

Earley

Martin Rau

February 6, 2026

Abstract

In 1968 Earley [1] introduced his parsing algorithm capable of parsing all context-free grammars in cubic space and time. This entry contains a formalization of an executable Earley parser. We base our development on Jones' [2] extensive paper proof of Earley's recognizer and the formalization of context-free grammars and derivations of Obua [4] [3]. We implement and prove correct a functional recognizer modeling Earley's original imperative implementation and extend it with the necessary data structures to enable the construction of parse trees following the work of Scott [5]. We then develop a functional algorithm that builds a single parse tree and prove its correctness. Finally, we generalize this approach to an algorithm for a complete parse forest and prove soundness.

Contents

1	Slightly adjusted content from AFP/LocalLexing	2
2	Adjusted content from AFP/LocalLexing	5
3	Adjusted content from AFP/LocalLexing	6
4	Additional derivation lemmas	7
5	Slices	8
6	Earley recognizer	9
6.1	Earley items	9
6.2	Well-formedness	11
6.3	Soundness	11
6.4	Completeness	12
6.5	Correctness	13
6.6	Finiteness	13

7	Earley fixpoint	13
7.1	Definitions	13
7.2	Monotonicity and Absorption	15
7.3	Soundness	17
7.4	Completeness	17
7.5	Correctness	18
8	Earley recognizer	19
8.1	List auxiliaries	19
8.2	Definitions	20
8.3	Epsilon productions	22
8.4	Bin lemmas	23
8.5	Well-formed bins	26
8.6	Soundness	30
8.7	Completeness	32
8.8	Correctness	35
9	Earley parser	35
9.1	Pointer lemmas	35
9.2	Common Definitions	38
9.3	foldl lemmas	38
9.4	Parse tree	39
10	Examples	42
10.1	Common symbols	42
10.2	$O(n^3)$ ambiguous grammars	42
10.2.1	$S \rightarrow SS \mid a$	42
10.3	$O(n^2)$ unambiguous or bounded ambiguity	42
10.3.1	$S \rightarrow aS \mid a$	42
10.3.2	$S \rightarrow aSa \mid a$	43
10.4	$O(n)$ bounded state, non-right recursive LR(k) grammars	43
10.4.1	$S \rightarrow Sa \mid a$	43
10.5	$S \rightarrow SX, X \rightarrow Y \mid Z, Y \rightarrow a, Z \rightarrow a$	43
11	Input and Evaluation	44
theory <i>Limit</i>		
imports		
<i>Main</i>		
begin		

1 Slightly adjusted content from AFP/LocalLexing

```
fun funpower :: ('a  $\Rightarrow$  'a)  $\Rightarrow$  nat  $\Rightarrow$  ('a  $\Rightarrow$  'a) where
  funpower f 0 x = x
```

| $\text{funpower } f \text{ (Suc } n) x = f \text{ (funpower } f \text{ } n x)$

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

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

definition $\text{setmonotone} :: ('a \text{ set} \Rightarrow 'a \text{ set}) \Rightarrow \text{bool}$ **where**
 $\text{setmonotone } f = (\forall X. X \subseteq f X)$

lemma $\text{subset-setmonotone}: \text{setmonotone } f \Longrightarrow X \subseteq f X$
{proof}

lemma[simp]: $\text{funpower } id \text{ } n = id$
{proof}

lemma[simp]: $\text{limit } id = id$
{proof}

definition $\text{chain} :: (\text{nat} \Rightarrow 'a \text{ set}) \Rightarrow \text{bool}$
where
 $\text{chain } C = (\forall i. C \text{ } i \subseteq C \text{ } (i + 1))$

definition $\text{continuous} :: ('a \text{ set} \Rightarrow 'b \text{ set}) \Rightarrow \text{bool}$
where
 $\text{continuous } f = (\forall C. \text{chain } C \longrightarrow (\text{chain } (f \circ C) \wedge f \text{ (natUnion } C) = \text{natUnion } (f \circ C)))$

lemma $\text{natUnion-upperbound}$:
 $(\bigwedge n. f \text{ } n \subseteq G) \Longrightarrow (\text{natUnion } f) \subseteq G$
{proof}

lemma $\text{funpower-upperbound}$:
 $(\bigwedge I. I \subseteq G \Longrightarrow f I \subseteq G) \Longrightarrow I \subseteq G \Longrightarrow \text{funpower } f \text{ } n I \subseteq G$
{proof}

lemma limit-upperbound :
 $(\bigwedge I. I \subseteq G \Longrightarrow f I \subseteq G) \Longrightarrow I \subseteq G \Longrightarrow \text{limit } f I \subseteq G$
{proof}

lemma $\text{elem-limit-simp}: x \in \text{limit } f X = (\exists n. x \in \text{funpower } f \text{ } n X)$
{proof}

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

lemma $\text{natUnion-elem}: x \in f n \Longrightarrow x \in \text{natUnion } f$
{proof}

lemma *limit-elim*: $x \in \text{funpower } f \ n \ X \implies x \in \text{limit } f \ X$
<proof>

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

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

lemma *chain-implies-mono*: $\text{chain } C \implies \text{mono } C$
<proof>

lemma *setmonotone-implies-chain-funpower*:
assumes *setmonotone*: $\text{setmonotone } f$
shows $\text{chain } (\lambda n. \text{funpower } f \ n \ I)$
<proof>

lemma *natUnion-subset*: $(\bigwedge n. \exists m. f \ n \subseteq g \ m) \implies \text{natUnion } f \subseteq \text{natUnion } g$
<proof>

lemma *natUnion-eq*[*case-names Subset Superset*]:
 $(\bigwedge n. \exists m. f \ n \subseteq g \ m) \implies (\bigwedge n. \exists m. g \ n \subseteq f \ m) \implies \text{natUnion } f = \text{natUnion } g$
<proof>

lemma *natUnion-shift*[*symmetric*]:
assumes *chain*: $\text{chain } C$
shows $\text{natUnion } C = \text{natUnion } (\lambda n. C \ (n + m))$
<proof>

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

lemma *regular-fixpoint*:
assumes *regular*: $\text{regular } f$
shows $f \ (\text{limit } f \ I) = \text{limit } f \ I$
<proof>

lemma *fix-is-fix-of-limit*:
assumes *fixpoint*: $f \ I = I$
shows $\text{limit } f \ I = I$
<proof>

lemma *limit-is-idempotent*: $\text{regular } f \implies \text{limit } f \ (\text{limit } f \ I) = \text{limit } f \ I$
<proof>

definition *mk-regular1* :: $('b \Rightarrow 'a \Rightarrow \text{bool}) \Rightarrow ('b \Rightarrow 'a \Rightarrow 'a) \Rightarrow 'a \text{ set} \Rightarrow 'a \text{ set}$

where

$mk\text{-regular1 } P F I = I \cup \{ F q x \mid q x. x \in I \wedge P q x \}$

definition $mk\text{-regular2} :: ('b \Rightarrow 'a \Rightarrow 'a \Rightarrow bool) \Rightarrow ('b \Rightarrow 'a \Rightarrow 'a \Rightarrow 'a) \Rightarrow 'a \text{ set} \Rightarrow 'a \text{ set}$ **where**

$mk\text{-regular2 } P F I = I \cup \{ F q x y \mid q x y. x \in I \wedge y \in I \wedge P q x y \}$

end

theory *CFG*

imports *Main*

begin

2 Adjusted content from AFP/LocalLexing

type-synonym $'a \text{ rule} = 'a \times 'a \text{ list}$

type-synonym $'a \text{ rules} = 'a \text{ rule list}$

datatype $'a \text{ cfg} = CFG (\mathfrak{R} : 'a \text{ rules}) (\mathfrak{S} : 'a)$

definition $nonterminals :: 'a \text{ cfg} \Rightarrow 'a \text{ set}$ **where**

$nonterminals G = set (map fst (\mathfrak{R} G)) \cup \{\mathfrak{S} G\}$

definition $is\text{-word} :: 'a \text{ cfg} \Rightarrow 'a \text{ list} \Rightarrow bool$ **where**

$is\text{-word } \mathcal{G} \omega = (nonterminals \mathcal{G} \cap set \omega = \{\})$

definition $derives1 :: 'a \text{ cfg} \Rightarrow 'a \text{ list} \Rightarrow 'a \text{ list} \Rightarrow bool$ **where**

$derives1 \mathcal{G} u v \equiv \exists x y A \alpha.$

$u = x @ [A] @ y \wedge$

$v = x @ \alpha @ y \wedge$

$(A, \alpha) \in set (\mathfrak{R} \mathcal{G})$

definition $derivations1 :: 'a \text{ cfg} \Rightarrow ('a \text{ list} \times 'a \text{ list}) \text{ set}$ **where**

$derivations1 \mathcal{G} \equiv \{ (u,v) \mid u v. derives1 \mathcal{G} u v \}$

definition $derivations :: 'a \text{ cfg} \Rightarrow ('a \text{ list} \times 'a \text{ list}) \text{ set}$ **where**

$derivations \mathcal{G} \equiv (derivations1 \mathcal{G})^*$

definition $derives :: 'a \text{ cfg} \Rightarrow 'a \text{ list} \Rightarrow 'a \text{ list} \Rightarrow bool$ **where**

$derives \mathcal{G} u v \equiv ((u, v) \in derivations \mathcal{G})$

syntax

$derives1 :: 'a \text{ cfg} \Rightarrow 'a \text{ list} \Rightarrow 'a \text{ list} \Rightarrow bool \ (\langle - \vdash - \Rightarrow - \rangle [1000,0,0] 1000)$

syntax

$derives :: 'a \text{ cfg} \Rightarrow 'a \text{ list} \Rightarrow 'a \text{ list} \Rightarrow bool \ (\langle - \vdash - \Rightarrow^* - \rangle [1000,0,0] 1000)$

notation (*latex output*)

$derives1 \ (\langle - \vdash - \Rightarrow - \rangle [1000,0,0] 1000)$

notation (*latex output*)
 $derives (\vdash - \Rightarrow^* -) [1000,0,0] 1000$

end
theory *Derivations*
imports
CFG
begin

3 Adjusted content from AFP/LocalLexing

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

lemma *is-word-empty: is-word* $\mathcal{G} []$ *<proof>*

lemma *derives1-implies-derives[simp]:*
 $derives1 \mathcal{G} a b \Longrightarrow derives \mathcal{G} a b$
<proof>

lemma *derives-trans:*
 $derives \mathcal{G} a b \Longrightarrow derives \mathcal{G} b c \Longrightarrow derives \mathcal{G} a c$
<proof>

lemma *derives1-eq-derivations1:*
 $derives1 \mathcal{G} x y = ((x, y) \in derivations1 \mathcal{G})$
<proof>

lemma *derives-induct[consumes 1, case-names Base Step]:*
assumes *derives:* $derives \mathcal{G} a b$
assumes *Pa:* $P a$
assumes *induct:* $\bigwedge y z. derives \mathcal{G} a y \Longrightarrow derives1 \mathcal{G} y z \Longrightarrow P y \Longrightarrow P z$
shows $P b$
<proof>

definition *Derives1* :: *'a cfg* ⇒ *'a list* ⇒ *nat* ⇒ *'a rule* ⇒ *'a list* ⇒ *bool* **where**
 $Derives1 \mathcal{G} u i r v \equiv \exists x y A \alpha.$
 $u = x @ [A] @ y \wedge$
 $v = x @ \alpha @ y \wedge$
 $(A, \alpha) \in set (\mathfrak{A} \mathcal{G}) \wedge r = (A, \alpha) \wedge i = length x$

lemma *Derives1-split:*
 $Derives1 \mathcal{G} u i r v \Longrightarrow \exists x y. u = x @ [fst r] @ y \wedge v = x @ (snd r) @ y \wedge$
 $length x = i$
<proof>

lemma *Derives1-implies-derives1:* $Derives1 \mathcal{G} u i r v \Longrightarrow derives1 \mathcal{G} u v$
<proof>

lemma *derives1-implies-Derives1*: $\text{derives1 } \mathcal{G} \ u \ v \implies \exists \ i \ r. \text{Derives1 } \mathcal{G} \ u \ i \ r \ v$
 ⟨proof⟩

fun *Derivation* :: 'a cfg \Rightarrow 'a list \Rightarrow 'a derivation \Rightarrow 'a list \Rightarrow bool **where**
Derivation - a [] b = (a = b)
 | *Derivation* \mathcal{G} a (d#D) b = ($\exists \ x. \text{Derives1 } \mathcal{G} \ a \ (\text{fst } d) \ (\text{snd } d) \ x \wedge \text{Derivation } \mathcal{G} \ x \ D \ b$)

lemma *Derivation-implies-derives*: $\text{Derivation } \mathcal{G} \ a \ D \ b \implies \text{derives } \mathcal{G} \ a \ b$
 ⟨proof⟩

lemma *Derivation-Derives1*: $\text{Derivation } \mathcal{G} \ a \ S \ y \implies \text{Derives1 } \mathcal{G} \ y \ i \ r \ z \implies \text{Derivation } \mathcal{G} \ a \ (S@[i,r]) \ z$
 ⟨proof⟩

lemma *derives-implies-Derivation*: $\text{derives } \mathcal{G} \ a \ b \implies \exists \ D. \text{Derivation } \mathcal{G} \ a \ D \ b$
 ⟨proof⟩

lemma *Derives1-rule [elim]*: $\text{Derives1 } \mathcal{G} \ a \ i \ r \ b \implies r \in \text{set } (\mathfrak{R} \ \mathcal{G})$
 ⟨proof⟩

lemma *Derivation-append*: $\text{Derivation } \mathcal{G} \ a \ (D@E) \ c = (\exists \ b. \text{Derivation } \mathcal{G} \ a \ D \ b \wedge \text{Derivation } \mathcal{G} \ b \ E \ c)$
 ⟨proof⟩

lemma *Derivation-implies-append*:
 $\text{Derivation } \mathcal{G} \ a \ D \ b \implies \text{Derivation } \mathcal{G} \ b \ E \ c \implies \text{Derivation } \mathcal{G} \ a \ (D@E) \ c$
 ⟨proof⟩

