Earley

Martin Rau

March 17, 2025

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 auxilaries . . . 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 -> SS | a . . . 42
10.3 $O(n^2)$ unambiguous or bounded ambiguity . . . 42
10.3.1 S -> aS | a . . . 42
10.3.2 S -> aSa | a . . . 43
10.4 O(n) bounded state, non-right recursive LR(k) grammars . . . 43
10.4.1 S -> Sa | a . . . 43
10.5 S -> SX, X -> Y | Z, Y -> a, Z -> a . . . 43

11 Input and Evaluation **44**

theory *Limit*
imports
Main
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*

| *funpower f* (*Suc n*) *x* = *f* (*funpower f n x*)

definition *natUnion* :: (*nat* ⇒ *'a set*) ⇒ *'a set* **where**
natUnion f = ⋃ { *f n* | *n*. *True* }

definition *limit* :: (*'a set* ⇒ *'a set*) ⇒ *'a set* ⇒ *'a set* **where**
limit f x = *natUnion* (λ *n*. *funpower f n x*)

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

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

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

lemma[*simp*]: *limit id* = *id*
⟨*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 *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* ∈ *limit f X* = (∃ *n*. *x* ∈ *funpower f n X*)
⟨*proof*⟩

definition *pointwise* :: (*'a set* ⇒ *'b set*) ⇒ *bool* **where**
pointwise f = (∀ *X*. *f X* = ⋃ { *f* {*x*} | *x*. *x* ∈ *X*})

lemma *natUnion-elem*: *x* ∈ *f n* ⟹ *x* ∈ *natUnion f*
⟨*proof*⟩

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

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

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

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

lemma *setmonotone-implies-chain-funpower*:
assumes *setmonotone*: *setmonotone* f
shows *chain* $(\lambda\ n.$ *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* $(\lambda\ n.\ C\ (n + m))$
⟨*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 $P\ F\ I = I \cup \{\ F\ q\ x \mid q\ x.\ x \in I \wedge P\ q\ x\ \}$

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

end
theory *CFG*
imports *Main*
begin

2 Adjusted content from AFP/LocalLexing

type-synonym $'a\ rule = 'a \times 'a\ list$

type-synonym $'a\ rules = 'a\ rule\ list$

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

definition *nonterminals* :: $'a\ cfg \Rightarrow 'a\ set$ **where**
$nonterminals\ G = set\ (map\ fst\ (\mathfrak{R}\ G)) \cup \{\mathfrak{S}\ G\}$

definition *is-word* :: $'a\ cfg \Rightarrow 'a\ list \Rightarrow bool$ **where**
$is\text{-}word\ \mathcal{G}\ \omega = (nonterminals\ \mathcal{G} \cap set\ \omega = \{\})$

definition *derives1* :: $'a\ cfg \Rightarrow 'a\ list \Rightarrow 'a\ list \Rightarrow bool$ **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* :: $'a\ cfg \Rightarrow ('a\ list \times 'a\ list)\ set$ **where**
$derivations1\ \mathcal{G} \equiv \{\ (u,v) \mid u\ v.\ derives1\ \mathcal{G}\ u\ v\ \}$

definition *derivations* :: $'a\ cfg \Rightarrow ('a\ list \times 'a\ list)\ set$ **where**
$derivations\ \mathcal{G} \equiv (derivations1\ \mathcal{G})\widehat{\ }*$

definition *derives* :: $'a\ cfg \Rightarrow 'a\ list \Rightarrow 'a\ list \Rightarrow bool$ **where**
$derives\ \mathcal{G}\ u\ v \equiv ((u,\ v) \in derivations\ \mathcal{G})$

syntax
$derives1$ :: $'a\ cfg \Rightarrow 'a\ list \Rightarrow 'a\ list \Rightarrow bool$ (‹- ⊢ - ⇒ -› [1000,0,0] 1000)

syntax
$derives$ :: $'a\ cfg \Rightarrow 'a\ list \Rightarrow 'a\ list \Rightarrow bool$ (‹- ⊢ - ⇒* -› [1000,0,0] 1000)

notation (*latex* **output**)
$derives1$ (‹- ⊢ - ⇒ -› [1000,0,0] 1000)

notation (*latex* **output**)
  derives (‹- ⊢ - ⇒* -› [*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* ⟹ *derives* $\mathcal{G}$ *a b*
  ⟨*proof*⟩

lemma *derives-trans*:
  derives $\mathcal{G}$ *a b* ⟹ *derives* $\mathcal{G}$ *b c* ⟹ *derives* $\mathcal{G}$ *a c*
  ⟨*proof*⟩

lemma *derives1-eq-derivations1*:
  derives1 $\mathcal{G}$ *x y* = ((*x*, *y*) ∈ *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*: ⋀*y z*. *derives* $\mathcal{G}$ *a y* ⟹ *derives1* $\mathcal{G}$ *y z* ⟹ *P y* ⟹ *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* ≡ ∃ *x y A* α.
    u = *x* @ [*A*] @ *y* ∧
    v = *x* @ α @ *y* ∧
    (*A*, α) ∈ *set* ($\mathfrak{R}$ $\mathcal{G}$) ∧ *r* = (*A*, α) ∧ *i* = *length x*

lemma *Derives1-split*:
  Derives1 $\mathcal{G}$ *u i r v* ⟹ ∃ *x y*. *u* = *x* @ [*fst r*] @ *y* ∧ *v* = *x* @ (*snd r*) @ *y* ∧ *length x* = *i*
  ⟨*proof*⟩

