq$ *length D*
⟨*proof*⟩

lemma *ladder-n-simp1*[*simp*]: *u* < *length L* $\Longrightarrow$ *ladder-n* (*L*@*L'*) *u* = *ladder-n L u*
⟨*proof*⟩

lemma *ladder-n-simp2*[*simp*]: *ladder-n* (*L*@[*d*]) (*length L*) = *deriv-n d*
⟨*proof*⟩

lemma *ladder-j-simp1*[*simp*]: *u* < *length L* $\Longrightarrow$ *ladder-j* (*L*@*L'*) *u* = *ladder-j L u*
⟨*proof*⟩

lemma *ladder-j-simp2*[*simp*]: *ladder-j* (*L*@[*d*]) (*length L*) = *deriv-j d*
⟨*proof*⟩

lemma *ladder-i-simp1*[*simp*]: *u* < *length L* $\Longrightarrow$ *ladder-i* (*L*@*L'*) *u* = *ladder-i L u*
⟨*proof*⟩

lemma *ladder-ix-simp1*[*simp*]: *u* < *length L* $\Longrightarrow$ *ladder-ix* (*L*@*L'*) *u* = *ladder-ix L u*
⟨*proof*⟩

lemma *ladder-ix-simp2*[*simp*]: *L* $\neq$ [] $\Longrightarrow$ *ladder-ix* (*L*@[*d*]) (*length L*) = *deriv-ix d*
⟨*proof*⟩

lemma *ladder-γ-simp1*[*simp*]: $u < length\ L \Longrightarrow ladder\text{-}\gamma\ a\ D\ (L@L')\ u = ladder\text{-}\gamma\ a\ D\ L\ u$
⟨*proof*⟩

lemma *ladder-γ-simp2*[*simp*]: $u < length\ L \Longrightarrow is\text{-}ladder\ D\ L \Longrightarrow$
$ladder\text{-}\gamma\ a\ (D@D')\ L\ u = ladder\text{-}\gamma\ a\ D\ L\ u$
⟨*proof*⟩

lemma *ladder-α-simp1*[*simp*]: $u < length\ L \Longrightarrow ladder\text{-}\alpha\ a\ D\ (L@L')\ u = ladder\text{-}\alpha\ a\ D\ L\ u$
⟨*proof*⟩

lemma *ladder-α-simp2*[*simp*]: $u < length\ L \Longrightarrow is\text{-}ladder\ D\ L \Longrightarrow$
$ladder\text{-}\alpha\ a\ (D@D')\ L\ u = ladder\text{-}\alpha\ a\ D\ L\ u$
⟨*proof*⟩

lemma *ladder-n-minus-1-bound*: $is\text{-}ladder\ D\ L \Longrightarrow index \geq 1 \Longrightarrow index < length\ L \Longrightarrow$
$ladder\text{-}n\ L\ (index - Suc\ 0) < length\ D$
⟨*proof*⟩

lemma *LeftDerivationIntrosAt-ignore-appendix*:
assumes *is-ladder*: *is-ladder D L*
assumes *hyp*: *LeftDerivationIntrosAt a D L index*
assumes *index-ge*: $index \geq 1$
assumes *index-less*: $index < length\ L$
shows *LeftDerivationIntrosAt a* $(D\ @\ D')\ (L\ @\ L')$ *index*
⟨*proof*⟩

lemma *ladder-i-eq-last-j*: $L \neq [] \Longrightarrow ladder\text{-}i\ (L\ @\ L')\ (length\ L) = ladder\text{-}last\text{-}j\ L$
⟨*proof*⟩

lemma *ladder-last-n-intro*: $L \neq [] \Longrightarrow ladder\text{-}n\ L\ (length\ L - Suc\ 0) = ladder\text{-}last\text{-}n\ L$
⟨*proof*⟩

lemma *is-ladder-not-empty*: $is\text{-}ladder\ D\ L \Longrightarrow L \neq []$
⟨*proof*⟩

lemma *last-ladder-γ*:
assumes *is-ladder*: *is-ladder D L*
assumes *ladder-last-n*: $ladder\text{-}last\text{-}n\ L = length\ D$
shows $ladder\text{-}\gamma\ a\ D\ L\ (length\ L - Suc\ 0) = Derive\ a\ D$
⟨*proof*⟩

lemma *ladder-α-full*:
assumes *is-ladder*: *is-ladder D L*
assumes *ladder-last-n*: $ladder\text{-}last\text{-}n\ L = length\ D$

shows *ladder-*α *a* (*D* @ *D′*) (*L* @ *L′*) (*length L*) = *Derive a D*
⟨*proof*⟩

lemma *LeftDerivationIntro-implies-LeftDerivation*:
LeftDerivationIntro α *i r ix D j* $\gamma \Longrightarrow$ *LeftDerivation* α ((*i*,*r*)#*D*) γ
⟨*proof*⟩

lemma *LeftDerivationLadder-grow*:
LeftDerivationLadder a D L $\alpha \Longrightarrow$ *ladder-last-j L* = *i* $\Longrightarrow$
LeftDerivationIntro α *i r ix E j* $\gamma \Longrightarrow$
LeftDerivationLadder a (*D*@[(*i*, *r*)]@*E*) (*L*@[*mk-deriv-intro ix* (*Suc*(*length D* + *length E*)) *j*]) γ
⟨*proof*⟩

lemma *LeftDerivationIntro-bounds-ij*:
LeftDerivationIntro α *i r ix D j* $\beta \Longrightarrow$ *i* < *length* $\alpha \wedge$ *j* < *length* β
⟨*proof*⟩

theorem *LeftDerivationLadder-exists*: *LeftDerivation a D* $\gamma \Longrightarrow$ *is-sentence* $\gamma \Longrightarrow$ *j* < *length* $\gamma \Longrightarrow$
$\exists$ *L*. *LeftDerivationLadder a D L* $\gamma \wedge$ *ladder-last-j L* = *j*
⟨*proof*⟩

lemma *LeftDerivationLadder-L-0*:
assumes *LeftDerivationLadder* α *D L* β
assumes *length L* = *1*
shows $\exists$ *i*. *LeftDerivationFix* α *i D* (*ladder-last-j L*) β
⟨*proof*⟩

lemma *LeftDerivationFix-splits-at-derives*:
assumes *LeftDerivationFix a i D j b*
shows $\exists$ *U a1 a2 b1 b2*. *splits-at a i a1 U a2* $\wedge$ *splits-at b j b1 U b2* $\wedge$
derives a1 b1 $\wedge$ *derives a2 b2*
⟨*proof*⟩

lemma *LeftDerivation-append-suffix*:
LeftDerivation a D b $\Longrightarrow$ *is-sentence c* $\Longrightarrow$ *LeftDerivation* (*a*@*c*) *D* (*b*@*c*)
⟨*proof*⟩

lemma *LeftDerivation-impossible*: *LeftDerivation a D b* $\Longrightarrow$ *i* < *length a* $\Longrightarrow$
is-nonterminal (*a* ! *i*) $\Longrightarrow$ *derivation-ge D* (*Suc i*) $\Longrightarrow$ *D* = []
⟨*proof*⟩

lemma *derivation-ge-shift*: *derivation-ge* (*derivation-shift F 0 j*) *j*
⟨*proof*⟩

lemma *LeftDerivationFix-splits-at-nonterminal*:
assumes *LeftDerivationFix a i D j b*
assumes *is-nonterminal* (*a* ! *i*)

shows $\exists$ *U a1 a2 b1. splits-at a i a1 U a2* $\wedge$ *splits-at b j b1 U a2* $\wedge$ *LeftDerivation a1 D b1*
⟨*proof*⟩