4 Additional derivation lemmas

lemma *Derives1-prepend*:
assumes $\text{Derives1 } \mathcal{G} \ u \ i \ r \ v$
shows $\text{Derives1 } \mathcal{G} \ (w@u) \ (i + \text{length } w) \ r \ (w@v)$
 ⟨proof⟩

lemma *Derivation-prepend*:
 $\text{Derivation } \mathcal{G} \ b \ D \ b' \implies \text{Derivation } \mathcal{G} \ (a@b) \ (\text{map } (\lambda(i, r). (i + \text{length } a, r)) \ D) \ (a@b')$
 ⟨proof⟩

lemma *Derives1-append*:
assumes $\text{Derives1 } \mathcal{G} \ u \ i \ r \ v$
shows $\text{Derives1 } \mathcal{G} \ (u@w) \ i \ r \ (v@w)$
 ⟨proof⟩

lemma *Derivation-append'*:
 $\text{Derivation } \mathcal{G} \ a \ D \ a' \implies \text{Derivation } \mathcal{G} \ (a@b) \ D \ (a'@b)$
 ⟨proof⟩

lemma *Derivation-append-rewrite:*
assumes *Derivation* $\mathcal{G} \ a \ D \ (b \ @ \ c \ @ \ d) \ \text{Derivation} \ \mathcal{G} \ c \ E \ c'$
shows $\exists F. \ \text{Derivation} \ \mathcal{G} \ a \ F \ (b \ @ \ c' \ @ \ d)$
 $\langle \text{proof} \rangle$

lemma *derives1-if-valid-rule:*
 $(A, \alpha) \in \text{set} \ (\mathfrak{R} \ \mathcal{G}) \implies \text{derives1} \ \mathcal{G} \ [A] \ \alpha$
 $\langle \text{proof} \rangle$

lemma *derives-if-valid-rule:*
 $(A, \alpha) \in \text{set} \ (\mathfrak{R} \ \mathcal{G}) \implies \text{derives} \ \mathcal{G} \ [A] \ \alpha$
 $\langle \text{proof} \rangle$

lemma *Derivation-from-empty:*
 $\text{Derivation} \ \mathcal{G} \ [] \ D \ a \implies a = []$
 $\langle \text{proof} \rangle$

lemma *Derivation-concat-split:*
 $\text{Derivation} \ \mathcal{G} \ (a@b) \ D \ c \implies \exists E \ F \ a' \ b'. \ \text{Derivation} \ \mathcal{G} \ a \ E \ a' \wedge \text{Derivation} \ \mathcal{G} \ b \ F \ b' \wedge$
 $c = a' \ @ \ b' \wedge \text{length} \ E \leq \text{length} \ D \wedge \text{length} \ F \leq \text{length} \ D$
 $\langle \text{proof} \rangle$

lemma *Derivation- $\mathfrak{S}1$:*
assumes $\text{Derivation} \ \mathcal{G} \ [\mathfrak{S} \ \mathcal{G}] \ D \ \omega \ \text{is-word} \ \mathcal{G} \ \omega$
shows $\exists \alpha \ E. \ \text{Derivation} \ \mathcal{G} \ \alpha \ E \ \omega \wedge (\mathfrak{S} \ \mathcal{G}, \alpha) \in \text{set} \ (\mathfrak{R} \ \mathcal{G})$
 $\langle \text{proof} \rangle$

end
theory *Earley*
imports
Derivations
begin

5 Slices

fun *slice* :: $'a \ \text{list} \Rightarrow \text{nat} \Rightarrow \text{nat} \Rightarrow 'a \ \text{list}$ **where**
 $\text{slice} \ [] \ - \ - = []$
 $| \ \text{slice} \ (x\#xs) \ - \ 0 = []$
 $| \ \text{slice} \ (x\#xs) \ 0 \ (\text{Suc} \ b) = x \ \# \ \text{slice} \ xs \ 0 \ b$
 $| \ \text{slice} \ (x\#xs) \ (\text{Suc} \ a) \ (\text{Suc} \ b) = \text{slice} \ xs \ a \ b$

syntax
 $\text{slice} \ :: \ 'a \ \text{list} \Rightarrow \text{nat} \Rightarrow \text{nat} \Rightarrow 'a \ \text{list} \ (\langle _ _ _ \rangle \ [1000,0,0] \ 1000)$

notation (*latex output*)
 $\text{slice} \ (\langle _ _ _ \rangle \ [1000,0,0] \ 1000)$

lemma *slice-drop-take*:

$$\text{slice } xs \ a \ b = \text{drop } a \ (\text{take } b \ xs)$$

<proof>

lemma *slice-append-aux*:

$$\text{Suc } b \leq c \implies \text{slice } (x\#xs) \ (\text{Suc } b) \ c = \text{slice } xs \ b \ (c-1)$$

<proof>

lemma *slice-concat*:

$$a \leq b \implies b \leq c \implies \text{slice } xs \ a \ b \ @ \ \text{slice } xs \ b \ c = \text{slice } xs \ a \ c$$

<proof>

lemma *slice-concat-Ex*:

$$a \leq c \implies \text{slice } xs \ a \ c = ys \ @ \ zs \implies \exists b. \ ys = \text{slice } xs \ a \ b \ \wedge \ zs = \text{slice } xs \ b \ c \ \wedge$$

$$a \leq b \ \wedge \ b \leq c$$

<proof>

lemma *slice-nth*:

$$a < \text{length } xs \implies \text{slice } xs \ a \ (a+1) = [xs!a]$$

<proof>

lemma *slice-append-nth*:

$$a \leq b \implies b < \text{length } xs \implies \text{slice } xs \ a \ b \ @ \ [xs!b] = \text{slice } xs \ a \ (b+1)$$

<proof>

lemma *slice-empty*:

$$b \leq a \implies \text{slice } xs \ a \ b = []$$

<proof>

lemma *slice-id[simp]*:

$$\text{slice } xs \ 0 \ (\text{length } xs) = xs$$

<proof>

lemma *slice-singleton*:

$$b \leq \text{length } xs \implies [x] = \text{slice } xs \ a \ b \implies b = a + 1$$

<proof>

6 Earley recognizer

6.1 Earley items

definition *lhs-rule* :: 'a rule \Rightarrow 'a where

$$\text{lhs-rule} \equiv \text{fst}$$

definition *rhs-rule* :: 'a rule \Rightarrow 'a list where

$$\text{rhs-rule} \equiv \text{snd}$$

datatype 'a item =

$$\text{Item } (\text{rule-item} : 'a \ \text{rule}) \ (\text{dot-item} : \text{nat}) \ (\text{start-item} : \text{nat}) \ (\text{end-item} : \text{nat})$$

definition *lhs-item* :: 'a item \Rightarrow 'a **where**

lhs-item $x \equiv$ *lhs-rule* (*rule-item* x)

definition *rhs-item* :: 'a item \Rightarrow 'a list **where**

rhs-item $x \equiv$ *rhs-rule* (*rule-item* x)

definition α -*item* :: 'a item \Rightarrow 'a list **where**

α -*item* $x \equiv$ *take* (*dot-item* x) (*rhs-item* x)

definition β -*item* :: 'a item \Rightarrow 'a list **where**

β -*item* $x \equiv$ *drop* (*dot-item* x) (*rhs-item* x)

definition *is-complete* :: 'a item \Rightarrow bool **where**

is-complete $x \equiv$ *dot-item* $x \geq$ *length* (*rhs-item* x)

definition *next-symbol* :: 'a item \Rightarrow 'a option **where**

next-symbol $x \equiv$ *if is-complete* x *then* None *else* Some (*rhs-item* x ! *dot-item* x)

lemmas *item-defs* = *lhs-item-def* *rhs-item-def* α -*item-def* β -*item-def* *lhs-rule-def* *rhs-rule-def*

definition *is-finished* :: 'a cfg \Rightarrow 'a list \Rightarrow 'a item \Rightarrow bool **where**

is-finished $\mathcal{G} \ \omega \ x \equiv$

lhs-item $x = \mathfrak{S} \ \mathcal{G} \ \wedge$

start-item $x = 0 \ \wedge$

end-item $x =$ *length* $\omega \ \wedge$

is-complete x

definition *recognizing* :: 'a item set \Rightarrow 'a cfg \Rightarrow 'a list \Rightarrow bool **where**

recognizing $I \ \mathcal{G} \ \omega \equiv \exists x \in I. \text{is-finished } \mathcal{G} \ \omega \ x$

inductive-set *Earley* :: 'a cfg \Rightarrow 'a list \Rightarrow 'a item set

for $\mathcal{G} ::$ 'a cfg **and** $\omega ::$ 'a list **where**

Init: $r \in \text{set } (\mathfrak{R} \ \mathcal{G}) \Longrightarrow \text{fst } r = \mathfrak{S} \ \mathcal{G} \Longrightarrow$

Item $r \ 0 \ 0 \ 0 \in \text{Earley } \mathcal{G} \ \omega$

| *Scan*: $x = \text{Item } r \ b \ i \ j \Longrightarrow x \in \text{Earley } \mathcal{G} \ \omega \Longrightarrow$

$\omega!j = a \Longrightarrow j < \text{length } \omega \Longrightarrow \text{next-symbol } x = \text{Some } a \Longrightarrow$

Item $r \ (b + 1) \ i \ (j + 1) \in \text{Earley } \mathcal{G} \ \omega$

| *Predict*: $x = \text{Item } r \ b \ i \ j \Longrightarrow x \in \text{Earley } \mathcal{G} \ \omega \Longrightarrow$

$r' \in \text{set } (\mathfrak{R} \ \mathcal{G}) \Longrightarrow \text{next-symbol } x = \text{Some } (\text{lhs-rule } r') \Longrightarrow$

Item $r' \ 0 \ j \ j \in \text{Earley } \mathcal{G} \ \omega$

| *Complete*: $x = \text{Item } r_x \ b_x \ i \ j \Longrightarrow x \in \text{Earley } \mathcal{G} \ \omega \Longrightarrow y = \text{Item } r_y \ b_y \ j \ k \Longrightarrow$

$y \in \text{Earley } \mathcal{G} \ \omega \Longrightarrow$

is-complete $y \Longrightarrow \text{next-symbol } x = \text{Some } (\text{lhs-item } y) \Longrightarrow$

Item $r_x \ (b_x + 1) \ i \ k \in \text{Earley } \mathcal{G} \ \omega$

6.2 Well-formedness

definition $wf\text{-item} :: 'a\ cfg \Rightarrow 'a\ list \Rightarrow 'a\ item \Rightarrow bool$ **where**

$wf\text{-item}\ \mathcal{G}\ \omega\ x \equiv$
 $rule\text{-item}\ x \in set\ (\mathfrak{R}\ \mathcal{G}) \wedge$
 $dot\text{-item}\ x \leq length\ (rhs\text{-item}\ x) \wedge$
 $start\text{-item}\ x \leq end\text{-item}\ x \wedge$
 $end\text{-item}\ x \leq length\ \omega$

lemma $wf\text{-Init}$:

assumes $r \in set\ (\mathfrak{R}\ \mathcal{G})\ fst\ r = \mathfrak{S}\ \mathcal{G}$
shows $wf\text{-item}\ \mathcal{G}\ \omega\ (Item\ r\ 0\ 0\ 0)$
 $\langle proof \rangle$

lemma $wf\text{-Scan}$:

assumes $x = Item\ r\ b\ i\ j\ wf\text{-item}\ \mathcal{G}\ \omega\ x\ \omega!j = a\ j < length\ \omega\ next\text{-symbol}\ x =$
 $Some\ a$
shows $wf\text{-item}\ \mathcal{G}\ \omega\ (Item\ r\ (b + 1)\ i\ (j+1))$
 $\langle proof \rangle$

lemma $wf\text{-Predict}$:

assumes $x = Item\ r\ b\ i\ j\ wf\text{-item}\ \mathcal{G}\ \omega\ x\ r' \in set\ (\mathfrak{R}\ \mathcal{G})\ next\text{-symbol}\ x = Some$
 $(lhs\text{-rule}\ r')$
shows $wf\text{-item}\ \mathcal{G}\ \omega\ (Item\ r'\ 0\ j\ j)$
 $\langle proof \rangle$

lemma $wf\text{-Complete}$:

assumes $x = Item\ r_x\ b_x\ i\ j\ wf\text{-item}\ \mathcal{G}\ \omega\ x\ y = Item\ r_y\ b_y\ j\ k\ wf\text{-item}\ \mathcal{G}\ \omega\ y$
assumes $is\text{-complete}\ y\ next\text{-symbol}\ x = Some\ (lhs\text{-item}\ y)$
shows $wf\text{-item}\ \mathcal{G}\ \omega\ (Item\ r_x\ (b_x + 1)\ i\ k)$
 $\langle proof \rangle$

lemma $wf\text{-Earley}$:

assumes $x \in Earley\ \mathcal{G}\ \omega$
shows $wf\text{-item}\ \mathcal{G}\ \omega\ x$
 $\langle proof \rangle$

6.3 Soundness

definition $sound\text{-item} :: 'a\ cfg \Rightarrow 'a\ list \Rightarrow 'a\ item \Rightarrow bool$ **where**

$sound\text{-item}\ \mathcal{G}\ \omega\ x \equiv \mathcal{G} \vdash [lhs\text{-item}\ x] \Rightarrow^* (slice\ \omega\ (start\text{-item}\ x)\ (end\text{-item}\ x)\ @$
 $\beta\text{-item}\ x)$

lemma $sound\text{-Init}$:

assumes $r \in set\ (\mathfrak{R}\ \mathcal{G})\ fst\ r = \mathfrak{S}\ \mathcal{G}$
shows $sound\text{-item}\ \mathcal{G}\ \omega\ (Item\ r\ 0\ 0\ 0)$
 $\langle proof \rangle$

lemma $sound\text{-Scan}$:

assumes $x = Item\ r\ b\ i\ j\ wf\text{-item}\ \mathcal{G}\ \omega\ x\ sound\text{-item}\ \mathcal{G}\ \omega\ x$

assumes $\omega!j = a \ j < \text{length } \omega \ \text{next-symbol } x = \text{Some } a$
shows $\text{sound-item } \mathcal{G} \ \omega \ (\text{Item } r \ (b+1) \ i \ (j+1))$
 $\langle \text{proof} \rangle$

lemma *sound-Predict*:

assumes $x = \text{Item } r \ b \ i \ j \ \text{wf-item } \mathcal{G} \ \omega \ x \ \text{sound-item } \mathcal{G} \ \omega \ x$
assumes $r' \in \text{set } (\mathfrak{R} \ \mathcal{G}) \ \text{next-symbol } x = \text{Some } (\text{lhs-rule } r')$
shows $\text{sound-item } \mathcal{G} \ \omega \ (\text{Item } r' \ 0 \ j \ j)$
 $\langle \text{proof} \rangle$

lemma *sound-Complete*:

assumes $x = \text{Item } r_x \ b_x \ i \ j \ \text{wf-item } \mathcal{G} \ \omega \ x \ \text{sound-item } \mathcal{G} \ \omega \ x$
assumes $y = \text{Item } r_y \ b_y \ j \ k \ \text{wf-item } \mathcal{G} \ \omega \ y \ \text{sound-item } \mathcal{G} \ \omega \ y$
assumes $\text{is-complete } y \ \text{next-symbol } x = \text{Some } (\text{lhs-item } y)$
shows $\text{sound-item } \mathcal{G} \ \omega \ (\text{Item } r_x \ (b_x + 1) \ i \ k)$
 $\langle \text{proof} \rangle$

lemma *sound-Earley*:

assumes $x \in \text{Earley } \mathcal{G} \ \omega \ \text{wf-item } \mathcal{G} \ \omega \ x$
shows $\text{sound-item } \mathcal{G} \ \omega \ x$
 $\langle \text{proof} \rangle$

theorem *soundness-Earley*:

assumes $\text{recognizing } (\text{Earley } \mathcal{G} \ \omega) \ \mathcal{G} \ \omega$
shows $\mathcal{G} \vdash [\mathfrak{G} \ \mathcal{G}] \Rightarrow^* \omega$
 $\langle \text{proof} \rangle$

6.4 Completeness

definition *partially-completed* $:: \text{nat} \Rightarrow 'a \ \text{cfg} \Rightarrow 'a \ \text{list} \Rightarrow 'a \ \text{item set} \Rightarrow ('a \ \text{derivation} \Rightarrow \text{bool}) \Rightarrow \text{bool}$ **where**

$\text{partially-completed } k \ \mathcal{G} \ \omega \ I \ P \equiv \forall r \ b \ i' \ i \ j \ x \ a \ D.$
 $i \leq j \wedge j \leq k \wedge k \leq \text{length } \omega \wedge$
 $x = \text{Item } r \ b \ i' \ i \wedge x \in I \wedge \text{next-symbol } x = \text{Some } a \wedge$
 $\text{Derivation } \mathcal{G} \ [a] \ D \ (\text{slice } \omega \ i \ j) \wedge P \ D \longrightarrow$
 $\text{Item } r \ (b+1) \ i' \ j \in I$

lemma *partially-completed-upto*:

assumes $j \leq k \ k \leq \text{length } \omega$
assumes $x = \text{Item } (N, \alpha) \ d \ i \ j \ x \in I \ \forall x \in I. \ \text{wf-item } \mathcal{G} \ \omega \ x$
assumes $\text{Derivation } \mathcal{G} \ (\beta\text{-item } x) \ D \ (\text{slice } \omega \ j \ k)$
assumes $\text{partially-completed } k \ \mathcal{G} \ \omega \ I \ (\lambda D'. \ \text{length } D' \leq \text{length } D)$
shows $\text{Item } (N, \alpha) \ (\text{length } \alpha) \ i \ k \in I$
 $\langle \text{proof} \rangle$

lemma *partially-completed-Earley*:

$\text{partially-completed } k \ \mathcal{G} \ \omega \ (\text{Earley } \mathcal{G} \ \omega) \ (\lambda-. \ \text{True})$
 $\langle \text{proof} \rangle$

theorem *completeness-Earley*:
 assumes $\mathcal{G} \vdash [\mathfrak{G} \mathcal{G}] \Rightarrow^* \omega$ *is-word* $\mathcal{G} \ \omega$
 shows *recognizing* (*Earley* $\mathcal{G} \ \omega$) $\mathcal{G} \ \omega$
<proof>

6.5 Correctness

theorem *correctness-Earley*:
 assumes *is-word* $\mathcal{G} \ \omega$
 shows *recognizing* (*Earley* $\mathcal{G} \ \omega$) $\mathcal{G} \ \omega \longleftrightarrow \mathcal{G} \vdash [\mathfrak{G} \mathcal{G}] \Rightarrow^* \omega$
<proof>

6.6 Finiteness

lemma *finiteness-empty*:
 set $(\mathfrak{R} \mathcal{G}) = \{\}$ \implies *finite* $\{ x \mid x. \text{wf-item } \mathcal{G} \ \omega \ x \}$
<proof>

fun *item-intro* :: 'a rule \times nat \times nat \times nat \Rightarrow 'a item **where**
item-intro (rule, dot, origin, ends) = *Item* rule dot origin ends

lemma *finiteness-nonempty*:
 assumes set $(\mathfrak{R} \mathcal{G}) \neq \{\}$
 shows *finite* $\{ x. \text{wf-item } \mathcal{G} \ \omega \ x \}$
<proof>

lemma *finiteness-UNIV-wf-item*:
finite $\{ x. \text{wf-item } \mathcal{G} \ \omega \ x \}$
<proof>

theorem *finiteness-Earley*:
finite (*Earley* $\mathcal{G} \ \omega$)
<proof>

end

theory *Earley-Fixpoint*

imports

Earley

Limit

begin

7 Earley fixpoint

7.1 Definitions

definition *init-item* :: 'a rule \Rightarrow nat \Rightarrow 'a item **where**
init-item r k \equiv *Item* r 0 k k

definition *inc-item* :: 'a item \Rightarrow nat \Rightarrow 'a item **where**

$inc\text{-item } x \ k \equiv Item \ (rule\text{-item } x) \ (dot\text{-item } x + 1) \ (start\text{-item } x) \ k$

definition $bin :: 'a \ item \ set \Rightarrow \ nat \Rightarrow 'a \ item \ set$ **where**

$bin \ I \ k \equiv \{ x . x \in I \wedge end\text{-item } x = k \}$

definition $prev\text{-symbol} :: 'a \ item \Rightarrow 'a \ option$ **where**

$prev\text{-symbol } x \equiv \text{if } dot\text{-item } x = 0 \text{ then } None \text{ else } Some \ (rhs\text{-item } x \ ! \ (dot\text{-item } x - 1))$

definition $base :: 'a \ list \Rightarrow 'a \ item \ set \Rightarrow \ nat \Rightarrow 'a \ item \ set$ **where**

$base \ \omega \ I \ k \equiv \{ x . x \in I \wedge end\text{-item } x = k \wedge k > 0 \wedge prev\text{-symbol } x = Some \ (\omega!(k-1)) \}$

definition $Init_F :: 'a \ cfg \Rightarrow 'a \ item \ set$ **where**

$Init_F \ \mathcal{G} \equiv \{ init\text{-item } r \ 0 \mid r . r \in set \ (\mathfrak{R} \ \mathcal{G}) \wedge fst \ r = (\mathfrak{S} \ \mathcal{G}) \}$

definition $Scan_F :: \ nat \Rightarrow 'a \ list \Rightarrow 'a \ item \ set \Rightarrow 'a \ item \ set$ **where**

$Scan_F \ k \ \omega \ I \equiv \{ inc\text{-item } x \ (k+1) \mid x \ a .$
 $x \in bin \ I \ k \wedge$
 $\omega!k = a \wedge$
 $k < length \ \omega \wedge$
 $next\text{-symbol } x = Some \ a \}$

definition $Predict_F :: \ nat \Rightarrow 'a \ cfg \Rightarrow 'a \ item \ set \Rightarrow 'a \ item \ set$ **where**

$Predict_F \ k \ \mathcal{G} \ I \equiv \{ init\text{-item } r \ k \mid r \ x .$
 $r \in set \ (\mathfrak{R} \ \mathcal{G}) \wedge$
 $x \in bin \ I \ k \wedge$
 $next\text{-symbol } x = Some \ (lhs\text{-rule } r) \}$

definition $Complete_F :: \ nat \Rightarrow 'a \ item \ set \Rightarrow 'a \ item \ set$ **where**

$Complete_F \ k \ I \equiv \{ inc\text{-item } x \ k \mid x \ y .$
 $x \in bin \ I \ (start\text{-item } y) \wedge$
 $y \in bin \ I \ k \wedge$
 $is\text{-complete } y \wedge$
 $next\text{-symbol } x = Some \ (lhs\text{-item } y) \}$

definition $Earley_F\text{-bin}\text{-step} :: \ nat \Rightarrow 'a \ cfg \Rightarrow 'a \ list \Rightarrow 'a \ item \ set \Rightarrow 'a \ item \ set$ **where**

$Earley_F\text{-bin}\text{-step} \ k \ \mathcal{G} \ \omega \ I = I \cup Scan_F \ k \ \omega \ I \cup Complete_F \ k \ I \cup Predict_F \ k \ \mathcal{G} \ I$

definition $Earley_F\text{-bin} :: \ nat \Rightarrow 'a \ cfg \Rightarrow 'a \ list \Rightarrow 'a \ item \ set \Rightarrow 'a \ item \ set$ **where**

$Earley_F\text{-bin} \ k \ \mathcal{G} \ \omega \ I \equiv limit \ (Earley_F\text{-bin}\text{-step} \ k \ \mathcal{G} \ \omega) \ I$

fun $Earley_F\text{-bins} :: \ nat \Rightarrow 'a \ cfg \Rightarrow 'a \ list \Rightarrow 'a \ item \ set$ **where**

$Earley_F\text{-bins} \ 0 \ \mathcal{G} \ \omega = Earley_F\text{-bin} \ 0 \ \mathcal{G} \ \omega \ (Init_F \ \mathcal{G})$
 $\mid Earley_F\text{-bins} \ (Suc \ n) \ \mathcal{G} \ \omega = Earley_F\text{-bin} \ (Suc \ n) \ \mathcal{G} \ \omega \ (Earley_F\text{-bins} \ n \ \mathcal{G} \ \omega)$

definition $Earley_F :: 'a \ cfg \Rightarrow 'a \ list \Rightarrow 'a \ item \ set$ **where**

$Earley_F \mathcal{G} \omega \equiv Earley_F\text{-bins} (\text{length } \omega) \mathcal{G} \omega$

7.2 Monotonicity and Absorption

lemma *Earley_F-bin-step-empty:*

$Earley_F\text{-bin-step } k \mathcal{G} \omega \{\} = \{\}$
<proof>

lemma *Earley_F-bin-step-setmonotone:*

$setmonotone (Earley_F\text{-bin-step } k \mathcal{G} \omega)$
<proof>

lemma *Earley_F-bin-step-continuous:*

$continuous (Earley_F\text{-bin-step } k \mathcal{G} \omega)$
<proof>

lemma *Earley_F-bin-step-regular:*

$regular (Earley_F\text{-bin-step } k \mathcal{G} \omega)$
<proof>

lemma *Earley_F-bin-idem:*

$Earley_F\text{-bin } k \mathcal{G} \omega (Earley_F\text{-bin } k \mathcal{G} \omega I) = Earley_F\text{-bin } k \mathcal{G} \omega I$
<proof>

lemma *Scan_F-bin-absorb:*

$Scan_F k \omega (bin I k) = Scan_F k \omega I$
<proof>

lemma *Predict_F-bin-absorb:*

$Predict_F k \mathcal{G} (bin I k) = Predict_F k \mathcal{G} I$
<proof>

lemma *Scan_F-Un:*

$Scan_F k \omega (I \cup J) = Scan_F k \omega I \cup Scan_F k \omega J$
<proof>

lemma *Predict_F-Un:*

$Predict_F k \mathcal{G} (I \cup J) = Predict_F k \mathcal{G} I \cup Predict_F k \mathcal{G} J$
<proof>

lemma *Scan_F-sub-mono:*

$I \subseteq J \implies Scan_F k \omega I \subseteq Scan_F k \omega J$
<proof>

lemma *Predict_F-sub-mono:*

$I \subseteq J \implies Predict_F k \mathcal{G} I \subseteq Predict_F k \mathcal{G} J$
<proof>

lemma *Complete_F-sub-mono:*

$I \subseteq J \implies \text{Complete}_F k I \subseteq \text{Complete}_F k J$
(proof)

lemma *Earley_F-bin-step-sub-mono:*

$I \subseteq J \implies \text{Earley}_F\text{-bin-step } k \mathcal{G} \omega I \subseteq \text{Earley}_F\text{-bin-step } k \mathcal{G} \omega J$
(proof)

lemma *funpower-sub-mono:*

$I \subseteq J \implies \text{funpower} (\text{Earley}_F\text{-bin-step } k \mathcal{G} \omega) n I \subseteq \text{funpower} (\text{Earley}_F\text{-bin-step } k \mathcal{G} \omega) n J$
(proof)

lemma *Earley_F-bin-sub-mono:*

$I \subseteq J \implies \text{Earley}_F\text{-bin } k \mathcal{G} \omega I \subseteq \text{Earley}_F\text{-bin } k \mathcal{G} \omega J$
(proof)

lemma *Scan_F-Earley_F-bin-step-mono:*

$\text{Scan}_F k \omega I \subseteq \text{Earley}_F\text{-bin-step } k \mathcal{G} \omega I$
(proof)

lemma *Predict_F-Earley_F-bin-step-mono:*

$\text{Predict}_F k \mathcal{G} I \subseteq \text{Earley}_F\text{-bin-step } k \mathcal{G} \omega I$
(proof)

lemma *Complete_F-Earley_F-bin-step-mono:*

$\text{Complete}_F k I \subseteq \text{Earley}_F\text{-bin-step } k \mathcal{G} \omega I$
(proof)

lemma *Earley_F-bin-step-Earley_F-bin-mono:*

$\text{Earley}_F\text{-bin-step } k \mathcal{G} \omega I \subseteq \text{Earley}_F\text{-bin } k \mathcal{G} \omega I$
(proof)

lemma *Scan_F-Earley_F-bin-mono:*

$\text{Scan}_F k \omega I \subseteq \text{Earley}_F\text{-bin } k \mathcal{G} \omega I$
(proof)

lemma *Predict_F-Earley_F-bin-mono:*

$\text{Predict}_F k \mathcal{G} I \subseteq \text{Earley}_F\text{-bin } k \mathcal{G} \omega I$
(proof)

lemma *Complete_F-Earley_F-bin-mono:*

$\text{Complete}_F k I \subseteq \text{Earley}_F\text{-bin } k \mathcal{G} \omega I$
(proof)

lemma *Earley_F-bin-mono:*

$I \subseteq \text{Earley}_F\text{-bin } k \mathcal{G} \omega I$
(proof)

lemma *Init_F-sub-Earley_F-bins:*

$Init_F \mathcal{G} \subseteq Earley_F\text{-bins } n \mathcal{G} \omega$
<proof>

7.3 Soundness

lemma *Init_F-sub-Earley*:

$Init_F \mathcal{G} \subseteq Earley \mathcal{G} \omega$
<proof>

lemma *Scan_F-sub-Earley*:

assumes $I \subseteq Earley \mathcal{G} \omega$
shows $Scan_F k \omega I \subseteq Earley \mathcal{G} \omega$
<proof>

lemma *Predict_F-sub-Earley*:

assumes $I \subseteq Earley \mathcal{G} \omega$
shows $Predict_F k \mathcal{G} I \subseteq Earley \mathcal{G} \omega$
<proof>

lemma *Complete_F-sub-Earley*:

assumes $I \subseteq Earley \mathcal{G} \omega$
shows $Complete_F k I \subseteq Earley \mathcal{G} \omega$
<proof>

lemma *Earley_F-bin-step-sub-Earley*:

assumes $I \subseteq Earley \mathcal{G} \omega$
shows $Earley_F\text{-bin-step } k \mathcal{G} \omega I \subseteq Earley \mathcal{G} \omega$
<proof>

lemma *Earley_F-bin-sub-Earley*:

assumes $I \subseteq Earley \mathcal{G} \omega$
shows $Earley_F\text{-bin } k \mathcal{G} \omega I \subseteq Earley \mathcal{G} \omega$
<proof>

lemma *Earley_F-bins-sub-Earley*:

shows $Earley_F\text{-bins } n \mathcal{G} \omega \subseteq Earley \mathcal{G} \omega$
<proof>

lemma *Earley_F-sub-Earley*:

shows $Earley_F \mathcal{G} \omega \subseteq Earley \mathcal{G} \omega$
<proof>

theorem *soundness-Earley_F*:

assumes *recognizing* $(Earley_F \mathcal{G} \omega) \mathcal{G} \omega$
shows $\mathcal{G} \vdash [\mathfrak{S} \mathcal{G}] \Rightarrow^* \omega$
<proof>

7.4 Completeness

lemma *Earley_F-bin-sub-Earley_F-bin*:

assumes $Init_F \mathcal{G} \subseteq I$
assumes $\forall k' < k. bin (Earley \mathcal{G} \omega) k' \subseteq I$
assumes $base \omega (Earley \mathcal{G} \omega) k \subseteq I$
shows $bin (Earley \mathcal{G} \omega) k \subseteq bin (Earley_F-bin k \mathcal{G} \omega I) k$
 <proof>

lemma *Earley-base-sub-Earley_F-bin*:
assumes $Init_F \mathcal{G} \subseteq I$
assumes $\forall k' < k. bin (Earley \mathcal{G} \omega) k' \subseteq I$
assumes $base \omega (Earley \mathcal{G} \omega) k \subseteq I$
assumes *is-word* $\mathcal{G} \omega$
shows $base \omega (Earley \mathcal{G} \omega) (k+1) \subseteq bin (Earley_F-bin k \mathcal{G} \omega I) (k+1)$
 <proof>

lemma *Earley_F-bin-k-sub-Earley_F-bins*:
assumes *is-word* $\mathcal{G} \omega k \leq n$
shows $bin (Earley \mathcal{G} \omega) k \subseteq Earley_F-bins n \mathcal{G} \omega$
 <proof>

lemma *Earley-sub-Earley_F*:
assumes *is-word* $\mathcal{G} \omega$
shows $Earley \mathcal{G} \omega \subseteq Earley_F \mathcal{G} \omega$
 <proof>

theorem *completeness-Earley_F*:
assumes $\mathcal{G} \vdash [\mathfrak{G} \mathcal{G}] \Rightarrow^* \omega$ *is-word* $\mathcal{G} \omega$
shows $recognizing (Earley_F \mathcal{G} \omega) \mathcal{G} \omega$
 <proof>