lemma *LeftDerivationIntro-implies-nonterminal*:
LeftDerivationIntro α *i* (*snd e*) *ix E j* $\gamma \Longrightarrow$ *is-nonterminal* (α ! *i*)
⟨*proof*⟩

lemma *LeftDerivationIntrosAt-implies-nonterminal*:
LeftDerivationIntrosAt a D L index $\Longrightarrow$ *is-nonterminal*((*ladder-*α *a D L index*) ! (*ladder-i L index*))
⟨*proof*⟩

lemma *LeftDerivationIntro-examine-rule*:
LeftDerivationIntro α *i r ix D j* $\gamma \Longrightarrow$ *splits-at* α *i* α*1 M* α*2* $\Longrightarrow$
$\exists\ \eta.\ M = fst\ r \wedge \eta = snd\ r \wedge (M, \eta) \in \mathfrak{R}$
⟨*proof*⟩

lemma *LeftDerivation-skip-prefixword-ex*:
assumes *LeftDerivation* (*u*@*v*) *D w*
assumes *is-word u*
shows $\exists$ *w'*. *w* = *u*@*w'* $\wedge$ *LeftDerivation v* (*derivation-shift D* (*length u*) *0*) *w'*
⟨*proof*⟩

definition *ladder-cut* :: *ladder* $\Rightarrow$ *nat* $\Rightarrow$ *ladder*
where *ladder-cut L n* = (*let i* = *length L* − *1 in L*[*i* := (*n*, *snd* (*L* ! *i*))])

fun *deriv-shift* :: *nat* $\Rightarrow$ *nat* $\Rightarrow$ *deriv* $\Rightarrow$ *deriv*
where *deriv-shift dn dj* (*n*, *j*, *i*) = (*n* − *dn*, *j* − *dj*, *i*)

definition *ladder-shift* :: *ladder* $\Rightarrow$ *nat* $\Rightarrow$ *nat* $\Rightarrow$ *ladder*
where *ladder-shift L dn dj* = *map* (*deriv-shift dn dj*) *L*

lemma *splits-at-append-suffix-prevails*:
assumes *splits-at* (*a*@*b*) *i u N v*
assumes *i* < *length a*
shows $\exists$ *v'*. *v* = *v'*@*b* $\wedge$ *a*=*u*@[*N*]@*v'*
⟨*proof*⟩

lemma *derivation-shift-right-left-cancel*:
derivation-shift (*derivation-shift D 0 r*) *r 0* = *D*
⟨*proof*⟩

lemma *derivation-shift-left-right-cancel*:
assumes *derivation-ge D r*
shows *derivation-shift* (*derivation-shift D r 0*) *0 r* = *D*
⟨*proof*⟩

lemma *LeftDerivation-ge-take*:

assumes *derivation-ge D k*
assumes *LeftDerivation a D b*
assumes $D \neq []$
shows *take k a = take k b* $\wedge$ *is-word* (*take k a*)
⟨*proof*⟩

lemma *LeftDerivationFix-splits-at-symbol*:
assumes *LeftDerivationFix a i D j b*
shows $\exists$ *U a1 a2 b1 b2 n. splits-at a i a1 U a2* $\wedge$ *splits-at b j b1 U b2* $\wedge$
$n \leq$ *length D* $\wedge$ *LeftDerivation a1* (*take n D*) *b1* $\wedge$ *derivation-ge* (*drop n D*) (*Suc*(*length b1*)) $\wedge$
LeftDerivation a2 (*derivation-shift* (*drop n D*) (*Suc*(*length b1*)) *0*) *b2* $\wedge$
($n =$ *length D* $\vee$ ($n <$ *length D* $\wedge$ *is-word* (*b1*@[*U*])))
⟨*proof*⟩

lemma *LeftDerivation-breakdown′*: *LeftDerivation* (*u* @ *v*) *D w* $\Longrightarrow$
$\exists$ *n w1 w2*.
$n \leq$ *length D* $\wedge$
w = w1 @ *w2* $\wedge$
LeftDerivation u (*take n D*) *w1* $\wedge$
derivation-ge (*drop n D*) (*length w1*) $\wedge$
LeftDerivation v (*derivation-shift* (*drop n D*) (*length w1*) *0*) *w2*
⟨*proof*⟩

lemma *LeftDerives1-append-replace-in-left*:
assumes *ld1*: *LeftDerives1* (α@δ) *i r* β
assumes *i-bound*: $i <$ *length* α
shows $\exists$ α'. $\beta = \alpha'$@δ $\wedge$ *LeftDerives1* α *i r* α' $\wedge$ $i +$ *length* (*snd r*) $\leq$ *length* α'
⟨*proof*⟩

lemma *LeftDerivationIntro-propagate*:
assumes *intro*: *LeftDerivationIntro* (α@δ) *i r ix D j* γ
assumes *i-*α: $i <$ *length* α
assumes *non*: *is-nonterminal* (γ ! *j*)
shows $\exists$ ω. *LeftDerivation* α ((*i*,*r*)#*D*) ω $\wedge$ $\gamma = \omega$@δ $\wedge$ $j <$ *length* ω
⟨*proof*⟩

lemma *LeftDerivationIntro-finish*:
assumes *intro*: *LeftDerivationIntro* (α@δ) *i r ix D j* γ
assumes *i-*α: $i <$ *length* α
shows $\exists$ *k* ω δ'.
$k \leq$ *length D* $\wedge$
LeftDerivation α ((*i*, *r*)#(*take k D*)) ω $\wedge$
LeftDerivation (α @ δ) ((*i*, *r*)#(*take k D*)) (ω @ δ) $\wedge$
derivation-ge (*drop k D*) (*length* ω) $\wedge$
LeftDerivation δ (*derivation-shift* (*drop k D*) (*length* ω) *0*) δ' $\wedge$
$\gamma = \omega$ @ δ' $\wedge$ $j <$ *length* ω
⟨*proof*⟩

lemma *LeftDerivationLadder-propagate*:
LeftDerivationLadder $(\alpha@\delta)$ *D L* $\gamma \Longrightarrow$ *ladder-i L 0* $<$ *length* $\alpha \Longrightarrow n =$ *ladder-n L index*
$\Longrightarrow$ *index* $<$ *length L* $\Longrightarrow$
if (*index* + *1* < *length L*) *then*
($\exists\ \beta$. *LeftDerivation* α (*take n D*) $\beta \wedge$ *ladder-*γ $(\alpha@\delta)$ *D L index* $= \beta@\delta \wedge$
ladder-j L index < *length* β)
else
($\exists\ n'\ \beta\ \delta'$. (*index* = *0* $\vee$ *ladder-prev-n L index* $< n'$) $\wedge\ n' \leq n \wedge$ *LeftDerivation* α (*take* n' *D*) $\beta \wedge$
LeftDerivation $(\alpha@\delta)$ (*take* n' *D*) $(\beta@\delta) \wedge$
derivation-ge (*drop* n' *D*) (*length* β) $\wedge$
LeftDerivation δ (*derivation-shift* (*drop* n' *D*) (*length* β) *0*) $\delta' \wedge$
ladder-γ $(\alpha@\delta)$ *D L index* $= \beta@\delta' \wedge$ *ladder-j L index* < *length* β)
⟨*proof*⟩

lemma *ladder-i-of-cut-at-0*:
assumes *L-non-empty*: $L \neq []$
shows *ladder-i* (*ladder-cut L n*) *0* = *ladder-i L 0*
⟨*proof*⟩

lemma *ladder-last-j-of-cut*:
assumes *L-non-empty*: $L \neq []$
shows *ladder-last-j* (*ladder-cut L n*) = *ladder-last-j L*
⟨*proof*⟩

lemma *length-ladder-cut*:
assumes *L-non-empty*: $L \neq []$
shows *length* (*ladder-cut L n*) = *length L*
⟨*proof*⟩