7.5 Correctness

theorem *Earley-eq-Earley_F*:
assumes *is-word* $\mathcal{G} \omega$
shows $Earley \mathcal{G} \omega = Earley_F \mathcal{G} \omega$
 <proof>

theorem *correctness-Earley_F*:
assumes *is-word* $\mathcal{G} \omega$
shows $recognizing (Earley_F \mathcal{G} \omega) \mathcal{G} \omega \longleftrightarrow \mathcal{G} \vdash [\mathfrak{G} \mathcal{G}] \Rightarrow^* \omega$
 <proof>

end
theory *Earley-Recognizer*
imports
 Earley-Fixpoint
begin

8 Earley recognizer

8.1 List auxiliaries

fun *filter-with-index'* :: *nat* \Rightarrow (*'a* \Rightarrow *bool*) \Rightarrow *'a list* \Rightarrow (*'a* \times *nat*) *list* **where**
 filter-with-index' - - [] = []
| *filter-with-index'* *i P (x#xs)* = (
 if *P x* then (*x,i*) # *filter-with-index'* (*i+1*) *P xs*
 else *filter-with-index'* (*i+1*) *P xs*)

definition *filter-with-index* :: (*'a* \Rightarrow *bool*) \Rightarrow *'a list* \Rightarrow (*'a* \times *nat*) *list* **where**
 filter-with-index P xs = *filter-with-index'* 0 *P xs*

lemma *filter-with-index'-P*:
 (*x, n*) \in *set (filter-with-index' i P xs)* \Longrightarrow *P x*
 <*proof*>

lemma *filter-with-index-P*:
 (*x, n*) \in *set (filter-with-index P xs)* \Longrightarrow *P x*
 <*proof*>

lemma *filter-with-index'-cong-filter*:
 map fst (filter-with-index' i P xs) = *filter P xs*
 <*proof*>

lemma *filter-with-index-cong-filter*:
 map fst (filter-with-index P xs) = *filter P xs*
 <*proof*>

lemma *size-index-filter-with-index'*:
 (*x, n*) \in *set (filter-with-index' i P xs)* \Longrightarrow $n \geq i$
 <*proof*>

lemma *index-filter-with-index'-lt-length*:
 (*x, n*) \in *set (filter-with-index' i P xs)* \Longrightarrow $n-i < \text{length } xs$
 <*proof*>

lemma *index-filter-with-index-lt-length*:
 (*x, n*) \in *set (filter-with-index P xs)* \Longrightarrow $n < \text{length } xs$
 <*proof*>

lemma *filter-with-index'-nth*:
 (*x, n*) \in *set (filter-with-index' i P xs)* \Longrightarrow $xs ! (n-i) = x$
 <*proof*>

lemma *filter-with-index-nth*:
 (*x, n*) \in *set (filter-with-index P xs)* \Longrightarrow $xs ! n = x$
 <*proof*>

lemma *filter-with-index-nonempty*:

$x \in \text{set } xs \implies P x \implies \text{filter-with-index } P \text{ } xs \neq []$
 ⟨proof⟩

lemma *filter-with-index'-Ex-first*:

$(\exists x \ i \ xs'. \text{filter-with-index}' \ n \ P \ xs = (x, i)\#xs') \longleftrightarrow (\exists x \in \text{set } xs. P \ x)$
 ⟨proof⟩

lemma *filter-with-index-Ex-first*:

$(\exists x \ i \ xs'. \text{filter-with-index} \ P \ xs = (x, i)\#xs') \longleftrightarrow (\exists x \in \text{set } xs. P \ x)$
 ⟨proof⟩

8.2 Definitions

datatype *pointer* =

Null
 | *Pre nat* — pre
 | *PreRed nat × nat × nat (nat × nat × nat) list* — k', pre, red

type-synonym *'a bin* = *'a item × pointer* *list*

type-synonym *'a bins* = *'a bin list*

definition *items* :: *'a bin* ⇒ *'a item list* **where**

items *b* ≡ *map fst b*

definition *pointers* :: *'a bin* ⇒ *pointer list* **where**

pointers *b* ≡ *map snd b*

definition *bins-eq-items* :: *'a bins* ⇒ *'a bins* ⇒ *bool* **where**

bins-eq-items *bs0* *bs1* ≡ *map items bs0 = map items bs1*

definition *bins* :: *'a bins* ⇒ *'a item set* **where**

bins *bs* ≡ $\bigcup \{ \text{set } (\text{items } (bs!k)) \mid k. k < \text{length } bs \}$

definition *bin-upto* :: *'a bin* ⇒ *nat* ⇒ *'a item set* **where**

bin-upto *b* *i* ≡ $\{ \text{items } b ! j \mid j. j < i \wedge j < \text{length } (\text{items } b) \}$

definition *bins-upto* :: *'a bins* ⇒ *nat* ⇒ *nat* ⇒ *'a item set* **where**

bins-upto *bs* *k* *i* ≡ $\bigcup \{ \text{set } (\text{items } (bs ! l)) \mid l. l < k \} \cup \text{bin-upto } (bs ! k) \ i$

definition *wf-bin-items* :: *'a cfg* ⇒ *'a list* ⇒ *nat* ⇒ *'a item list* ⇒ *bool* **where**

wf-bin-items $\mathcal{G} \ \omega \ k \ xs \equiv \forall x \in \text{set } xs. \text{wf-item } \mathcal{G} \ \omega \ x \wedge \text{end-item } x = k$

definition *wf-bin* :: *'a cfg* ⇒ *'a list* ⇒ *nat* ⇒ *'a bin* ⇒ *bool* **where**

wf-bin $\mathcal{G} \ \omega \ k \ b \equiv \text{distinct } (\text{items } b) \wedge \text{wf-bin-items } \mathcal{G} \ \omega \ k \ (\text{items } b)$

definition *wf-bins* :: *'a cfg* ⇒ *'a list* ⇒ *'a bins* ⇒ *bool* **where**

wf-bins $\mathcal{G} \ \omega \ bs \equiv \forall k < \text{length } bs. \text{wf-bin } \mathcal{G} \ \omega \ k \ (bs!k)$

definition *ε-free* :: *'a cfg* ⇒ *bool* **where**

ε -free $\mathcal{G} = (\forall r \in \text{set } (\mathfrak{R} \mathcal{G}). \text{rhs-rule } r \neq [])$

definition *nonempty-derives* :: 'a cfg \Rightarrow bool **where**
nonempty-derives $\mathcal{G} \equiv \forall s. \neg \mathcal{G} \vdash [s] \Rightarrow^* []$

definition *Init_L* :: 'a cfg \Rightarrow 'a list \Rightarrow 'a bins **where**

Init_L $\mathcal{G} \ \omega \equiv$
 let *rs* = filter ($\lambda r. \text{lhs-rule } r = \mathfrak{S} \mathcal{G}$) (remdups ($\mathfrak{R} \mathcal{G}$)) in
 let *b0* = map ($\lambda r. (\text{init-item } r \ 0, \text{Null})$) *rs* in
 let *bs* = replicate (length $\omega + 1$) ([]) in
bs[0 := *b0*]

definition *Scan_L* :: nat \Rightarrow 'a list \Rightarrow 'a \Rightarrow 'a item \Rightarrow nat \Rightarrow ('a item \times pointer) list **where**

Scan_L *k* ω *a* *x* *pre* \equiv
 if $\omega!k = a$ then
 let *x'* = inc-item *x* (*k*+1) in
 [(*x'*, *Pre pre*)]
 else []

definition *Predict_L* :: nat \Rightarrow 'a cfg \Rightarrow 'a \Rightarrow ('a item \times pointer) list **where**

Predict_L *k* \mathcal{G} *X* \equiv
 let *rs* = filter ($\lambda r. \text{lhs-rule } r = X$) ($\mathfrak{R} \mathcal{G}$) in
 map ($\lambda r. (\text{init-item } r \ k, \text{Null})$) *rs*

definition *Complete_L* :: nat \Rightarrow 'a item \Rightarrow 'a bins \Rightarrow nat \Rightarrow ('a item \times pointer) list **where**

Complete_L *k* *y* *bs* *red* \equiv
 let *orig* = *bs* ! (start-item *y*) in
 let *is* = filter-with-index ($\lambda x. \text{next-symbol } x = \text{Some } (\text{lhs-item } y)$) (*items orig*)
 in
 map ($\lambda(x, \text{pre}). (\text{inc-item } x \ k, \text{PreRed } (\text{start-item } y, \text{pre}, \text{red}) \ [])$) *is*

fun *upd-bin* :: 'a item \times pointer \Rightarrow 'a bin \Rightarrow 'a bin **where**

upd-bin *e'* [] = [*e'*]
 | *upd-bin* *e'* (*e*#*es*) = (
 case (*e'*, *e*) of
 ((*x*, *PreRed px xs*), (*y*, *PreRed py ys*)) \Rightarrow
 if *x* = *y* then (*x*, *PreRed py (px#xs@ys)*) # *es*
 else *e* # *upd-bin e' es*
 | - \Rightarrow
 if fst *e'* = fst *e* then *e* # *es*
 else *e* # *upd-bin e' es*)

fun *upds-bin* :: ('a item \times pointer) list \Rightarrow 'a bin \Rightarrow 'a bin **where**

upds-bin [] *b* = *b*
 | *upds-bin* (*e*#*es*) *b* = *upds-bin es (upd-bin e b)*

definition *upd-bins* :: 'a bins \Rightarrow nat \Rightarrow ('a item \times pointer) list \Rightarrow 'a bins **where**

$upd\text{-}bins\ bs\ k\ es \equiv bs[k := upds\text{-}bin\ es\ (bs!k)]$

partial-function (*tailrec*) $Earley_L\text{-}bin' :: nat \Rightarrow 'a\ cfg \Rightarrow 'a\ list \Rightarrow 'a\ bins \Rightarrow nat \Rightarrow 'a\ bins$ **where**
 $Earley_L\text{-}bin' k\ \mathcal{G}\ \omega\ bs\ i =$
 if $i \geq length\ (items\ (bs\ !\ k))$ then bs
 else
 let $x = items\ (bs!k)\ !\ i$ in
 let $bs' =$
 case *next-symbol* x of
 Some $a \Rightarrow$
 if $a \notin nonterminals\ \mathcal{G}$ then
 if $k < length\ \omega$ then $upd\text{-}bins\ bs\ (k+1)\ (Scan_L\ k\ \omega\ a\ x\ i)$
 else bs
 else $upd\text{-}bins\ bs\ k\ (Predict_L\ k\ \mathcal{G}\ a)$
 | None $\Rightarrow upd\text{-}bins\ bs\ k\ (Complete_L\ k\ x\ bs\ i)$
 in $Earley_L\text{-}bin' k\ \mathcal{G}\ \omega\ bs'\ (i+1)$

declare $Earley_L\text{-}bin'.simps[code]$

definition $Earley_L\text{-}bin :: nat \Rightarrow 'a\ cfg \Rightarrow 'a\ list \Rightarrow 'a\ bins \Rightarrow 'a\ bins$ **where**
 $Earley_L\text{-}bin\ k\ \mathcal{G}\ \omega\ bs = Earley_L\text{-}bin' k\ \mathcal{G}\ \omega\ bs\ 0$

fun $Earley_L\text{-}bins :: nat \Rightarrow 'a\ cfg \Rightarrow 'a\ list \Rightarrow 'a\ bins$ **where**
 $Earley_L\text{-}bins\ 0\ \mathcal{G}\ \omega = Earley_L\text{-}bin\ 0\ \mathcal{G}\ \omega\ (Init_L\ \mathcal{G}\ \omega)$
 | $Earley_L\text{-}bins\ (Suc\ n)\ \mathcal{G}\ \omega = Earley_L\text{-}bin\ (Suc\ n)\ \mathcal{G}\ \omega\ (Earley_L\text{-}bins\ n\ \mathcal{G}\ \omega)$

definition $Earley_L :: 'a\ cfg \Rightarrow 'a\ list \Rightarrow 'a\ bins$ **where**
 $Earley_L\ \mathcal{G}\ \omega \equiv Earley_L\text{-}bins\ (length\ \omega)\ \mathcal{G}\ \omega$

definition $recognizer :: 'a\ cfg \Rightarrow 'a\ list \Rightarrow bool$ **where**
 $recognizer\ \mathcal{G}\ \omega = (\exists x \in set\ (items\ (Earley_L\ \mathcal{G}\ \omega\ !\ length\ \omega)).\ is\text{-}finished\ \mathcal{G}\ \omega\ x)$

8.3 Epsilon productions

lemma ε -free-impl-non-empty-word-deriv:
 $\varepsilon\text{-free}\ \mathcal{G} \Rightarrow a \neq [] \Rightarrow \neg Derivation\ \mathcal{G}\ a\ D\ []$
 ⟨proof⟩

lemma ε -free-impl-nonempty-derives:
 $\varepsilon\text{-free}\ \mathcal{G} \Rightarrow nonempty\text{-derives}\ \mathcal{G}$
 ⟨proof⟩

lemma $nonempty\text{-derives}\text{-impl-}\varepsilon\text{-free}$:
 assumes $nonempty\text{-derives}\ \mathcal{G}$
 shows $\varepsilon\text{-free}\ \mathcal{G}$
 ⟨proof⟩

lemma $nonempty\text{-derives}\text{-iff-}\varepsilon\text{-free}$:

shows *nonempty-derives* $\mathcal{G} \longleftrightarrow \varepsilon\text{-free } \mathcal{G}$
<proof>

8.4 Bin lemmas

lemma *length-upd-bins[simp]*:
 $\text{length } (\text{upd-bins } bs \ k \ es) = \text{length } bs$
<proof>

lemma *length-upd-bin*:
 $\text{length } (\text{upd-bin } e \ b) \geq \text{length } b$
<proof>

lemma *length-upds-bin*:
 $\text{length } (\text{upds-bin } es \ b) \geq \text{length } b$
<proof>

lemma *length-nth-upd-bin-bins*:
 $\text{length } (\text{upd-bins } bs \ k \ es \ ! \ n) \geq \text{length } (bs \ ! \ n)$
<proof>

lemma *nth-idem-upd-bins*:
 $k \neq n \implies \text{upd-bins } bs \ k \ es \ ! \ n = bs \ ! \ n$
<proof>

lemma *items-nth-idem-upd-bin*:
 $n < \text{length } b \implies \text{items } (\text{upd-bin } e \ b) \ ! \ n = \text{items } b \ ! \ n$
<proof>

lemma *items-nth-idem-upds-bin*:
 $n < \text{length } b \implies \text{items } (\text{upds-bin } es \ b) \ ! \ n = \text{items } b \ ! \ n$
<proof>

lemma *items-nth-idem-upd-bins*:
 $n < \text{length } (bs \ ! \ k) \implies \text{items } (\text{upd-bins } bs \ k \ es \ ! \ k) \ ! \ n = \text{items } (bs \ ! \ k) \ ! \ n$
<proof>

lemma *bin-upto-eq-set-items*:
 $i \geq \text{length } b \implies \text{bin-upto } b \ i = \text{set } (\text{items } b)$
<proof>

lemma *bins-upto-empty*:
 $\text{bins-upto } bs \ 0 \ 0 = \{\}$
<proof>

lemma *set-items-upd-bin*:
 $\text{set } (\text{items } (\text{upd-bin } e \ b)) = \text{set } (\text{items } b) \cup \{\text{fst } e\}$
<proof>

lemma *set-items-upds-bin*:

$$\text{set } (\text{items } (\text{upds-bin } es \ b)) = \text{set } (\text{items } b) \cup \text{set } (\text{items } es)$$

<proof>

lemma *bins-upd-bins*:

assumes $k < \text{length } bs$

shows $\text{bins } (\text{upd-bins } bs \ k \ es) = \text{bins } bs \cup \text{set } (\text{items } es)$

<proof>

lemma *kth-bin-sub-bins*:

$k < \text{length } bs \implies \text{set } (\text{items } (bs \ ! \ k)) \subseteq \text{bins } bs$