lemma *ladder-last-n-of-cut*:
assumes *L-non-empty*: $L \neq []$
shows *ladder-last-n* (*ladder-cut L n*) = *n*
⟨*proof*⟩

lemma *ladder-n-of-cut*:
assumes *L-non-empty*: $L \neq []$
assumes *index* < *length L* − *1*
shows *ladder-n* (*ladder-cut L n*) *index* = *ladder-n L index*
⟨*proof*⟩

lemma *ladder-n-prev-bound*:
assumes *ladder*: *is-ladder D L*
assumes *u-bound*: *u* < *length L* − *1*
shows *ladder-n L u* $\leq$ *ladder-prev-n L* (*length L* − *1*)
⟨*proof*⟩

lemma *ladder-n-last-is-length*:

assumes *is-ladder D L*
shows *ladder-n L* (*length L* − *1*) = *length D*
⟨*proof*⟩

lemma *derivation-ge-shift-implies-derivation-ge*:
assumes *dge*: *derivation-ge* (*derivation-shift F 0 j*) *k*
shows *derivation-ge F* (*k* − *j*)
⟨*proof*⟩

lemma *Derives1-bound′*: *Derives1 a i r b* ⟹ *i* ≤ *length b*
⟨*proof*⟩

lemma *LeftDerivation-Derives1-last*:
assumes *LeftDerivation a D b*
assumes *D* ≠ []
shows *Derives1* (*Derive a* (*take* (*length D* − *1*) *D*)) (*fst* (*last D*)) (*snd* (*last D*)) *b*
⟨*proof*⟩

lemma *last-of-prefix-in-set*:
assumes *n* < *length E*
assumes *D* = *E*@*F*
shows *last E* ∈ *set* (*drop n D*)
⟨*proof*⟩

lemma *LeftDerivationFix-cut-appendix*:
assumes *ldfix*: *LeftDerivationFix* (*α*@*δ*) *i D j* (*β*@*δ′*)
assumes *α-β*: *LeftDerivation α* (*take n D*) *β*
assumes *n-bound*: *n* ≤ *length D*
assumes *dge*: *derivation-ge* (*drop n D*) (*length β*)
assumes *i-in*: *i* < *length α*
assumes *j-in*: *j* < *length β*
shows *LeftDerivationFix α i* (*take n D*) *j β*
⟨*proof*⟩

lemma *LeftDerivationFix-cut-appendix′*:
assumes *ldfix*: *LeftDerivationFix* (*α*@*δ*) *i D j* (*β*@*δ′*)
assumes *α-β*: *LeftDerivation α D β*
assumes *i-in*: *i* < *length α*
assumes *j-in*: *j* < *length β*
shows *LeftDerivationFix α i D j β*
⟨*proof*⟩

lemma *LeftDerivationIntro-cut-appendix*:
assumes *ldfix*: *LeftDerivationIntro* (*α*@*δ*) *i r ix D j* (*β*@*δ′*)
assumes *α-β*: *LeftDerivation α* ((*i*,*r*)#(*take n D*)) *β*
assumes *n-bound*: *n* ≤ *length D*
assumes *dge*: *derivation-ge* (*drop n D*) (*length β*)
assumes *i-in*: *i* < *length α*

assumes *j-in*: $j < length\ \beta$
shows *LeftDerivationIntro* $\alpha\ i\ r\ ix\ (take\ n\ D)\ j\ \beta$
⟨*proof*⟩

lemma *LeftDerivationIntro-cut-appendix'*:
assumes *ldfix*: *LeftDerivationIntro* $(\alpha @ \delta)\ i\ r\ ix\ D\ j\ (\beta @ \delta')$
assumes α-β: *LeftDerivation* $\alpha\ ((i,r)\#D)\ \beta$
assumes *i-in*: $i < length\ \alpha$
assumes *j-in*: $j < length\ \beta$
shows *LeftDerivationIntro* $\alpha\ i\ r\ ix\ D\ j\ \beta$
⟨*proof*⟩

lemma *ladder-n-monotone*: *is-ladder* $D\ L \Longrightarrow u \leq v \Longrightarrow v < length\ L \Longrightarrow$ *ladder-n* $L\ u \leq$ *ladder-n* $L\ v$
⟨*proof*⟩

lemma *ladder-i-cut*:
assumes *index-bound*: $index < length\ L$
shows *ladder-i* (*ladder-cut* $L\ n$) $index$ = *ladder-i* $L\ index$
⟨*proof*⟩

lemma *ladder-j-cut*:
assumes *index-bound*: $index < length\ L$
shows *ladder-j* (*ladder-cut* $L\ n$) $index$ = *ladder-j* $L\ index$
⟨*proof*⟩

lemma *ladder-ix-cut*:
assumes *index-lower-bound*: $index > 0$
assumes *index-upper-bound*: $index < length\ L$
shows *ladder-ix* (*ladder-cut* $L\ n$) $index$ = *ladder-ix* $L\ index$
⟨*proof*⟩

lemma *LeftDerivation-from-in-between*:
assumes α-β: *LeftDerivation* $\alpha\ (take\ u\ D)\ \beta$
assumes α-γ: *LeftDerivation* $\alpha\ (take\ v\ D)\ \gamma$
assumes *u-le-v*: $u \leq v$
shows *LeftDerivation* $\beta\ (drop\ u\ (take\ v\ D))\ \gamma$
⟨*proof*⟩

lemma *LeftDerivationLadder-cut-appendix-helper*:
assumes *LDLadder*: *LeftDerivationLadder* $(\alpha @ \delta)\ D\ L\ \gamma$
assumes *ladder-i-in-*α: *ladder-i* $L\ 0 < length\ \alpha$
shows $\exists\ E\ F\ \gamma 1\ \gamma 2\ L'.\ D = E @ F\ \wedge$
$\gamma = \gamma 1\ @\ \gamma 2\ \wedge$
LeftDerivationLadder $\alpha\ E\ L'\ \gamma 1\ \wedge$
derivation-ge $F\ (length\ \gamma 1)\ \wedge$
LeftDerivation δ (*derivation-shift* $F\ (length\ \gamma 1)\ 0)\ \gamma 2\ \wedge$
L' = *ladder-cut* $L\ (length\ E)$
⟨*proof*⟩

theorem *LeftDerivationLadder-cut-appendix*:
  assumes *LDLadder*: *LeftDerivationLadder* (α@δ) *D L* γ
  assumes *ladder-i-in-*α: *ladder-i L 0* < *length* α
  shows ∃ *E F* γ*1* γ*2* *L'*. *D* = *E*@*F* ∧
    γ = γ*1* @ γ*2* ∧
    LeftDerivationLadder α *E L'* γ*1* ∧
    derivation-ge F (*length* γ*1*) ∧
    LeftDerivation δ (*derivation-shift F* (*length* γ*1*) *0*) γ*2* ∧
    length L' = *length L* ∧ *ladder-i L' 0* = *ladder-i L 0* ∧
    ladder-last-j L' = *ladder-last-j L*
⟨*proof*⟩

definition *ladder-stepdown-diff* :: *ladder* ⇒ *nat* **where**
  ladder-stepdown-diff L = *Suc* (*ladder-n L 0*)

definition *ladder-stepdown-*α*-0* :: *sentence* ⇒ *derivation* ⇒ *ladder* ⇒ *sentence* **where**
  ladder-stepdown-α*-0 a D L* = *Derive a* (*take* (*ladder-stepdown-diff L*) *D*)

lemma *LeftDerivationIntro-LeftDerives1*:
  assumes *LeftDerivationIntro* α *i r ix D j* γ
  assumes *splits-at* α *i a1 A a2*
  shows *LeftDerives1* α *i r* (*a1*@(*snd r*)@*a2*)
⟨*proof*⟩