<proof>

lemma *bin-upto-Cons-0*:

$\text{bin-upto } (e\#es) \ 0 = \{\}$

<proof>

lemma *bin-upto-Cons*:

assumes $0 < n$

shows $\text{bin-upto } (e\#es) \ n = \{\text{fst } e\} \cup \text{bin-upto } es \ (n-1)$

<proof>

lemma *bin-upto-nth-idem-upd-bin*:

$n < \text{length } b \implies \text{bin-upto } (\text{upd-bin } e \ b) \ n = \text{bin-upto } b \ n$

<proof>

lemma *bin-upto-nth-idem-upds-bin*:

$n < \text{length } b \implies \text{bin-upto } (\text{upds-bin } es \ b) \ n = \text{bin-upto } b \ n$

<proof>

lemma *bins-upto-kth-nth-idem*:

assumes $l < \text{length } bs \ k \leq l \ n < \text{length } (bs \ ! \ k)$

shows $\text{bins-upto } (\text{upd-bins } bs \ l \ es) \ k \ n = \text{bins-upto } bs \ k \ n$

<proof>

lemma *bins-upto-sub-bins*:

$k < \text{length } bs \implies \text{bins-upto } bs \ k \ n \subseteq \text{bins } bs$

<proof>

lemma *bins-upto-Suc-Un*:

$n < \text{length } (bs \ ! \ k) \implies \text{bins-upto } bs \ k \ (n+1) = \text{bins-upto } bs \ k \ n \cup \{\text{items } (bs \ ! \ k) \ ! \ n\}$

<proof>

lemma *bins-bin-exists*:

$x \in \text{bins } bs \implies \exists k < \text{length } bs. x \in \text{set } (\text{items } (bs \ ! \ k))$

<proof>

lemma *distinct-upd-bin*:

$distinct (items\ b) \implies distinct (items (upd-bin\ e\ b))$
<proof>

lemma *distinct-upds-bin:*

$distinct (items\ b) \implies distinct (items (upds-bin\ es\ b))$
<proof>

lemma *wf-bins-kth-bin:*

$wf-bins\ \mathcal{G}\ \omega\ bs \implies k < length\ bs \implies x \in set (items (bs\ !\ k)) \implies wf-item\ \mathcal{G}\ \omega\ x$
 $\wedge\ end-item\ x = k$
<proof>

lemma *wf-bin-upd-bin:*

assumes $wf-bin\ \mathcal{G}\ \omega\ k\ b\ wf-item\ \mathcal{G}\ \omega\ (fst\ e) \wedge end-item\ (fst\ e) = k$
shows $wf-bin\ \mathcal{G}\ \omega\ k\ (upd-bin\ e\ b)$
<proof>

lemma *wf-upd-bins-bin:*

assumes $wf-bin\ \mathcal{G}\ \omega\ k\ b$
assumes $\forall x \in set (items\ es). wf-item\ \mathcal{G}\ \omega\ x \wedge end-item\ x = k$
shows $wf-bin\ \mathcal{G}\ \omega\ k\ (upds-bin\ es\ b)$
<proof>

lemma *wf-bins-upd-bins:*

assumes $wf-bins\ \mathcal{G}\ \omega\ bs$
assumes $\forall x \in set (items\ es). wf-item\ \mathcal{G}\ \omega\ x \wedge end-item\ x = k$
shows $wf-bins\ \mathcal{G}\ \omega\ (upd-bins\ bs\ k\ es)$
<proof>

lemma *wf-bins-impl-wf-items:*

$wf-bins\ \mathcal{G}\ \omega\ bs \implies \forall x \in (bins\ bs). wf-item\ \mathcal{G}\ \omega\ x$
<proof>

lemma *upds-bin-eq-items:*

$set (items\ es) \subseteq set (items\ b) \implies set (items (upds-bin\ es\ b)) = set (items\ b)$
<proof>

lemma *bin-eq-items-upd-bin:*

$fst\ e \in set (items\ b) \implies items (upd-bin\ e\ b) = items\ b$
<proof>

lemma *bin-eq-items-upds-bin:*

assumes $set (items\ es) \subseteq set (items\ b)$
shows $items (upds-bin\ es\ b) = items\ b$
<proof>

lemma *bins-eq-items-upd-bins:*

assumes $set (items\ es) \subseteq set (items (bs!k))$
shows $bins-eq-items (upd-bins\ bs\ k\ es)\ bs$

<proof>

lemma *bins-eq-items-imp-eq-bins:*

bins-eq-items bs bs' \implies bins bs = bins bs'

<proof>

lemma *bin-eq-items-dist-upd-bin-bin:*

assumes *items a = items b*

shows *items (upd-bin e a) = items (upd-bin e b)*

<proof>

lemma *bin-eq-items-dist-upds-bin-bin:*

assumes *items a = items b*

shows *items (upds-bin es a) = items (upds-bin es b)*

<proof>

lemma *bin-eq-items-dist-upd-bin-entry:*

assumes *fst e = fst e'*

shows *items (upd-bin e b) = items (upd-bin e' b)*

<proof>

lemma *bin-eq-items-dist-upds-bin-entries:*

assumes *items es = items es'*

shows *items (upds-bin es b) = items (upds-bin es' b)*

<proof>

lemma *bins-eq-items-dist-upd-bins:*

assumes *bins-eq-items as bs items aes = items bes k < length as*

shows *bins-eq-items (upd-bins as k aes) (upd-bins bs k bes)*

<proof>

8.5 Well-formed bins

lemma *wf-bins-Scan_L':*

assumes *wf-bins \mathcal{G} ω bs k < length bs x \in set (items (bs ! k))*

assumes *k < length ω next-symbol x \neq None y = inc-item x (k+1)*

shows *wf-item \mathcal{G} ω y \wedge end-item y = k+1*

<proof>

lemma *wf-bins-Scan_L:*

assumes *wf-bins \mathcal{G} ω bs k < length bs x \in set (items (bs ! k)) k < length ω
next-symbol x \neq None*

shows $\forall y \in \text{set (items (Scan}_L k \omega a x \text{pre))}$. *wf-item \mathcal{G} ω y \wedge end-item y = (k+1)*

<proof>

lemma *wf-bins-Predict_L:*

assumes *wf-bins \mathcal{G} ω bs k < length bs k \leq length ω*

shows $\forall y \in \text{set (items (Predict}_L k \mathcal{G} X))$. *wf-item \mathcal{G} ω y \wedge end-item y = k*

<proof>

lemma *wf-item-inc-item*:

assumes *wf-item* $\mathcal{G} \ \omega \ x$ *next-symbol* $x = \text{Some } a$ *start-item* $x \leq k \ k \leq \text{length } \omega$
shows *wf-item* $\mathcal{G} \ \omega \ (\text{inc-item } x \ k) \wedge \text{end-item } (\text{inc-item } x \ k) = k$
<proof>

lemma *wf-bins-Complete_L*:

assumes *wf-bins* $\mathcal{G} \ \omega \ bs \ k < \text{length } bs \ y \in \text{set } (\text{items } (bs \ ! \ k))$
shows $\forall x \in \text{set } (\text{items } (\text{Complete}_L \ k \ y \ bs \ \text{red}))$. *wf-item* $\mathcal{G} \ \omega \ x \wedge \text{end-item } x = k$
<proof>

lemma *Ex-wf-bins*:

$\exists n \ bs \ \omega \ \mathcal{G}$. $n \leq \text{length } \omega \wedge \text{length } bs = \text{Suc } (\text{length } \omega) \wedge \text{wf-bins } \mathcal{G} \ \omega \ bs$
<proof>

definition *wf-earley-input* :: $(\text{nat} \times 'a \ \text{cfg} \times 'a \ \text{list} \times 'a \ \text{bins}) \ \text{set}$ **where**

wf-earley-input = {
 $(k, \mathcal{G}, \omega, bs) \mid k \ \mathcal{G} \ \omega \ bs$.
 $k \leq \text{length } \omega \wedge$
 $\text{length } bs = \text{length } \omega + 1 \wedge$
 $\text{wf-bins } \mathcal{G} \ \omega \ bs$
}

typedef $'a \ \text{wf-bins} = \text{wf-earley-input} :: (\text{nat} \times 'a \ \text{cfg} \times 'a \ \text{list} \times 'a \ \text{bins}) \ \text{set}$

morphisms *from-wf-bins to-wf-bins*
<proof>

lemma *wf-earley-input-elim*:

assumes $(k, \mathcal{G}, \omega, bs) \in \text{wf-earley-input}$
shows $k \leq \text{length } \omega \wedge k < \text{length } bs \wedge \text{length } bs = \text{length } \omega + 1 \wedge \text{wf-bins } \mathcal{G} \ \omega \ bs$
<proof>

lemma *wf-earley-input-intro*:

assumes $k \leq \text{length } \omega \ \text{length } bs = \text{length } \omega + 1 \ \text{wf-bins } \mathcal{G} \ \omega \ bs$
shows $(k, \mathcal{G}, \omega, bs) \in \text{wf-earley-input}$
<proof>

lemma *wf-earley-input-Complete_L*:

assumes $(k, \mathcal{G}, \omega, bs) \in \text{wf-earley-input} \ \neg \text{length } (\text{items } (bs \ ! \ k)) \leq i$
assumes $x = \text{items } (bs \ ! \ k) \ ! \ i \ \text{next-symbol } x = \text{None}$
shows $(k, \mathcal{G}, \omega, \text{upd-bins } bs \ k \ (\text{Complete}_L \ k \ x \ bs \ \text{red})) \in \text{wf-earley-input}$
<proof>

lemma *wf-earley-input-Scan_L*:

assumes $(k, \mathcal{G}, \omega, bs) \in \text{wf-earley-input} \ \neg \text{length } (\text{items } (bs \ ! \ k)) \leq i$
assumes $x = \text{items } (bs \ ! \ k) \ ! \ i \ \text{next-symbol } x = \text{Some } a$

assumes $k < \text{length } \omega$
shows $(k, \mathcal{G}, \omega, \text{upd-bins } bs \ (k+1) \ (\text{Scan}_L \ k \ \omega \ a \ x \ \text{pre})) \in \text{wf-earley-input}$
 $\langle \text{proof} \rangle$

lemma *wf-earley-input-Predict_L*:

assumes $(k, \mathcal{G}, \omega, bs) \in \text{wf-earley-input} \ \neg \text{length } (\text{items } (bs \ ! \ k)) \leq i$
assumes $x = \text{items } (bs \ ! \ k) \ ! \ i \ \text{next-symbol } x = \text{Some } a$
shows $(k, \mathcal{G}, \omega, \text{upd-bins } bs \ k \ (\text{Predict}_L \ k \ \mathcal{G} \ a)) \in \text{wf-earley-input}$
 $\langle \text{proof} \rangle$

fun *earley-measure* :: $\text{nat} \times 'a \ \text{cfg} \times 'a \ \text{list} \times 'a \ \text{bins} \Rightarrow \text{nat} \Rightarrow \text{nat}$ **where**
earley-measure $(k, \mathcal{G}, \omega, bs) \ i = \text{card } \{ x \mid x. \text{wf-item } \mathcal{G} \ \omega \ x \wedge \text{end-item } x = k \}$
 $- \ i$

lemma *Earley_L-bin'-simps[simp]*:

$i \geq \text{length } (\text{items } (bs \ ! \ k)) \Longrightarrow \text{Earley}_L\text{-bin}' \ k \ \mathcal{G} \ \omega \ bs \ i = bs$
 $\neg i \geq \text{length } (\text{items } (bs \ ! \ k)) \Longrightarrow x = \text{items } (bs!k) \ ! \ i \Longrightarrow \text{next-symbol } x = \text{None}$
 \Longrightarrow
 $\text{Earley}_L\text{-bin}' \ k \ \mathcal{G} \ \omega \ bs \ i = \text{Earley}_L\text{-bin}' \ k \ \mathcal{G} \ \omega \ (\text{upd-bins } bs \ k \ (\text{Complete}_L \ k \ x \ bs \ i)) \ (i+1)$
 $\neg i \geq \text{length } (\text{items } (bs \ ! \ k)) \Longrightarrow x = \text{items } (bs!k) \ ! \ i \Longrightarrow \text{next-symbol } x = \text{Some } a \Longrightarrow$
 $a \notin \text{nonterminals } \mathcal{G} \Longrightarrow k < \text{length } \omega \Longrightarrow \text{Earley}_L\text{-bin}' \ k \ \mathcal{G} \ \omega \ bs \ i = \text{Earley}_L\text{-bin}' \ k \ \mathcal{G} \ \omega \ (\text{upd-bins } bs \ (k+1) \ (\text{Scan}_L \ k \ \omega \ a \ x \ i)) \ (i+1)$
 $\neg i \geq \text{length } (\text{items } (bs \ ! \ k)) \Longrightarrow x = \text{items } (bs!k) \ ! \ i \Longrightarrow \text{next-symbol } x = \text{Some } a \Longrightarrow$
 $a \notin \text{nonterminals } \mathcal{G} \Longrightarrow \neg k < \text{length } \omega \Longrightarrow \text{Earley}_L\text{-bin}' \ k \ \mathcal{G} \ \omega \ bs \ i = \text{Earley}_L\text{-bin}' \ k \ \mathcal{G} \ \omega \ bs \ (i+1)$
 $\neg i \geq \text{length } (\text{items } (bs \ ! \ k)) \Longrightarrow x = \text{items } (bs!k) \ ! \ i \Longrightarrow \text{next-symbol } x = \text{Some } a \Longrightarrow$
 $a \in \text{nonterminals } \mathcal{G} \Longrightarrow \text{Earley}_L\text{-bin}' \ k \ \mathcal{G} \ \omega \ bs \ i = \text{Earley}_L\text{-bin}' \ k \ \mathcal{G} \ \omega \ (\text{upd-bins } bs \ k \ (\text{Predict}_L \ k \ \mathcal{G} \ a)) \ (i+1)$
 $\langle \text{proof} \rangle$

lemma *Earley_L-bin'-induct[case-names Base Complete_F Scan_F Pass Predict_F]*:

assumes $(k, \mathcal{G}, \omega, bs) \in \text{wf-earley-input}$
assumes *base*: $\bigwedge k \ \mathcal{G} \ \omega \ bs \ i. \ i \geq \text{length } (\text{items } (bs \ ! \ k)) \Longrightarrow P \ k \ \mathcal{G} \ \omega \ bs \ i$
assumes *complete*: $\bigwedge k \ \mathcal{G} \ \omega \ bs \ i \ x. \ \neg i \geq \text{length } (\text{items } (bs \ ! \ k)) \Longrightarrow x = \text{items } (bs \ ! \ k) \ ! \ i \Longrightarrow$
 $\text{next-symbol } x = \text{None} \Longrightarrow P \ k \ \mathcal{G} \ \omega \ (\text{upd-bins } bs \ k \ (\text{Complete}_L \ k \ x \ bs \ i))$
 $(i+1) \Longrightarrow P \ k \ \mathcal{G} \ \omega \ bs \ i$
assumes *scan*: $\bigwedge k \ \mathcal{G} \ \omega \ bs \ i \ x \ a. \ \neg i \geq \text{length } (\text{items } (bs \ ! \ k)) \Longrightarrow x = \text{items } (bs \ ! \ k) \ ! \ i \Longrightarrow$
 $\text{next-symbol } x = \text{Some } a \Longrightarrow a \notin \text{nonterminals } \mathcal{G} \Longrightarrow k < \text{length } \omega \Longrightarrow$
 $P \ k \ \mathcal{G} \ \omega \ (\text{upd-bins } bs \ (k+1) \ (\text{Scan}_L \ k \ \omega \ a \ x \ i)) \ (i+1) \Longrightarrow P \ k \ \mathcal{G} \ \omega \ bs \ i$
assumes *pass*: $\bigwedge k \ \mathcal{G} \ \omega \ bs \ i \ x \ a. \ \neg i \geq \text{length } (\text{items } (bs \ ! \ k)) \Longrightarrow x = \text{items } (bs \ ! \ k) \ ! \ i \Longrightarrow$
 $\text{next-symbol } x = \text{Some } a \Longrightarrow a \notin \text{nonterminals } \mathcal{G} \Longrightarrow \neg k < \text{length } \omega$
 \Longrightarrow

$P k \mathcal{G} \omega bs (i+1) \implies P k \mathcal{G} \omega bs i$
assumes $predict: \bigwedge k \mathcal{G} \omega bs i x a. \neg i \geq \text{length} (\text{items} (bs ! k)) \implies x = \text{items} (bs ! k) ! i \implies$
 $\text{next-symbol } x = \text{Some } a \implies a \in \text{nonterminals } \mathcal{G} \implies$
 $P k \mathcal{G} \omega (\text{upd-bins } bs k (\text{Predict}_L k \mathcal{G} a)) (i+1) \implies P k \mathcal{G} \omega bs i$
shows $P k \mathcal{G} \omega bs i$
 $\langle \text{proof} \rangle$

lemma *wf-earley-input-Earley_L-bin'*:
assumes $(k, \mathcal{G}, \omega, bs) \in \text{wf-earley-input}$
shows $(k, \mathcal{G}, \omega, \text{Earley}_L\text{-bin}' k \mathcal{G} \omega bs i) \in \text{wf-earley-input}$
 $\langle \text{proof} \rangle$

lemma *wf-earley-input-Earley_L-bin*:
assumes $(k, \mathcal{G}, \omega, bs) \in \text{wf-earley-input}$
shows $(k, \mathcal{G}, \omega, \text{Earley}_L\text{-bin } k \mathcal{G} \omega bs) \in \text{wf-earley-input}$
 $\langle \text{proof} \rangle$

lemma *length-bins-Earley_L-bin'*:
assumes $(k, \mathcal{G}, \omega, bs) \in \text{wf-earley-input}$
shows $\text{length} (\text{Earley}_L\text{-bin}' k \mathcal{G} \omega bs i) = \text{length } bs$
 $\langle \text{proof} \rangle$

lemma *length-nth-bin-Earley_L-bin'*:
assumes $(k, \mathcal{G}, \omega, bs) \in \text{wf-earley-input}$
shows $\text{length} (\text{items} (\text{Earley}_L\text{-bin}' k \mathcal{G} \omega bs i ! l)) \geq \text{length} (\text{items} (bs ! l))$
 $\langle \text{proof} \rangle$

lemma *wf-bins-Earley_L-bin'*:
assumes $(k, \mathcal{G}, \omega, bs) \in \text{wf-earley-input}$
shows $\text{wf-bins } \mathcal{G} \omega (\text{Earley}_L\text{-bin}' k \mathcal{G} \omega bs i)$
 $\langle \text{proof} \rangle$

lemma *wf-bins-Earley_L-bin*:
assumes $(k, \mathcal{G}, \omega, bs) \in \text{wf-earley-input}$
shows $\text{wf-bins } \mathcal{G} \omega (\text{Earley}_L\text{-bin } k \mathcal{G} \omega bs)$
 $\langle \text{proof} \rangle$

lemma *kth-Earley_L-bin'-bins*:
assumes $(k, \mathcal{G}, \omega, bs) \in \text{wf-earley-input}$
assumes $j < \text{length} (\text{items} (bs ! l))$
shows $\text{items} (\text{Earley}_L\text{-bin}' k \mathcal{G} \omega bs i ! l) ! j = \text{items} (bs ! l) ! j$
 $\langle \text{proof} \rangle$

lemma *nth-bin-sub-Earley_L-bin'*:
assumes $(k, \mathcal{G}, \omega, bs) \in \text{wf-earley-input}$
shows $\text{set} (\text{items} (bs ! l)) \subseteq \text{set} (\text{items} (\text{Earley}_L\text{-bin}' k \mathcal{G} \omega bs i ! l))$
 $\langle \text{proof} \rangle$

lemma *nth-Earley_L-bin'-eq*:

assumes $(k, \mathcal{G}, \omega, bs) \in \text{wf-earley-input}$

shows $l < k \implies \text{Earley}_L\text{-bin}' k \mathcal{G} \omega bs i ! l = bs ! l$

<proof>

lemma *set-items-Earley_L-bin'-eq*:

assumes $(k, \mathcal{G}, \omega, bs) \in \text{wf-earley-input}$

shows $l < k \implies \text{set} (\text{items} (\text{Earley}_L\text{-bin}' k \mathcal{G} \omega bs i ! l)) = \text{set} (\text{items} (bs ! l))$

<proof>

lemma *bins-upto-k0-Earley_L-bin'-eq*:

assumes $(k, \mathcal{G}, \omega, bs) \in \text{wf-earley-input}$

shows $\text{bins-upto} (\text{Earley}_L\text{-bin} k \mathcal{G} \omega bs) k 0 = \text{bins-upto} bs k 0$

<proof>

lemma *wf-earley-input-Init_L*:

assumes $k \leq \text{length } \omega$

shows $(k, \mathcal{G}, \omega, \text{Init}_L \mathcal{G} \omega) \in \text{wf-earley-input}$

<proof>

lemma *length-bins-Init_L[simp]*:

$\text{length} (\text{Init}_L \mathcal{G} \omega) = \text{length } \omega + 1$

<proof>

lemma *wf-earley-input-Earley_L-bins[simp]*:

assumes $k \leq \text{length } \omega$

shows $(k, \mathcal{G}, \omega, \text{Earley}_L\text{-bins } k \mathcal{G} \omega) \in \text{wf-earley-input}$

<proof>

lemma *length-Earley_L-bins[simp]*:

assumes $k \leq \text{length } \omega$

shows $\text{length} (\text{Earley}_L\text{-bins } k \mathcal{G} \omega) = \text{length} (\text{Init}_L \mathcal{G} \omega)$

<proof>

lemma *wf-bins-Earley_L-bins[simp]*:

assumes $k \leq \text{length } \omega$

shows $\text{wf-bins } \mathcal{G} \omega (\text{Earley}_L\text{-bins } k \mathcal{G} \omega)$

<proof>

lemma *wf-bins-Earley_L*:

$\text{wf-bins } \mathcal{G} \omega (\text{Earley}_L \mathcal{G} \omega)$

<proof>

8.6 Soundness

lemma *Init_L-eq-Init_F*:

$\text{bins} (\text{Init}_L \mathcal{G} \omega) = \text{Init}_F \mathcal{G}$

<proof>

lemma *Scan_L-sub-Scan_F*:

assumes *wf-bins* $\mathcal{G} \omega$ *bs bins* $bs \subseteq I$ $x \in \text{set}(\text{items}(bs ! k))$ $k < \text{length } bs$ $k < \text{length } \omega$

assumes *next-symbol* $x = \text{Some } a$

shows $\text{set}(\text{items}(\text{Scan}_L k \omega a x \text{pre})) \subseteq \text{Scan}_F k \omega I$

<proof>

lemma *Predict_L-sub-Predict_F*:

assumes *wf-bins* $\mathcal{G} \omega$ *bs bins* $bs \subseteq I$ $x \in \text{set}(\text{items}(bs ! k))$ $k < \text{length } bs$

assumes *next-symbol* $x = \text{Some } X$

shows $\text{set}(\text{items}(\text{Predict}_L k \mathcal{G} X)) \subseteq \text{Predict}_F k \mathcal{G} I$

<proof>

lemma *Complete_L-sub-Complete_F*:

assumes *wf-bins* $\mathcal{G} \omega$ *bs bins* $bs \subseteq I$ $y \in \text{set}(\text{items}(bs ! k))$ $k < \text{length } bs$

assumes *next-symbol* $y = \text{None}$

shows $\text{set}(\text{items}(\text{Complete}_L k y bs \text{red})) \subseteq \text{Complete}_F k I$

<proof>

lemma *sound-Scan_L*:

assumes *wf-bins* $\mathcal{G} \omega$ *bs bins* $bs \subseteq I$ $x \in \text{set}(\text{items}(bs!k))$ $k < \text{length } bs$ $k < \text{length } \omega$

assumes *next-symbol* $x = \text{Some } a$ $\forall x \in I. \text{wf-item } \mathcal{G} \omega x$ $\forall x \in I. \text{sound-item } \mathcal{G} \omega x$

shows $\forall x \in \text{set}(\text{items}(\text{Scan}_L k \omega a x i)). \text{sound-item } \mathcal{G} \omega x$

<proof>

lemma *sound-Predict_L*:

assumes *wf-bins* $\mathcal{G} \omega$ *bs bins* $bs \subseteq I$ $x \in \text{set}(\text{items}(bs!k))$ $k < \text{length } bs$

assumes *next-symbol* $x = \text{Some } X$ $\forall x \in I. \text{wf-item } \mathcal{G} \omega x$ $\forall x \in I. \text{sound-item } \mathcal{G} \omega x$

shows $\forall x \in \text{set}(\text{items}(\text{Predict}_L k \mathcal{G} X)). \text{sound-item } \mathcal{G} \omega x$

<proof>

lemma *sound-Complete_L*:

assumes *wf-bins* $\mathcal{G} \omega$ *bs bins* $bs \subseteq I$ $y \in \text{set}(\text{items}(bs!k))$ $k < \text{length } bs$

assumes *next-symbol* $y = \text{None}$ $\forall x \in I. \text{wf-item } \mathcal{G} \omega x$ $\forall x \in I. \text{sound-item } \mathcal{G} \omega x$

shows $\forall x \in \text{set}(\text{items}(\text{Complete}_L k y bs i)). \text{sound-item } \mathcal{G} \omega x$

<proof>

lemma *sound-Earley_L-bin'*:

assumes $(k, \mathcal{G}, \omega, bs) \in \text{wf-earley-input}$

assumes $\forall x \in \text{bins } bs. \text{sound-item } \mathcal{G} \omega x$

shows $\forall x \in \text{bins}(\text{Earley}_L\text{-bin}' k \mathcal{G} \omega bs i). \text{sound-item } \mathcal{G} \omega x$

<proof>

lemma *sound-Earley_L-bin*:

assumes $(k, \mathcal{G}, \omega, bs) \in \text{wf-earley-input}$

assumes $\forall x \in \text{bins } bs. \text{ sound-item } \mathcal{G} \ \omega \ x$
shows $\forall x \in \text{bins } (\text{Earley}_L\text{-bin } k \ \mathcal{G} \ \omega \ bs). \text{ sound-item } \mathcal{G} \ \omega \ x$
 $\langle \text{proof} \rangle$

lemma *Earley_L-bin'-sub-Earley_F-bin:*
assumes $(k, \mathcal{G}, \omega, bs) \in \text{wf-earley-input}$
assumes $\text{bins } bs \subseteq I$
shows $\text{bins } (\text{Earley}_L\text{-bin}' k \ \mathcal{G} \ \omega \ bs \ i) \subseteq \text{Earley}_F\text{-bin } k \ \mathcal{G} \ \omega \ I$
 $\langle \text{proof} \rangle$

lemma *Earley_L-bin-sub-Earley_F-bin:*
assumes $(k, \mathcal{G}, \omega, bs) \in \text{wf-earley-input}$
assumes $\text{bins } bs \subseteq I$
shows $\text{bins } (\text{Earley}_L\text{-bin } k \ \mathcal{G} \ \omega \ bs) \subseteq \text{Earley}_F\text{-bin } k \ \mathcal{G} \ \omega \ I$
 $\langle \text{proof} \rangle$

lemma *Earley_L-bins-sub-Earley_F-bins:*
assumes $k \leq \text{length } \omega$
shows $\text{bins } (\text{Earley}_L\text{-bins } k \ \mathcal{G} \ \omega) \subseteq \text{Earley}_F\text{-bins } k \ \mathcal{G} \ \omega$
 $\langle \text{proof} \rangle$

lemma *Earley_L-sub-Earley_F:*
 $\text{bins } (\text{Earley}_L \ \mathcal{G} \ \omega) \subseteq \text{Earley}_F \ \mathcal{G} \ \omega$
 $\langle \text{proof} \rangle$

theorem *soundness-Earley_L:*
assumes *recognizing* $(\text{bins } (\text{Earley}_L \ \mathcal{G} \ \omega)) \ \mathcal{G} \ \omega$
shows $\mathcal{G} \vdash [\mathcal{G}] \Rightarrow^* \omega$
 $\langle \text{proof} \rangle$

8.7 Completeness

lemma *bin-bins-upto-bins-eq:*
assumes $\text{wf-bins } \mathcal{G} \ \omega \ bs \ k < \text{length } bs \ i \geq \text{length } (\text{items } (bs \ ! \ k)) \ l \leq k$
shows $\text{bin } (\text{bins-upto } bs \ k \ i) \ l = \text{bin } (\text{bins } bs) \ l$
 $\langle \text{proof} \rangle$

lemma *impossible-complete-item:*
assumes *sound-item* $\mathcal{G} \ \omega \ x$ *is-complete* x *start-item* $x = k$ *end-item* $x = k$
nonempty-derives \mathcal{G}
shows *False*
 $\langle \text{proof} \rangle$

lemma *Complete_F-Un-eq-terminal:*
assumes *next-symbol* $z = \text{Some } a \ a \notin \text{nonterminals } \mathcal{G} \ \forall x \in I. \text{wf-item } \mathcal{G} \ \omega \ x$
wf-item $\mathcal{G} \ \omega \ z$
shows $\text{Complete}_F \ k \ (I \cup \{z\}) = \text{Complete}_F \ k \ I$
 $\langle \text{proof} \rangle$

lemma *Complete_F-Un-eq-nonterminal:*

assumes $\forall x \in I. \text{wf-item } \mathcal{G} \ \omega \ x \ \forall x \in I. \text{sound-item } \mathcal{G} \ \omega \ x$

assumes *nonempty-derives* $\mathcal{G} \ \text{wf-item } \mathcal{G} \ \omega \ z$

assumes *end-item* $z = k \ \text{next-symbol } z \neq \text{None}$

shows $\text{Complete}_F \ k \ (I \cup \{z\}) = \text{Complete}_F \ k \ I$

<proof>

lemma *wf-item-in-kth-bin:*

wf-bins $\mathcal{G} \ \omega \ bs \implies x \in \text{bins } bs \implies \text{end-item } x = k \implies x \in \text{set } (\text{items } (bs \ ! \ k))$

<proof>

lemma *Complete_F-bins-upto-eq-bins:*

assumes *wf-bins* $\mathcal{G} \ \omega \ bs \ k < \text{length } bs \ i \geq \text{length } (\text{items } (bs \ ! \ k))$

shows $\text{Complete}_F \ k \ (\text{bins-upto } bs \ k \ i) = \text{Complete}_F \ k \ (\text{bins } bs)$

<proof>

lemma *Complete_F-sub-bins-Un-Complete_L:*

assumes $\text{Complete}_F \ k \ I \subseteq \text{bins } bs \ I \subseteq \text{bins } bs \ \text{is-complete } z \ \text{wf-bins } \mathcal{G} \ \omega \ bs$
wf-item $\mathcal{G} \ \omega \ z$

shows $\text{Complete}_F \ k \ (I \cup \{z\}) \subseteq \text{bins } bs \cup \text{set } (\text{items } (\text{Complete}_L \ k \ z \ bs \ \text{red}))$

<proof>

lemma *Complete_L-eq-start-item:*

bs ! start-item $y = bs' \ ! \ \text{start-item } y \implies \text{Complete}_L \ k \ y \ bs \ \text{red} = \text{Complete}_L \ k \ y \ bs' \ \text{red}$