lemma *LeftDerives1-Derive*:
  assumes *LeftDerives1* α *i r* γ
  shows *Derive* α [(*i*, *r*)] = γ
⟨*proof*⟩

lemma *ladder-stepdown-*α*-0-altdef*:
  assumes *ladder*: *LeftDerivationLadder* α *D L* γ
  assumes *length-L*: *length L* > *1*
  assumes *split*: *splits-at* (*ladder-*α α *D L 1*) (*ladder-i L 1*) *a1 A a2*
  shows *ladder-stepdown-*α*-0* α *D L* = *a1* @ (*snd* (*snd* (*D* ! (*ladder-n L 0*)))) @ *a2*
⟨*proof*⟩

lemma *ladder-i-0-bound*:
  assumes *ld*: *LeftDerivationLadder* α *D L* γ
  shows *ladder-i L 0* < *length* α
⟨*proof*⟩

lemma *ladder-j-bound*:
  assumes *ld*: *LeftDerivationLadder* α *D L* γ
  assumes *index-bound*: *index* < *length L*
  shows *ladder-j L index* < *length* (*ladder-*γ α *D L index*)
⟨*proof*⟩

lemma *ladder-last-j-bound*:
assumes *ld*: *LeftDerivationLadder* α *D L* γ
shows *ladder-last-j L* $<$ *length* γ
⟨*proof*⟩

fun *ladder-shift-n* :: *nat* ⇒ *ladder* ⇒ *ladder* **where**
ladder-shift-n N [] = []
| *ladder-shift-n N* ((*n*, *j*, *i*)#*L*) = ((*n* − *N*, *j*, *i*)#(*ladder-shift-n N L*))

fun *ladder-stepdown* :: *ladder* ⇒ *ladder*
where
ladder-stepdown [] = *undefined*
| *ladder-stepdown* [*v*] = *undefined*
| *ladder-stepdown* ((*n0*, *j0*, *i0*)#(*n1*, *j1*, *ix1*)#*L*) =
(*n1* − *Suc n0*, *j1*, *j0* + *ix1*) # (*ladder-shift-n* (*Suc n0*) *L*)

lemma *ladder-shift-n-length*:
length (*ladder-shift-n N L*) = *length L*
⟨*proof*⟩

lemma *ladder-stepdown-prepare*:
assumes *length L* > *1*
shows *L* = (*ladder-n L 0*, *ladder-j L 0*, *ladder-i L 0*)#
(*ladder-n L 1*, *ladder-j L 1*, *ladder-ix L 1*)#(*drop 2 L*)
⟨*proof*⟩

lemma *ladder-stepdown-length*:
assumes *length L* > *1*
shows *length* (*ladder-stepdown L*) = *length L* − *1*
⟨*proof*⟩

lemma *ladder-stepdown-i-0*:
assumes *length L* > *1*
shows *ladder-i* (*ladder-stepdown L*) *0* = *ladder-i L 1* + *ladder-ix L 1*
⟨*proof*⟩

lemma *ladder-shift-n-cons*: *ladder-shift-n N* (*x*#*L*) = (*fst x* − *N*, *snd x*)#(*ladder-shift-n N L*)
⟨*proof*⟩

lemma *ladder-shift-n-drop*: *ladder-shift-n N* (*drop n L*) = *drop n* (*ladder-shift-n N L*)
⟨*proof*⟩

lemma *drop-2-shift*:
assumes *index* > *0*
assumes *length L* > *1*
shows *drop 2 L* ! (*index* − *Suc 0*) = *L* ! *Suc index*

⟨*proof*⟩

lemma *ladder-shift-n-at*:
index $<$ *length* $L \Longrightarrow$ (*ladder-shift-n* N L) ! *index* $=$ (*fst* (L ! *index*) $-$ N, *snd* (L ! *index*))
⟨*proof*⟩

lemma *ladder-stepdown-j*:
assumes *length-L-greater-1*: *length* $L > 1$
assumes L': $L' =$ *ladder-stepdown* L
assumes *index-bound*: *index* $<$ *length* L'
shows *ladder-j* L' *index* $=$ *ladder-j* L (*Suc index*)
⟨*proof*⟩

lemma *ladder-stepdown-last-j*:
assumes *length-L-greater-1*: *length* $L > 1$
shows *ladder-last-j* (*ladder-stepdown* L) $=$ *ladder-last-j* L
⟨*proof*⟩

lemma *ladder-stepdown-n*:
assumes *length-L-greater-1*: *length* $L > 1$
assumes L': $L' =$ *ladder-stepdown* L
assumes *index-bound*: *index* $<$ *length* L'
shows *ladder-n* L' *index* $=$ *ladder-n* L (*Suc index*) $-$ *ladder-stepdown-diff* L
⟨*proof*⟩

lemma *ladder-stepdown-ix*:
assumes *length-L-greater-1*: *length* $L > 1$
assumes L': $L' =$ *ladder-stepdown* L
assumes *index-lower-bound*: $0 <$ *index*
assumes *index-upper-bound*: *index* $<$ *length* L'
shows *ladder-ix* L' *index* $=$ *ladder-ix* L (*Suc index*)
⟨*proof*⟩

lemma *Derive-Derive*:
assumes *Derivation* α (D@E) γ
shows *Derive* (*Derive* α D) E $=$ *Derive* α (D@E)
⟨*proof*⟩

lemma *drop-at-shift*:
assumes $n \leq$ *index*
assumes *index* $<$ *length* D
shows *drop* n D ! (*index* $-$ n) $=$ D ! *index*
⟨*proof*⟩

theorem *LeftDerivationLadder-stepdown*:
assumes *ldl*: *LeftDerivationLadder* α D L γ
assumes *length-L*: *length* $L > 1$
shows $\exists$ L'. *LeftDerivationLadder* (*ladder-stepdown-α-0* α D L) (*drop* (*ladder-stepdown-diff*

L) *D*)
L′ γ ∧ *length L′* = *length L* − *1* ∧ *ladder-i L′ 0* = *ladder-i L 1* + *ladder-ix L 1* ∧
ladder-last-j L′ = *ladder-last-j L*
⟨*proof*⟩

fun *ladder-shift-j* :: *nat* ⇒ *ladder* ⇒ *ladder* **where**
ladder-shift-j d [] = []
| *ladder-shift-j d* ((*n*, *j*, *i*)#*L*) = ((*n*, *j* − *d*, *i*)#(*ladder-shift-j d L*))

definition *ladder-cut-prefix* :: *nat* ⇒ *ladder* ⇒ *ladder*
where
ladder-cut-prefix d L =
(*ladder-shift-j d L*)[*0* := (*ladder-n L 0*, *ladder-j L 0* − *d*, *ladder-i L 0* − *d*)]

lemma *ladder-shift-j-length*:
length (*ladder-shift-j d L*) = *length L*
⟨*proof*⟩

lemma *ladder-cut-prefix-length*:
shows *length* (*ladder-cut-prefix d L*) = *length L*
⟨*proof*⟩

lemma *ladder-shift-j-cons*: *ladder-shift-j d* (*x*#*L*) = (*fst x*, *fst* (*snd x*) − *d*, *snd*(*snd x*))#
(*ladder-shift-j d L*)
⟨*proof*⟩

lemma *deriv-j-ladder-shift-j*:
index < *length L* ⟹ *deriv-j* (*ladder-shift-j d L* ! *index*) = *deriv-j* (*L* ! *index*) − *d*
⟨*proof*⟩

lemma *deriv-n-ladder-shift-j*:
index < *length L* ⟹ *deriv-n* (*ladder-shift-j d L* ! *index*) = *deriv-n* (*L* ! *index*)
⟨*proof*⟩

lemma *deriv-ix-ladder-shift-j*:
index < *length L* ⟹ *deriv-ix* (*ladder-shift-j d L* ! *index*) = *deriv-ix* (*L* ! *index*)
⟨*proof*⟩

lemma *ladder-cut-prefix-j*:
assumes *index-bound*: *index* < *length L*
assumes *length-L*: *length L* > *0*
shows *ladder-j* (*ladder-cut-prefix d L*) *index* = *ladder-j L index* − *d*
⟨*proof*⟩

lemma *hd-0-subst*: *length L* > *0* ⟹ *hd* (*L* [*0* := *x*]) = *x*
⟨*proof*⟩

lemma *ladder-cut-prefix-i*:
assumes *index-bound*: *index* $<$ *length L*
assumes *length-L*: *length L* > 0
shows *ladder-i* (*ladder-cut-prefix d L*) *index* $=$ *ladder-i L index* $-$ *d*
⟨*proof*⟩

lemma *ladder-cut-prefix-n*:
assumes *index-bound*: *index* $<$ *length L*
assumes *length-L*: *length L* > 0
shows *ladder-n* (*ladder-cut-prefix d L*) *index* $=$ *ladder-n L index*
⟨*proof*⟩

lemma *ladder-cut-prefix-ix*:
assumes *index-bound*: *index* $<$ *length L*
assumes *length-L*: *length L* > 0
shows *ladder-ix* (*ladder-cut-prefix d L*) *index* $=$ *ladder-ix L index*
⟨*proof*⟩

lemma *LeftDerivationFix-derivation-ge-is-nonterminal*:
assumes *ldfix*: *LeftDerivationFix* α *i D j* γ
assumes *derivation-ge-d*: *derivation-ge D d*
assumes *is-nonterminal*: *is-nonterminal* (γ ! *j*)
shows ($D = [] \wedge \alpha = \gamma \wedge i = j) \vee (i > d \wedge j \geq d$)
⟨*proof*⟩

lemma *LeftDerivationFix-derivation-ge*:
assumes *ldfix*: *LeftDerivationFix* α *i D j* γ
assumes *derivation-ge-d*: *derivation-ge D d*
shows $i = j \vee (i > d \wedge j \geq d)$
⟨*proof*⟩

lemma *LeftDerivationIntro-derivation-ge*:
assumes *ldintro*: *LeftDerivationIntro* α *i r ix D j* γ
assumes *i-ge-d*: $i \geq d$
assumes *derivation-ge-d*: *derivation-ge D d*
shows $j \geq d$
⟨*proof*⟩

lemma *derivation-ge-LeftDerivationLadder*:
assumes *derivation-ge-d*: *derivation-ge D d*
assumes *ladder*: *LeftDerivationLadder* α *D L* γ
assumes *ladder-i-0*: *ladder-i L 0* $\geq d$
shows *index* $<$ *length L* $\Longrightarrow$ *ladder-i L index* $\geq d \wedge$ *ladder-j L index* $\geq d$
⟨*proof*⟩

lemma *derivation-shift-append*:
derivation-shift (*A@B*) *left right* =
(*derivation-shift A left right*) @ (*derivation-shift B left right*)

⟨*proof*⟩

lemma *derivation-shift-right-left-subtract*:
right $\geq$ *left* $\Longrightarrow$ *derivation-shift* (*derivation-shift L 0 right*) *left 0* =
derivation-shift L 0 (*right* − *left*)
⟨*proof*⟩

lemma *LeftDerivationFix-cut-prefix*:
assumes *LeftDerivationFix* (δ@α) *i D j* γ
assumes *derivation-ge D* (*length* δ)
assumes *i* $\geq$ *length* δ
assumes *is-word-*δ: *is-word* δ
shows $\exists\ \gamma'.\ \gamma = \delta$ @ $\gamma' \wedge$
LeftDerivationFix α (*i* − *length* δ) (*derivation-shift D* (*length* δ) *0*) (*j* − *length* δ) γ'
⟨*proof*⟩

lemma *LeftDerives1-propagate-prefix*:
LeftDerives1 (δ @ α) *i r* β $\Longrightarrow$ *i* $\geq$ *length* δ $\Longrightarrow$ *is-prefix* δ β
⟨*proof*⟩

lemma *LeftDerivationIntro-cut-prefix*:
assumes *LeftDerivationIntro* (δ@α) *i r ix D j* γ
assumes *derivation-ge D* (*length* δ)
assumes *i* $\geq$ *length* δ
assumes *is-word-*δ: *is-word* δ
shows $\exists\ \gamma'.\ \gamma = \delta$ @ $\gamma' \wedge$
LeftDerivationIntro α (*i* − *length* δ) *r ix* (*derivation-shift D* (*length* δ) *0*) (*j* − *length* δ) γ'
⟨*proof*⟩

lemma *LeftDerivationLadder-implies-LeftDerivation-at-index*:
assumes *LeftDerivationLadder* α *D L* γ
assumes *index* < *length L*
shows *LeftDerivation* α (*take* (*ladder-n L index*) *D*) (*ladder-*γ α *D L index*)
⟨*proof*⟩

lemma *LeftDerivationLadder-cut-prefix-propagate*:
assumes *ladder*: *LeftDerivationLadder* (δ@α) *D L* γ
assumes *is-word-*δ: *is-word* δ
assumes *derivation-ge-*δ: *derivation-ge D* (*length* δ)
assumes *ladder-i-0*: *ladder-i L 0* $\geq$ *length* δ
assumes L': L' = *ladder-cut-prefix* (*length* δ) *L*
assumes D': D' = *derivation-shift D* (*length* δ) *0*
shows *index* < *length L* $\Longrightarrow$
LeftDerivation α (*take* (*ladder-n* L' *index*) D') (*ladder-*γ α D' L' *index*) $\wedge$
ladder-α (δ@α) *D L index* = δ@(*ladder-*α α D' L' *index*) $\wedge$
ladder-γ (δ@α) *D L index* = δ@(*ladder-*γ α D' L' *index*)
⟨*proof*⟩

theorem *LeftDerivationLadder-cut-prefix*:
assumes *ladder*: *LeftDerivationLadder* $(\delta @ \alpha)$ *D L* γ
assumes *is-word-*δ: *is-word* δ
assumes *ladder-i-0*: *ladder-i L 0* $\geq$ *length* δ
shows $\exists\ D'\ L'\ \gamma'.\ \gamma = \delta\ @\ \gamma' \wedge$
LeftDerivationLadder $\alpha\ D'\ L'\ \gamma' \wedge$
D' = *derivation-shift D* (*length* δ) *0* $\wedge$
length L' = *length L* $\wedge$ *ladder-i* L' *0* + *length* δ = *ladder-i L 0* $\wedge$
ladder-last-j L' + *length* δ = *ladder-last-j L*
⟨*proof*⟩

end

end
theory *TheoremD10*
imports *TheoremD9 Ladder*
begin

context *LocalLexing* **begin**

lemma $\mathcal{P}$*-wellformed*: $p \in \mathcal{P}\ k\ u \Longrightarrow$ *wellformed-tokens p*
⟨*proof*⟩

lemma $\mathcal{X}$*-token-length*: $t \in \mathcal{X}\ k \Longrightarrow k$ + *length* (*chars-of-token t*) $\leq$ *length Doc*
⟨*proof*⟩

lemma *mono-Scan*: *mono* (*Scan T k*)
⟨*proof*⟩

lemma π*-apply-setmonotone*: $x \in I \Longrightarrow x \in \pi\ k\ T\ I$
⟨*proof*⟩

lemma *Scan-apply-setmonotone*: $x \in I \Longrightarrow x \in$ *Scan T k I*
⟨*proof*⟩

lemma *leftderives-padfront*:
assumes *leftderives* $\alpha\ \beta$
assumes *is-word u*
shows *leftderives* $(u @ \alpha)\ (u @ \beta)$
⟨*proof*⟩