<proof>

lemma *kth-bin-bins-upto-empty:*

assumes *wf-bins* $\mathcal{G} \ \omega \ bs \ k < \text{length } bs$

shows $\text{bin } (\text{bins-upto } bs \ k \ 0) \ k = \{\}$

<proof>

lemma *Earley_L-bin'-mono:*

assumes $(k, \mathcal{G}, \omega, bs) \in \text{wf-earley-input}$

shows $\text{bins } bs \subseteq \text{bins } (\text{Earley}_L\text{-bin}' \ k \ \mathcal{G} \ \omega \ bs \ i)$

<proof>

lemma *Earley_F-bin-step-sub-Earley_L-bin':*

assumes $(k, \mathcal{G}, \omega, bs) \in \text{wf-earley-input}$

assumes *Earley_F-bin-step* $k \ \mathcal{G} \ \omega \ (\text{bins-upto } bs \ k \ i) \subseteq \text{bins } bs$

assumes $\forall x \in \text{bins } bs. \ \text{sound-item } \mathcal{G} \ \omega \ x \ \text{is-word } \mathcal{G} \ \omega \ \text{nonempty-derives } \mathcal{G}$

shows *Earley_F-bin-step* $k \ \mathcal{G} \ \omega \ (\text{bins } bs) \subseteq \text{bins } (\text{Earley}_L\text{-bin}' \ k \ \mathcal{G} \ \omega \ bs \ i)$

<proof>

lemma *Earley_F-bin-step-sub-Earley_L-bin:*

assumes $(k, \mathcal{G}, \omega, bs) \in \text{wf-earley-input}$

assumes *Earley_F-bin-step* $k \ \mathcal{G} \ \omega \ (\text{bins-upto } bs \ k \ 0) \subseteq \text{bins } bs$

assumes $\forall x \in \text{bins } bs. \ \text{sound-item } \mathcal{G} \ \omega \ x \ \text{is-word } \mathcal{G} \ \omega \ \text{nonempty-derives } \mathcal{G}$

shows *Earley_F-bin-step* $k \ \mathcal{G} \ \omega \ (\text{bins } bs) \subseteq \text{bins } (\text{Earley}_L\text{-bin } k \ \mathcal{G} \ \omega \ bs)$

<proof>

lemma *bins-eq-items-Complete_L*:

assumes *bins-eq-items as bs start-item x < length as*

shows *items (Complete_L k x as i) = items (Complete_L k x bs i)*

<proof>

lemma *Earley_L-bin'-bins-eq*:

assumes *(k, G, ω, as) ∈ wf-earley-input*

assumes *bins-eq-items as bs wf-bins G ω as*

shows *bins-eq-items (Earley_L-bin' k G ω as i) (Earley_L-bin' k G ω bs i)*

<proof>

lemma *Earley_L-bin'-idem*:

assumes *(k, G, ω, bs) ∈ wf-earley-input*

assumes *i ≤ j ∀ x ∈ bins bs. sound-item G ω x nonempty-derives G*

shows *bins (Earley_L-bin' k G ω (Earley_L-bin' k G ω bs i) j) = bins (Earley_L-bin' k G ω bs i)*

<proof>

lemma *Earley_L-bin-idem*:

assumes *(k, G, ω, bs) ∈ wf-earley-input*

assumes *∀ x ∈ bins bs. sound-item G ω x nonempty-derives G*

shows *bins (Earley_L-bin k G ω (Earley_L-bin k G ω bs)) = bins (Earley_L-bin k G ω bs)*

<proof>

lemma *funpower-Earley_F-bin-step-sub-Earley_L-bin*:

assumes *(k, G, ω, bs) ∈ wf-earley-input*

assumes *Earley_F-bin-step k G ω (bins-upto bs k 0) ⊆ bins bs ∀ x ∈ bins bs. sound-item G ω x*

assumes *is-word G ω nonempty-derives G*

shows *funpower (Earley_F-bin-step k G ω) n (bins bs) ⊆ bins (Earley_L-bin k G ω bs)*

<proof>

lemma *Earley_F-bin-sub-Earley_L-bin*:

assumes *(k, G, ω, bs) ∈ wf-earley-input*

assumes *Earley_F-bin-step k G ω (bins-upto bs k 0) ⊆ bins bs ∀ x ∈ bins bs. sound-item G ω x*

assumes *is-word G ω nonempty-derives G*

shows *Earley_F-bin k G ω (bins bs) ⊆ bins (Earley_L-bin k G ω bs)*

<proof>

lemma *Earley_F-bins-sub-Earley_L-bins*:

assumes *k ≤ length ω*

assumes *is-word G ω nonempty-derives G*

shows *Earley_F-bins k G ω ⊆ bins (Earley_L-bins k G ω)*

<proof>

lemma *Earley_F-sub-Earley_L*:
assumes *is-word* \mathcal{G} ω ε -free \mathcal{G}
shows $\text{Earley}_F \mathcal{G} \omega \subseteq \text{bins} (\text{Earley}_L \mathcal{G} \omega)$
<proof>

theorem *completeness-Earley_L*:
assumes $\mathcal{G} \vdash [\mathfrak{G} \mathcal{G}] \Rightarrow^* \omega$ *is-word* \mathcal{G} ω ε -free \mathcal{G}
shows *recognizing* ($\text{bins} (\text{Earley}_L \mathcal{G} \omega)$) $\mathcal{G} \omega$
<proof>

8.8 Correctness

theorem *Earley-eq-Earley_L*:
assumes *is-word* \mathcal{G} ω ε -free \mathcal{G}
shows $\text{Earley} \mathcal{G} \omega = \text{bins} (\text{Earley}_L \mathcal{G} \omega)$
<proof>

lemma *correctness-recognizer*:
assumes *is-word* \mathcal{G} ω ε -free \mathcal{G}
shows *recognizer* $\mathcal{G} \omega \longleftrightarrow \mathcal{G} \vdash [\mathfrak{G} \mathcal{G}] \Rightarrow^* \omega$ (**is** $?L \longleftrightarrow ?R$)
<proof>

end

theory *Earley-Parser*
imports
Earley-Recognizer
HOL-Library.Monad-Syntax
begin

9 Earley parser

9.1 Pointer lemmas

definition *predicts* :: 'a item \Rightarrow bool **where**
predicts $x \equiv \text{start-item } x = \text{end-item } x \wedge \text{dot-item } x = 0$

definition *scans* :: 'a list \Rightarrow nat \Rightarrow 'a item \Rightarrow 'a item \Rightarrow bool **where**
scans ω k x $y \equiv y = \text{inc-item } x$ $k \wedge (\exists a. \text{next-symbol } x = \text{Some } a \wedge \omega!(k-1) = a)$

definition *completes* :: nat \Rightarrow 'a item \Rightarrow 'a item \Rightarrow 'a item \Rightarrow bool **where**
completes k x y $z \equiv y = \text{inc-item } x$ $k \wedge \text{is-complete } z \wedge \text{start-item } z = \text{end-item } x \wedge$
 $(\exists N. \text{next-symbol } x = \text{Some } N \wedge N = \text{lhs-item } z)$

definition *sound-null-ptr* :: 'a item \times pointer \Rightarrow bool **where**
sound-null-ptr $e \equiv (\text{snd } e = \text{Null} \longrightarrow \text{predicts } (\text{fst } e))$

definition *sound-pre-ptr* :: 'a list \Rightarrow 'a bins \Rightarrow nat \Rightarrow 'a item \times pointer \Rightarrow bool
where

sound-pre-ptr ω bs k e $\equiv \forall$ pre. snd e = Pre pre \longrightarrow
k > 0 \wedge pre < length (bs!(k-1)) \wedge scans ω k (fst (bs!(k-1)!pre)) (fst e)

definition *sound-prered-ptr* :: 'a bins \Rightarrow nat \Rightarrow 'a item \times pointer \Rightarrow bool **where**

sound-prered-ptr bs k e $\equiv \forall$ p ps k' pre red. snd e = PreRed p ps \wedge (k', pre, red) \in set (p#ps) \longrightarrow
k' < k \wedge pre < length (bs!k') \wedge red < length (bs!k) \wedge completes k (fst (bs!k'!pre))
(fst e) (fst (bs!k!red))

definition *sound-ptrs* :: 'a list \Rightarrow 'a bins \Rightarrow bool **where**

sound-ptrs ω bs $\equiv \forall$ k < length bs. \forall e \in set (bs!k).
sound-null-ptr e \wedge *sound-pre-ptr* ω bs k e \wedge *sound-prered-ptr* bs k e

definition *mono-red-ptr* :: 'a bins \Rightarrow bool **where**

mono-red-ptr bs $\equiv \forall$ k < length bs. \forall i < length (bs!k).
 \forall k' pre red ps. snd (bs!k'i) = PreRed (k', pre, red) ps \longrightarrow red < i

lemma *nth-item-upd-bin*:

n < length es \implies fst (upd-bin e es ! n) = fst (es!n)
<proof>

lemma *upd-bin-append*:

fst e \notin set (items es) \implies upd-bin e es = es @ [e]
<proof>

lemma *upd-bin-null-pre*:

fst e \in set (items es) \implies snd e = Null \vee snd e = Pre pre \implies upd-bin e es = es
<proof>

lemma *upd-bin-prered-nop*:

assumes distinct (items es) i < length es
assumes fst e = fst (es!i) snd e = PreRed p ps \nexists p ps. snd (es!i) = PreRed p ps
shows upd-bin e es = es
<proof>

lemma *upd-bin-prered-upd*:

assumes distinct (items es) i < length es
assumes fst e = fst (es!i) snd e = PreRed p rs snd (es!i) = PreRed p' rs' upd-bin
e es = es'
shows snd (es!i) = PreRed p' (p#rs@rs') \wedge (\forall j < length es'. i \neq j \longrightarrow es!j =
es!j) \wedge length (upd-bin e es) = length es
<proof>

lemma *sound-ptrs-upd-bin*:

assumes *sound-ptrs* ω bs k < length bs es = bs!k distinct (items es)
assumes *sound-null-ptr* e *sound-pre-ptr* ω bs k e *sound-prered-ptr* bs k e
shows *sound-ptrs* ω (bs[k := upd-bin e es])

<proof>

lemma *mono-red-ptr-upd-bin:*

assumes *mono-red-ptr* $bs\ k < \text{length}\ bs\ es = bs!k\ \text{distinct}\ (\text{items}\ es)$

assumes $\forall k'\ pre\ red\ ps.\ \text{snd}\ e = \text{PreRed}\ (k',\ pre,\ red)\ ps \longrightarrow red < \text{length}\ es$

shows *mono-red-ptr* $(bs[k := \text{upd-bin}\ e\ es])$

<proof>

lemma *sound-mono-ptrs-upds-bin:*

assumes *sound-ptrs* $\omega\ bs\ \text{mono-red-ptr}\ bs\ k < \text{length}\ bs\ b = bs!k\ \text{distinct}\ (\text{items}\ b)$

assumes $\forall e \in \text{set}\ es.\ \text{sound-null-ptr}\ e \wedge \text{sound-pre-ptr}\ \omega\ bs\ k\ e \wedge \text{sound-prered-ptr}\ bs\ k\ e$

assumes $\forall e \in \text{set}\ es.\ \forall k'\ pre\ red\ ps.\ \text{snd}\ e = \text{PreRed}\ (k',\ pre,\ red)\ ps \longrightarrow red < \text{length}\ b$

shows *sound-ptrs* $\omega\ (bs[k := \text{upds-bin}\ es\ b]) \wedge \text{mono-red-ptr}\ (bs[k := \text{upds-bin}\ es\ b])$

<proof>

lemma *sound-mono-ptrs-Earley_L-bin':*

assumes $(k,\ \mathcal{G},\ \omega,\ bs) \in \text{wf-earley-input}$

assumes *sound-ptrs* $\omega\ bs\ \forall x \in \text{bins}\ bs.\ \text{sound-item}\ \mathcal{G}\ \omega\ x$

assumes *mono-red-ptr* bs

assumes *nonempty-derives* \mathcal{G}

shows *sound-ptrs* $\omega\ (\text{Earley}_L\text{-bin}'\ k\ \mathcal{G}\ \omega\ bs\ i) \wedge \text{mono-red-ptr}\ (\text{Earley}_L\text{-bin}'\ k\ \mathcal{G}\ \omega\ bs\ i)$

<proof>

lemma *sound-mono-ptrs-Earley_L-bin:*

assumes $(k,\ \mathcal{G},\ \omega,\ bs) \in \text{wf-earley-input}$

assumes *sound-ptrs* $\omega\ bs\ \forall x \in \text{bins}\ bs.\ \text{sound-item}\ \mathcal{G}\ \omega\ x$

assumes *mono-red-ptr* bs

assumes *nonempty-derives* \mathcal{G}

shows *sound-ptrs* $\omega\ (\text{Earley}_L\text{-bin}\ k\ \mathcal{G}\ \omega\ bs) \wedge \text{mono-red-ptr}\ (\text{Earley}_L\text{-bin}\ k\ \mathcal{G}\ \omega\ bs)$

<proof>

lemma *sound-ptrs-Init_L:*

sound-ptrs $\omega\ (\text{Init}_L\ \mathcal{G}\ \omega)$

<proof>

lemma *mono-red-ptr-Init_L:*

mono-red-ptr $(\text{Init}_L\ \mathcal{G}\ \omega)$

<proof>

lemma *sound-mono-ptrs-Earley_L-bins:*

assumes $k \leq \text{length}\ \omega\ \text{nonempty-derives}\ \mathcal{G}$

shows *sound-ptrs* $\omega\ (\text{Earley}_L\text{-bins}\ k\ \mathcal{G}\ \omega) \wedge \text{mono-red-ptr}\ (\text{Earley}_L\text{-bins}\ k\ \mathcal{G}\ \omega)$

<proof>

lemma *sound-mono-ptrs-Earley_L*:
assumes *nonempty-derives* \mathcal{G}
shows *sound-ptrs* ω (*Earley_L* \mathcal{G} ω) \wedge *mono-red-ptr* (*Earley_L* \mathcal{G} ω)
 \langle *proof* \rangle

9.2 Common Definitions

datatype *'a tree* =
Leaf *'a*
| *Branch* *'a 'a tree list*

fun *yield* :: *'a tree* \Rightarrow *'a list* **where**
yield (*Leaf* *a*) = [*a*]
| *yield* (*Branch* - *ts*) = *concat* (*map yield ts*)

fun *root* :: *'a tree* \Rightarrow *'a* **where**
root (*Leaf* *a*) = *a*
| *root* (*Branch* *N* -) = *N*

fun *wf-rule-tree* :: *'a cfg* \Rightarrow *'a tree* \Rightarrow *bool* **where**
wf-rule-tree - (*Leaf* *a*) \longleftrightarrow *True*
| *wf-rule-tree* \mathcal{G} (*Branch* *N* *ts*) \longleftrightarrow (
 $(\exists r \in \text{set } (\mathfrak{R} \mathcal{G}). N = \text{lhs-rule } r \wedge \text{map root } ts = \text{rhs-rule } r) \wedge$
 $(\forall t \in \text{set } ts. \text{wf-rule-tree } \mathcal{G} t)$)

fun *wf-item-tree* :: *'a cfg* \Rightarrow *'a item* \Rightarrow *'a tree* \Rightarrow *bool* **where**
wf-item-tree \mathcal{G} - (*Leaf* *a*) \longleftrightarrow *True*
| *wf-item-tree* \mathcal{G} *x* (*Branch* *N* *ts*) \longleftrightarrow (
 $N = \text{lhs-item } x \wedge \text{map root } ts = \text{take } (\text{dot-item } x) (\text{rhs-item } x) \wedge$
 $(\forall t \in \text{set } ts. \text{wf-rule-tree } \mathcal{G} t)$)

definition *wf-yield* :: *'a list* \Rightarrow *'a item* \Rightarrow *'a tree* \Rightarrow *bool* **where**
wf-yield ω *x* *t* \longleftrightarrow *yield* *t* = *slice* ω (*start-item* *x*) (*end-item* *x*)