lemma *leftderives-padback*:
assumes *leftderives* $\alpha\ \beta$
assumes *is-sentence u*
shows *leftderives* $(\alpha @ u)\ (\beta @ u)$
⟨*proof*⟩

lemma *leftderives-pad*:
  assumes α-β: *leftderives* α β
  assumes *is-word*: *is-word* u
  assumes *is-sentence*: *is-sentence* v
  shows *leftderives* $(u@\alpha@v)$ $(u@\beta@v)$
⟨*proof*⟩

lemma *leftderives-rule*:
  assumes $(N, w) \in \mathfrak{R}$
  shows *leftderives* $[N]$ w
⟨*proof*⟩

lemma *leftderives-rule-step*:
  assumes *ld*: *leftderives* a $(u@[N]@v)$
  assumes *rule*: $(N, w) \in \mathfrak{R}$
  assumes *is-word*: *is-word* u
  assumes *is-sentence*: *is-sentence* v
  shows *leftderives* a $(u@w@v)$
⟨*proof*⟩

lemma *leftderives-trans-step*:
  assumes *ld*: *leftderives* a $(u@b@v)$
  assumes *rule*: *leftderives* b c
  assumes *is-word*: *is-word* u
  assumes *is-sentence*: *is-sentence* v
  shows *leftderives* a $(u@c@v)$
⟨*proof*⟩

lemma *charslength-of-prefix*:
  assumes *is-prefix* a b
  shows *charslength* $a \leq$ *charslength* b
⟨*proof*⟩

lemma *item-rhs-simp*[*simp*]: *item-rhs* (*Item* (N, α) d i j) $= \alpha$
  ⟨*proof*⟩

definition *Prefixes* :: *'a list* $\Rightarrow$ *'a list set*
where
  Prefixes $p = \{\ q$. *is-prefix* q p $\}$

lemma $\mathfrak{P}$-*wellformed*: $p \in \mathfrak{P} \Longrightarrow$ *wellformed-tokens* p
  ⟨*proof*⟩

lemma *Prefixes-reflexive*[*simp*]: $p \in$ *Prefixes* p
  ⟨*proof*⟩

lemma *Prefixes-is-prefix*: $q \in$ *Prefixes* $p =$ *is-prefix* q p
  ⟨*proof*⟩

lemma *prefixes-are-paths'*: $p \in \mathfrak{P} \Longrightarrow$ *is-prefix* $q\ p \Longrightarrow q \in \mathfrak{P}$
⟨*proof*⟩

lemma *thmD10-ladder*:
$p \in \mathfrak{P} \Longrightarrow$
charslength $p = k \Longrightarrow$
$X \in T \Longrightarrow$
$T \subseteq \mathcal{X}\ k \Longrightarrow$
$(N, \alpha @ \beta) \in \mathfrak{R} \Longrightarrow$
$r \leq$ *length* $p \Longrightarrow$
leftderives $[\mathfrak{S}]$ ((*terminals* (*take* $r\ p$))@$[N]$@γ) $\Longrightarrow$
LeftDerivationLadder $\alpha\ D\ L$ (*terminals* ((*drop* $r\ p$)@$[X]$)) $\Longrightarrow$
ladder-last-j $L =$ *length* (*drop* $r\ p$) $\Longrightarrow$
$k' = k +$ *length* (*chars-of-token* X) $\Longrightarrow$
$x =$ *Item* $(N, \alpha @ \beta)$ (*length* α) (*charslength* (*take* $r\ p$)) $k' \Longrightarrow$
$I =$ *items-le* k' ($\pi\ k'$ {} (*Scan* $T\ k$ (*Gen* (*Prefixes* p))))
$\Longrightarrow x \in I$
⟨*proof*⟩

theorem *thmD10*:
assumes *p-dom*: $p \in \mathfrak{P}$
assumes *p-charslength*: *charslength* $p = k$
assumes *X-dom*: $X \in T$
assumes *T-dom*: $T \subseteq \mathcal{X}\ k$
assumes *rule-dom*: $(N, \alpha @ \beta) \in \mathfrak{R}$
assumes *r*: $r \leq$ *length* p
assumes *leftderives-start*: *leftderives* $[\mathfrak{S}]$ ((*terminals* (*take* $r\ p$))@$[N]$@γ)
assumes *leftderives-α*: *leftderives* α (*terminals* ((*drop* $r\ p$)@$[X]$))
assumes *k'*: $k' = k +$ *length* (*chars-of-token* X)
assumes *item-def*: $x =$ *Item* $(N, \alpha @ \beta)$ (*length* α) (*charslength* (*take* $r\ p$)) k'
assumes *I*: $I =$ *items-le* k' ($\pi\ k'$ {} (*Scan* $T\ k$ (*Gen* (*Prefixes* p))))
shows $x \in I$
⟨*proof*⟩

end

end
theory *TheoremD11*
imports *TheoremD10*
begin

context *LocalLexing* **begin**

lemma *LeftDerivationLadder-length-1*:
assumes *ladder*: *LeftDerivationLadder* $\alpha\ D\ L\ \gamma$
assumes *singleton-L*: *length* $L = 1$
shows *LeftDerivationFix* α (*ladder-i* L *0*) D (*ladder-last-j* L) γ
⟨*proof*⟩

lemma *LeftDerivationFix-from-singleton-helper*:
  assumes *LeftDerivationFix* $[A]$ *0* D (*length* u) (u @ $[B]$ @ v)
  shows $D = []$
⟨*proof*⟩

lemma *LeftDerivationFix-from-singleton*:
  assumes *LeftDerivationFix* $[A]$ i D j γ
  shows $D = []$
⟨*proof*⟩

lemma *LeftDerivationLadder-ladder-γ-last*:
  assumes *LeftDerivationLadder* α D L γ
  shows γ = *ladder-γ* α D L (*length* L − *1*)
⟨*proof*⟩

theorem *thmD11-helper*:
  $p \in \mathfrak{P} \Longrightarrow$
   charslength $p = k \Longrightarrow$
   $X \in T \Longrightarrow$
   $T \subseteq \mathcal{X}\ k \Longrightarrow$
   $q = p$ @ $[X] \Longrightarrow$
   $(N, \alpha@\beta) \in \mathfrak{R} \Longrightarrow$
   $r \leq$ *length* $q \Longrightarrow$
   LeftDerivation $[\mathfrak{S}]$ D ((*terminals* (*take* r q))@$[N]$@γ) $\Longrightarrow$
   leftderives α (*terminals* (*drop* r q)) $\Longrightarrow$
   $k' = k$ + *length* (*chars-of-token* X) $\Longrightarrow$
   x = *Item* $(N, \alpha@\beta)$ (*length* α) (*charslength* (*take* r q)) $k' \Longrightarrow$
   I = *items-le* k' (π k' {} (*Scan* T k (*Gen* (*Prefixes* p)))) $\Longrightarrow$
   $x \in I$
⟨*proof*⟩

theorem *thmD11*:
  assumes *p-dom*: $p \in \mathfrak{P}$
  assumes *p-charslength*: *charslength* $p = k$
  assumes *X-dom*: $X \in T$
  assumes *T-dom*: $T \subseteq \mathcal{X}\ k$
  assumes *q-def*: $q = p$ @ $[X]$
  assumes *rule-dom*: $(N, \alpha@\beta) \in \mathfrak{R}$
  assumes *r*: $r \leq$ *length* q
  assumes *leftderives-start*: *leftderives* $[\mathfrak{S}]$ ((*terminals* (*take* r q))@$[N]$@γ)
  assumes *leftderives-α*: *leftderives* α (*terminals* (*drop* r q))
  assumes *k'*: $k' = k$ + *length* (*chars-of-token* X)
  assumes *item-def*: x = *Item* $(N, \alpha@\beta)$ (*length* α) (*charslength* (*take* r q)) k'
  assumes *I*: I = *items-le* k' (π k' {} (*Scan* T k (*Gen* (*Prefixes* p))))
  shows $x \in I$
⟨*proof*⟩

end

end
theory *TheoremD12*
imports *TheoremD11*
begin

context *LocalLexing* **begin**

lemma *charslength-appendix-is-empty*:
  $charslength\ (p@ts) = charslength\ p \Longrightarrow (\bigwedge\ t.\ t \in set\ ts \Longrightarrow chars\text{-}of\text{-}token\ t = [])$
⟨*proof*⟩

lemma *empty-tokens-have-charslength-0*:
  $(\bigwedge\ t.\ t \in set\ ts \Longrightarrow chars\text{-}of\text{-}token\ t = []) \Longrightarrow charslength\ ts = 0$
⟨*proof*⟩

lemma *π-idempotent'*: $\pi\ k\ \{\}\ (\pi\ k\ T\ I) = \pi\ k\ T\ I$
  ⟨*proof*⟩

theorem *thmD12*:
  assumes *induct*: $items\text{-}le\ k\ (\mathcal{J}\ k\ u) = Gen\ (paths\text{-}le\ k\ (\mathcal{P}\ k\ u))$
  assumes *induct-tokens*: $\mathcal{T}\ k\ u = \mathcal{Z}\ k\ u$
  shows $items\text{-}le\ k\ (\mathcal{J}\ k\ (Suc\ u)) \supseteq Gen\ (paths\text{-}le\ k\ (\mathcal{P}\ k\ (Suc\ u)))$
⟨*proof*⟩

end

end
theory *TheoremD13*
imports *TheoremD12*
begin

context *LocalLexing* **begin**

lemma *pointwise-natUnion-swap*:
  assumes *pointwise-f*: *pointwise f*
  shows $f\ (natUnion\ G) = natUnion\ (\lambda\ u.\ f\ (G\ u))$
⟨*proof*⟩

lemma *pointwise-Gen*: *pointwise Gen*
  ⟨*proof*⟩

lemma *thmD13-part1*:
  assumes *start*: $items\text{-}le\ k\ (\mathcal{J}\ k\ 0) = Gen\ (paths\text{-}le\ k\ (\mathcal{P}\ k\ 0))$
  assumes *valid-k*: $k \leq length\ Doc$
  shows $items\text{-}le\ k\ (\mathcal{J}\ k\ u) = Gen\ (paths\text{-}le\ k\ (\mathcal{P}\ k\ u)) \wedge \mathcal{T}\ k\ u = \mathcal{Z}\ k\ u$
⟨*proof*⟩

lemma *thmD13-part2*:

assumes *start*: *items-le k* ($\mathcal{J}$ *k 0*) = *Gen* (*paths-le k* ($\mathcal{P}$ *k 0*))
assumes *valid-k*: *k* ≤ *length Doc*
shows *items-le k* ($\mathcal{I}$ *k*) = *Gen* (*paths-le k* ($\mathcal{Q}$ *k*))
⟨*proof*⟩

theorem *thmD13*:
assumes *start*: *items-le k* ($\mathcal{J}$ *k 0*) = *Gen* (*paths-le k* ($\mathcal{P}$ *k 0*))
assumes *valid-k*: *k* ≤ *length Doc*
shows *items-le k* ($\mathcal{J}$ *k u*) = *Gen* (*paths-le k* ($\mathcal{P}$ *k u*)) ∧ $\mathcal{T}$ *k u* = $\mathcal{Z}$ *k u*
∧ *items-le k* ($\mathcal{I}$ *k*) = *Gen* (*paths-le k* ($\mathcal{Q}$ *k*))
⟨*proof*⟩

end

end
theory *TheoremD14*
imports *TheoremD13*
begin

context *LocalLexing* **begin**

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

lemma *items-le-split-via-eq*: *items-le* (*Suc k*) *J* = *items-le k J* ∪ *items-eq* (*Suc k*) *J*
⟨*proof*⟩

lemma *paths-le-split-via-eq*: *paths-le* (*Suc k*) *P* = *paths-le k P* ∪ *paths-eq* (*Suc k*) *P*
⟨*proof*⟩

lemma *natUnion-superset*:
shows *g i* ⊆ *natUnion g*
⟨*proof*⟩

definition *indexle* :: *nat* ⇒ *nat* ⇒ *nat* ⇒ *nat* ⇒ *bool* **where**
indexle k′ u′ k u = ((*indexlt k′ u′ k u*) ∨ (*k′* = *k* ∧ *u′* = *u*))

definition *produced-by-scan-step* :: *item* ⇒ *nat* ⇒ *nat* ⇒ *bool* **where**
produced-by-scan-step x k u = (∃ *k′ u′ y X*. *indexle k′ u′ k u* ∧ *y* ∈ $\mathcal{J}$ *k′ u′* ∧
item-end y = *k′* ∧ *X* ∈ ($\mathcal{T}$ *k′ u′*) ∧ *x* = *inc-item y* (*k′* + *length* (*chars-of-token X*)) ∧
next-symbol y = *Some* (*terminal-of-token X*))

lemma *indexle-trans*: *indexle k″ u″ k′ u′* ⟹ *indexle k′ u′ k u* ⟹ *indexle k″ u″ k u*
⟨*proof*⟩

lemma *produced-by-scan-step-trans*:
assumes $indexle\ k'\ u'\ k\ u$
assumes $produced\text{-}by\text{-}scan\text{-}step\ x\ k'\ u'$
shows $produced\text{-}by\text{-}scan\text{-}step\ x\ k\ u$
⟨*proof*⟩

lemma $\mathcal{J}$*-induct*[*consumes 1*, *case-names Induct*]:
assumes $x \in \mathcal{J}\ k\ u$
assumes *induct*: $\bigwedge\ x\ k\ u\ .\ (\bigwedge\ x'\ k'\ u'.\ x' \in \mathcal{J}\ k'\ u' \Longrightarrow indexlt\ k'\ u'\ k\ u \Longrightarrow P\ x'\ k'\ u')$
$\Longrightarrow x \in \mathcal{J}\ k\ u \Longrightarrow P\ x\ k\ u$
shows $P\ x\ k\ u$
⟨*proof*⟩

lemma π*-no-tokens-item-end*:
assumes *x-in-*π: $x \in \pi\ k\ \{\}\ I$
shows $item\text{-}end\ x = k \lor x \in I$
⟨*proof*⟩

lemma *natUnion-ex*: $x \in natUnion\ f \Longrightarrow \exists\ i.\ x \in f\ i$
⟨*proof*⟩

lemma *locate-in-limit*:
assumes *x-in-limit*: $x \in limit\ f\ X$
assumes *x-notin-X*: $x \notin X$
shows $\exists\ n.\ x \in funpower\ f\ (Suc\ n)\ X \land x \notin funpower\ f\ n\ X$
⟨*proof*⟩

lemma *produced-by-scan-step*:
$x \in \mathcal{J}\ k\ u \Longrightarrow item\text{-}end\ x > k \Longrightarrow produced\text{-}by\text{-}scan\text{-}step\ x\ k\ u$
⟨*proof*⟩

lemma *limit-single-step*:
assumes $x \in f\ X$
shows $x \in limit\ f\ X$
⟨*proof*⟩

lemma *Gen-union*: $Gen\ (A \cup B) = Gen\ A \cup Gen\ B$
⟨*proof*⟩

lemma *is-prefix-Prefixes-subset*:
assumes $is\text{-}prefix\ q\ p$
shows $Prefixes\ q \subseteq Prefixes\ p$
⟨*proof*⟩