9.3 foldl lemmas

lemma *foldl-add-nth*:
 $k < \text{length } xs \implies \text{foldl } (+) z (\text{map length } (\text{take } k \text{ } xs)) + \text{length } (xs!k) = \text{foldl}$
 $(+) z (\text{map length } (\text{take } (k+1) \text{ } xs))$
 \langle *proof* \rangle

lemma *foldl-acc-mono*:
 $a \leq b \implies \text{foldl } (+) a \text{ } xs \leq \text{foldl } (+) b \text{ } xs$ **for** $a :: \text{nat}$
 \langle *proof* \rangle

lemma *foldl-ge-z-nth*:
 $j < \text{length } xs \implies z + \text{length } (xs!j) \leq \text{foldl } (+) z (\text{map length } (\text{take } (j+1) \text{ } xs))$
 \langle *proof* \rangle

lemma *foldl-add-nth-ge*:

$i \leq j \implies j < \text{length } xs \implies \text{foldl } (+) z (\text{map length } (\text{take } i \text{ } xs)) + \text{length } (xs!j)$
 $\leq \text{foldl } (+) z (\text{map length } (\text{take } (j+1) \text{ } xs))$
 ⟨proof⟩

lemma *foldl-ge-acc*:

$\text{foldl } (+) z (\text{map length } xs) \geq z$
 ⟨proof⟩

lemma *foldl-take-mono*:

$i \leq j \implies \text{foldl } (+) z (\text{map length } (\text{take } i \text{ } xs)) \leq \text{foldl } (+) z (\text{map length } (\text{take } j \text{ } xs))$
 ⟨proof⟩

9.4 Parse tree

partial-function (*option*) *build-tree'* :: 'a bins \Rightarrow 'a list \Rightarrow nat \Rightarrow nat \Rightarrow 'a tree
option where

build-tree' bs ω k i = (
 let e = bs!k!i in (
 case snd e of
 | Null \Rightarrow Some (Branch (lhs-item (fst e)) []) — start building sub-tree
 | Pre pre \Rightarrow (— add sub-tree starting from terminal
 do {
 t \leftarrow *build-tree'* bs ω (k-1) pre;
 case t of
 | Branch N ts \Rightarrow Some (Branch N (ts @ [Leaf ($\omega!(k-1)$)]))
 | - \Rightarrow undefined — impossible case
 })
 | PreRed (k', pre, red) - \Rightarrow (— add sub-tree starting from non-terminal
 do {
 t \leftarrow *build-tree'* bs ω k' pre;
 case t of
 | Branch N ts \Rightarrow
 do {
 t \leftarrow *build-tree'* bs ω k red;
 Some (Branch N (ts @ [t]))
 }
 | - \Rightarrow undefined — impossible case
 })
))

declare *build-tree'.simps* [code]

definition *build-tree* :: 'a cfg \Rightarrow 'a list \Rightarrow 'a bins \Rightarrow 'a tree **option where**

build-tree \mathcal{G} ω bs = (
 let k = length bs - 1 in (
 case filter-with-index ($\lambda x. \text{is-finished } \mathcal{G} \omega x$) (items (bs!k)) of
 [] \Rightarrow None

| (-, i)#- => build-tree' bs ω k i
))

lemma build-tree'-simps[simp]:

$e = bs!k!i \implies snd\ e = Null \implies build-tree'\ bs\ \omega\ k\ i = Some\ (Branch\ (lhs-item\ (fst\ e))\ [])$

$e = bs!k!i \implies snd\ e = Pre\ pre \implies build-tree'\ bs\ \omega\ (k-1)\ pre = None \implies build-tree'\ bs\ \omega\ k\ i = None$

$e = bs!k!i \implies snd\ e = Pre\ pre \implies build-tree'\ bs\ \omega\ (k-1)\ pre = Some\ (Branch\ N\ ts) \implies$

$build-tree'\ bs\ \omega\ k\ i = Some\ (Branch\ N\ (ts\ @\ [Leaf\ (\omega!(k-1))]))$

$e = bs!k!i \implies snd\ e = Pre\ pre \implies build-tree'\ bs\ \omega\ (k-1)\ pre = Some\ (Leaf\ a) \implies$

$build-tree'\ bs\ \omega\ k\ i = undefined$

$e = bs!k!i \implies snd\ e = PreRed\ (k',\ pre,\ red)\ reds \implies build-tree'\ bs\ \omega\ k'\ pre = None \implies$

$build-tree'\ bs\ \omega\ k\ i = None$

$e = bs!k!i \implies snd\ e = PreRed\ (k',\ pre,\ red)\ reds \implies build-tree'\ bs\ \omega\ k'\ pre = Some\ (Branch\ N\ ts) \implies$

$build-tree'\ bs\ \omega\ k\ red = None \implies build-tree'\ bs\ \omega\ k\ i = None$

$e = bs!k!i \implies snd\ e = PreRed\ (k',\ pre,\ red)\ reds \implies build-tree'\ bs\ \omega\ k'\ pre = Some\ (Leaf\ a) \implies$

$build-tree'\ bs\ \omega\ k\ i = undefined$

$e = bs!k!i \implies snd\ e = PreRed\ (k',\ pre,\ red)\ reds \implies build-tree'\ bs\ \omega\ k'\ pre = Some\ (Branch\ N\ ts) \implies$

$build-tree'\ bs\ \omega\ k\ red = Some\ t \implies$

$build-tree'\ bs\ \omega\ k\ i = Some\ (Branch\ N\ (ts\ @\ [t]))$

<proof>

definition wf-tree-input :: ('a bins × 'a list × nat × nat) set **where**

wf-tree-input = {
 (bs, ω, k, i) | bs ω k i.
 sound-ptrs ω bs ∧
 mono-red-ptr bs ∧
 k < length bs ∧
 k ≤ length ω ∧
 i < length (bs!k)
 }

fun build-tree'-measure :: ('a bins × 'a list × nat × nat) => nat **where**

build-tree'-measure (bs, ω, k, i) = foldl (+) 0 (map length (take k bs)) + i

lemma wf-tree-input-pre:

assumes (bs, ω, k, i) ∈ wf-tree-input

assumes e = bs!k!i snd e = Pre pre

shows (bs, ω, (k-1), pre) ∈ wf-tree-input

<proof>

lemma wf-tree-input-prered-pre:

assumes $(bs, \omega, k, i) \in wf\text{-tree-input}$
assumes $e = bs!k!i$ *snd* $e = PreRed (k', pre, red)$ *ps*
shows $(bs, \omega, k', pre) \in wf\text{-tree-input}$
 <proof>

lemma *wf-tree-input-prered-red:*

assumes $(bs, \omega, k, i) \in wf\text{-tree-input}$
assumes $e = bs!k!i$ *snd* $e = PreRed (k', pre, red)$ *ps*
shows $(bs, \omega, k, red) \in wf\text{-tree-input}$
 <proof>

lemma *build-tree'-induct:*

assumes $(bs, \omega, k, i) \in wf\text{-tree-input}$
assumes $\bigwedge bs \ \omega \ k \ i.$
 $(\bigwedge e \ pre. \ e = bs!k!i \implies \text{snd } e = Pre \ pre \implies P \ bs \ \omega \ (k-1) \ pre) \implies$
 $(\bigwedge e \ k' \ pre \ red \ ps. \ e = bs!k!i \implies \text{snd } e = PreRed (k', pre, red) \ ps \implies P \ bs \ \omega$
 $k' \ pre) \implies$
 $(\bigwedge e \ k' \ pre \ red \ ps. \ e = bs!k!i \implies \text{snd } e = PreRed (k', pre, red) \ ps \implies P \ bs \ \omega$
 $k \ red) \implies$
 $P \ bs \ \omega \ k \ i$
shows $P \ bs \ \omega \ k \ i$
 <proof>

lemma *build-tree'-termination:*

assumes $(bs, \omega, k, i) \in wf\text{-tree-input}$
shows $\exists N \ ts. \ build\text{-tree}' \ bs \ \omega \ k \ i = Some \ (Branch \ N \ ts)$
 <proof>

lemma *wf-item-tree-build-tree':*

assumes $(bs, \omega, k, i) \in wf\text{-tree-input}$
assumes $wf\text{-bins } \mathcal{G} \ \omega \ bs$
assumes $build\text{-tree}' \ bs \ \omega \ k \ i = Some \ t$
shows $wf\text{-item-tree } \mathcal{G} \ (fst \ (bs!k!i)) \ t$
 <proof>

lemma *wf-yield-build-tree':*

assumes $(bs, \omega, k, i) \in wf\text{-tree-input}$
assumes $wf\text{-bins } \mathcal{G} \ \omega \ bs$
assumes $build\text{-tree}' \ bs \ \omega \ k \ i = Some \ t$
shows $wf\text{-yield } \omega \ (fst \ (bs!k!i)) \ t$
 <proof>

theorem *wf-rule-root-yield-build-tree:*

assumes $wf\text{-bins } \mathcal{G} \ \omega \ bs$ *sound-ptrs* $\omega \ bs$ *mono-red-ptr* $bs \ length \ bs = length \ \omega +$
 1
assumes $build\text{-tree } \mathcal{G} \ \omega \ bs = Some \ t$
shows $wf\text{-rule-tree } \mathcal{G} \ t \wedge root \ t = \mathfrak{S} \ \mathcal{G} \wedge yield \ t = \omega$
 <proof>

corollary *wf-rule-root- $\text{yield-build-tree-Earley}_L$:*

assumes ε -free \mathcal{G}

assumes *build-tree* $\mathcal{G} \ \omega$ ($\text{Earley}_L \ \mathcal{G} \ \omega$) = *Some* t

shows *wf-rule-tree* $\mathcal{G} \ t \wedge \text{root } t = \mathfrak{S} \ \mathcal{G} \wedge \text{yield } t = \omega$

<proof>

theorem *correctness-build-tree-Earley_L:*

assumes *is-word* $\mathcal{G} \ \omega$ ε -free \mathcal{G}

shows $(\exists t. \text{build-tree } \mathcal{G} \ \omega \ (\text{Earley}_L \ \mathcal{G} \ \omega) = \text{Some } t) \longleftrightarrow \mathcal{G} \vdash [\mathfrak{S} \ \mathcal{G}] \Rightarrow^* \omega$ (**is** ? L
 \longleftrightarrow ? R)

<proof>

end

theory *Examples*

imports

Earley-Parser

HOL-Library.Code-Target-Nat

begin

10 Examples

10.1 Common symbols

datatype *symbol* = $a \mid S \mid X \mid Y \mid Z$

10.2 $O(n^3)$ ambiguous grammars

10.2.1 $S \rightarrow SS \mid a$

definition *rules1* :: *symbol rule list* **where**

```
rules1 = [  
  (S, [S, S]),  
  (S, [a])  
]
```

definition *cfg1* :: *symbol cfg* **where**

cfg1 = *CFG rules1 S*

lemma ε -free1:

ε -free *cfg1*

<proof>

10.3 $O(n^2)$ unambiguous or bounded ambiguity

10.3.1 $S \rightarrow aS \mid a$

definition *rules2* :: *symbol rule list* **where**

```
rules2 = [  
  (S, [a, S]),  
  (S, [a])  
]
```

]

definition *cfg2* :: *symbol cfg* where

cfg2 = CFG *rules2* *S*

lemma *ε-free2*:

ε-free cfg2

<proof>

10.3.2 $S \rightarrow aSa \mid a$

definition *rules3* :: *symbol rule list* where

rules3 = [
 (*S*, [*a*, *S*, *a*]),
 (*S*, [*a*])
]

definition *cfg3* :: *symbol cfg* where

cfg3 = CFG *rules3* *S*

lemma *ε-free3*:

ε-free cfg3

<proof>

10.4 $O(n)$ bounded state, non-right recursive LR(k) grammars

10.4.1 $S \rightarrow Sa \mid a$

definition *rules4* :: *symbol rule list* where

rules4 = [
 (*S*, [*S*, *a*]),
 (*S*, [*a*])
]

definition *cfg4* :: *symbol cfg* where

cfg4 = CFG *rules4* *S*

lemma *ε-free4*:

ε-free cfg4

<proof>

10.5 $S \rightarrow SX, X \rightarrow Y \mid Z, Y \rightarrow a, Z \rightarrow a$

definition *rules5* :: *symbol rule list* where

rules5 = [
 (*S*, [*S*, *X*]),
 (*S*, [*a*]),
 (*X*, [*Y*]),
 (*X*, [*Z*]),

```

    (Y, [a]),
    (Z, [a])
  ]

```

definition *cfg5* :: *symbol cfg* **where**
cfg5 = *CFG rules5 S*

lemma *ε-free5*:
ε-free cfg5
 ⟨*proof*⟩

11 Input and Evaluation

definition *inp* :: *symbol list* **where**

```

  inp = [a,
    a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a,
    a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a,
    a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a,
    a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a
  ]

```

lemma *is-word-inp1*:
is-word cfg1 inp
 ⟨*proof*⟩

lemma *is-word-inp2*:
is-word cfg2 inp
 ⟨*proof*⟩

lemma *is-word-inp3*:
is-word cfg3 inp
 ⟨*proof*⟩

lemma *is-word-inp4*:
is-word cfg4 inp
 ⟨*proof*⟩

lemma *is-word-inp5*:
is-word cfg5 inp
 ⟨*proof*⟩

definition *size-bins* :: '*a bins* ⇒ *nat* **where**
size-bins *bs* = *fold (+) (map length bs) 0*

fun *size-pointer* :: '*a item* × *pointer* ⇒ *nat* **where**
size-pointer *(-, (PreRed - ps))* = *1 + length ps*
 | *size-pointer -* = *1*

definition *size-pointers* :: '*a bins* ⇒ *nat* **where**

```
size-pointers bs = fold (+) (map (λb. fold (+) (map (λe. size-pointer e) b) 0) bs)
0
```

```
export-code EarleyL build-tree rules1 cfg1 rules2 cfg2 rules3 cfg3 rules4 cfg4
rules5 cfg5 inp size-bins size-pointers in Scala
```

```
value size-bins (EarleyL cfg1 inp)
value size-pointers (EarleyL cfg1 inp)
```

```
value size-bins (EarleyL cfg2 inp)
value size-pointers (EarleyL cfg2 inp)
```

```
value size-bins (EarleyL cfg3 inp)
value size-pointers (EarleyL cfg3 inp)
```

```
value size-bins (EarleyL cfg4 inp)
value size-pointers (EarleyL cfg4 inp)
```

```
value size-bins (EarleyL cfg5 inp)
value size-pointers (EarleyL cfg5 inp)
```

```
end
```

References

- [1] J. Earley. An efficient context-free parsing algorithm. *Commun. ACM*, 13(2):94102, 1970.
- [2] C. B. Jones. Formal development of correct algorithms: An example based on earley’s recogniser. In *Proceedings of ACM Conference on Proving Assertions about Programs*, page 150169, New York, NY, USA, 1972. Association for Computing Machinery.
- [3] S. Obua. Local lexing. *Archive of Formal Proofs*, 2017. <https://isa-afp.org/entries/LocalLexing.html>, Formal proof development.
- [4] S. Obua, P. Scott, and J. Fleuriot. Local lexing, 2017.
- [5] E. Scott. Sppf-style parsing from earley recognisers. *Electronic Notes in Theoretical Computer Science*, 203(2):53–67, 2008. Proceedings of the Seventh Workshop on Language Descriptions, Tools, and Applications (LDTA 2007).