lemma *Prefixes-subset-*$\mathcal{P}$:
assumes $p \in \mathcal{P}\ k\ u$
shows $Prefixes\ p \subseteq \mathcal{P}\ k\ u$
⟨*proof*⟩

lemma *Prefixes-subset-paths-le*:
assumes $\mathit{Prefixes}\ p \subseteq P$
shows $\mathit{Prefixes}\ p \subseteq \mathit{paths\text{-}le}\ (\mathit{charslength}\ p)\ P$
⟨*proof*⟩

lemma *Scan-$\mathcal{J}$-subset-$\mathcal{J}$*:
$\mathit{Scan}\ (\mathcal{T}\ k\ (\mathit{Suc}\ u))\ k\ (\mathcal{J}\ k\ u) \subseteq \mathcal{J}\ k\ (\mathit{Suc}\ u)$
⟨*proof*⟩

lemma *subset-$\mathcal{J}$k*: $u \leq v \Longrightarrow \mathcal{J}\ k\ u \subseteq \mathcal{J}\ k\ v$
thm *$\mathcal{J}$-subset-Suc-u*
⟨*proof*⟩

lemma *subset-$\mathcal{J}\mathcal{I}$k*: $\mathcal{J}\ k\ u \subseteq \mathcal{I}\ k$ ⟨*proof*⟩

lemma *subset-$\mathcal{I}\mathcal{J}$Suc*: $\mathcal{I}\ k \subseteq \mathcal{J}\ (\mathit{Suc}\ k)\ u$
⟨*proof*⟩

lemma *subset-$\mathcal{I}$Suc*: $\mathcal{I}\ k \subseteq \mathcal{I}\ (\mathit{Suc}\ k)$
⟨*proof*⟩

lemma *subset-$\mathcal{I}$*: $i \leq j \Longrightarrow \mathcal{I}\ i \subseteq \mathcal{I}\ j$
⟨*proof*⟩

lemma *subset-$\mathcal{J}$* :
assumes *leq*: $k' < k \vee (k' = k \wedge u' \leq u)$
shows $\mathcal{J}\ k'\ u' \subseteq \mathcal{J}\ k\ u$
⟨*proof*⟩

lemma *$\mathcal{J}$-subset*:
assumes $\mathit{indexle}\ k'\ u'\ k\ u$
shows $\mathcal{J}\ k'\ u' \subseteq \mathcal{J}\ k\ u$
⟨*proof*⟩

lemma *Scan-items-le*:
assumes *bounded-T*: $\bigwedge\ t\ .\ t \in T \Longrightarrow \mathit{length}\ (\mathit{chars\text{-}of\text{-}token}\ t) \leq l$
shows $\mathit{Scan}\ T\ k\ (\mathit{items\text{-}le}\ k\ P) \subseteq \mathit{items\text{-}le}\ (k + l)\ (\mathit{Scan}\ T\ k\ P)$
⟨*proof*⟩

lemma *Scan-mono-tokens*:
$P \subseteq Q \Longrightarrow \mathit{Scan}\ P\ k\ I \subseteq \mathit{Scan}\ Q\ k\ I$
⟨*proof*⟩

theorem *thmD14*: $k \leq \mathit{length}\ \mathit{Doc} \Longrightarrow \mathit{items\text{-}le}\ k\ (\mathcal{J}\ k\ u) = \mathit{Gen}\ (\mathit{paths\text{-}le}\ k\ (\mathcal{P}\ k\ u)) \wedge \mathcal{T}\ k\ u = \mathcal{Z}\ k\ u$
$\wedge\ \mathit{items\text{-}le}\ k\ (\mathcal{I}\ k) = \mathit{Gen}\ (\mathit{paths\text{-}le}\ k\ (\mathcal{Q}\ k))$
⟨*proof*⟩

end

end
theory *PathLemmas*
imports *TheoremD14*
begin

context *LocalLexing* **begin**

lemma *characterize-*$\mathcal{P}$:
$(\forall\ i < length\ p.\ \exists u.\ p\ !\ i \in \mathcal{Z}\ (charslength\ (take\ i\ p))\ u) \Longrightarrow admissible\ p \Longrightarrow$
$\exists\ u.\ p \in \mathcal{P}\ (charslength\ p)\ u$
⟨*proof*⟩

lemma *drop-empty-tokens*:
assumes *p*: $p \in \mathfrak{P}$
assumes *r*: $r \leq length\ p$
assumes *empty*: $charslength\ (take\ r\ p) = 0$
assumes *admissible*: *admissible* (*drop r p*)
shows $drop\ r\ p \in \mathfrak{P}$
⟨*proof*⟩

end

end
theory *MainTheorems*
imports *PathLemmas*
begin

context *LocalLexing* **begin**

theorem $\mathfrak{I}$*-is-generated-by-*$\mathfrak{P}$: $\mathfrak{I} = Gen\ \mathfrak{P}$
⟨*proof*⟩

definition *finished-item* :: *symbol list* ⇒ *item*
where
finished-item α = *Item* $(\mathfrak{S}, \alpha)$ (*length* α) *0* (*length Doc*)

lemma *item-rule-finished-item*[*simp*]: *item-rule* (*finished-item* α) = $(\mathfrak{S}, \alpha)$
⟨*proof*⟩

lemma *item-origin-finished-item*[*simp*]: *item-origin* (*finished-item* α) = *0*
⟨*proof*⟩

lemma *item-end-finished-item*[*simp*]: *item-end* (*finished-item* α) = *length Doc*
⟨*proof*⟩

lemma *item-dot-finished-item*[*simp*]: *item-dot* (*finished-item* α) = *length* α
⟨*proof*⟩

lemma *item-rhs-finished-item*[*simp*]: *item-rhs* (*finished-item* α) = α
⟨*proof*⟩

lemma *item-α-finished-item*[*simp*]: *item-α* (*finished-item* α) = α
⟨*proof*⟩

lemma *item-nonterminal-finished-item*[*simp*]: *item-nonterminal* (*finished-item* α) = $\mathfrak{S}$
⟨*proof*⟩

lemma *Derives1-of-singleton*:
assumes *Derives1* [*N*] *i r* α
shows $i = 0 \wedge r = (N, \alpha)$
⟨*proof*⟩

definition *pvalid-with* :: *tokens* ⇒ *item* ⇒ *nat* ⇒ *symbol list* ⇒ *bool*
where
pvalid-with p x u γ =
(*wellformed-tokens p* ∧
wellformed-item x ∧
$u \leq$ *length p* ∧
charslength p = *item-end x* ∧
charslength (*take u p*) = *item-origin x* ∧
is-derivation (*terminals* (*take u p*) @ [*item-nonterminal x*] @ γ) ∧
derives (*item-α x*) (*terminals* (*drop u p*)))

lemma *pvalid-with*: *pvalid p x* = (∃ *u* γ. *pvalid-with p x u* γ)
⟨*proof*⟩

theorem *Completeness*:
assumes *p-in-ll*: *p* ∈ *ll*
shows ∃ α. *pvalid-with p* (*finished-item* α) *0* [] ∧ *finished-item* $\alpha \in \mathfrak{I}$
⟨*proof*⟩

theorem *Soundness*:
assumes *finished-item-α*: *finished-item* $\alpha \in \mathfrak{I}$
shows ∃ *p*. *pvalid-with p* (*finished-item* α) *0* [] ∧ *p* ∈ *ll*
⟨*proof*⟩

lemma *is-finished-and-finished-item*:
assumes *wellformed-x*: *wellformed-item x*
shows *is-finished x* = (∃ α. *x* = *finished-item* α)
⟨*proof*⟩

theorem *Correctness*:
shows (*ll* ≠ {}) = *earley-recognised*
⟨*proof*⟩

end

end