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

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

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

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

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

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

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

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

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

4 Additional derivation lemmas

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

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

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

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

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

lemma *derives1-if-valid-rule*:
(*A*, α) $\in$ *set* ($\mathfrak{R}$ $\mathcal{G}$) $\Longrightarrow$ *derives1* $\mathcal{G}$ [*A*] α
⟨*proof*⟩

lemma *derives-if-valid-rule*:
(*A*, α) $\in$ *set* ($\mathfrak{R}$ $\mathcal{G}$) $\Longrightarrow$ *derives* $\mathcal{G}$ [*A*] α
⟨*proof*⟩

lemma *Derivation-from-empty*:
Derivation $\mathcal{G}$ [] *D a* $\Longrightarrow$ *a* = []
⟨*proof*⟩

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

lemma *Derivation-*$\mathfrak{S}$*1*:
assumes *Derivation* $\mathcal{G}$ [$\mathfrak{S}$ $\mathcal{G}$] *D* ω *is-word* $\mathcal{G}$ ω
shows $\exists$ α *E*. *Derivation* $\mathcal{G}$ α *E* ω $\wedge$ ($\mathfrak{S}$ $\mathcal{G}$,α) $\in$ *set* ($\mathfrak{R}$ $\mathcal{G}$)
⟨*proof*⟩

end
theory *Earley*
imports
Derivations
begin

5 Slices

fun *slice* :: *'a list* ⇒ *nat* ⇒ *nat* ⇒ *'a list* **where**
slice [] - - = []
| *slice* (*x*#*xs*) - *0* = []
| *slice* (*x*#*xs*) *0* (*Suc b*) = *x* # *slice xs 0 b*
| *slice* (*x*#*xs*) (*Suc a*) (*Suc b*) = *slice xs a b*

syntax
slice :: *'a list* ⇒ *nat* ⇒ *nat* ⇒ *'a list* (‹-_'/_› [*1000*,*0*,*0*] *1000*)

notation (*latex* **output**)
slice (‹-_'/_› [*1000*,*0*,*0*] *1000*)

lemma *slice-drop-take*:
$slice\ xs\ a\ b = drop\ a\ (take\ b\ xs)$
⟨*proof*⟩

lemma *slice-append-aux*:
$Suc\ b \leq c \Longrightarrow slice\ (x\#xs)\ (Suc\ b)\ c = slice\ xs\ b\ (c-1)$
⟨*proof*⟩

lemma *slice-concat*:
$a \leq b \Longrightarrow b \leq c \Longrightarrow slice\ xs\ a\ b\ @\ slice\ xs\ b\ c = slice\ xs\ a\ c$
⟨*proof*⟩

lemma *slice-concat-Ex*:
$a \leq c \Longrightarrow slice\ xs\ a\ c = ys\ @\ zs \Longrightarrow \exists\, b.\ ys = slice\ xs\ a\ b \wedge zs = slice\ xs\ b\ c \wedge a \leq b \wedge b \leq c$
⟨*proof*⟩

lemma *slice-nth*:
$a < length\ xs \Longrightarrow slice\ xs\ a\ (a+1) = [xs!a]$
⟨*proof*⟩

lemma *slice-append-nth*:
$a \leq b \Longrightarrow b < length\ xs \Longrightarrow slice\ xs\ a\ b\ @\ [xs!b] = slice\ xs\ a\ (b+1)$
⟨*proof*⟩

lemma *slice-empty*:
$b \leq a \Longrightarrow slice\ xs\ a\ b = []$
⟨*proof*⟩

lemma *slice-id*[*simp*]:
$slice\ xs\ 0\ (length\ xs) = xs$
⟨*proof*⟩

lemma *slice-singleton*:
$b \leq length\ xs \Longrightarrow [x] = slice\ xs\ a\ b \Longrightarrow b = a + 1$
⟨*proof*⟩

6 Earley recognizer

6.1 Earley items

definition *lhs-rule* :: $'a\ rule \Rightarrow 'a$ **where**
$lhs\text{-}rule \equiv fst$

definition *rhs-rule* :: $'a\ rule \Rightarrow 'a\ list$ **where**
$rhs\text{-}rule \equiv snd$

datatype $'a\ item =$
Item (*rule-item*: $'a\ rule$) (*dot-item* : *nat*) (*start-item* : *nat*) (*end-item* : *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 $\mathcal{G}$ ω $x \equiv$
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 $\mathcal{G}$ $\omega \equiv \exists x \in I.$ *is-finished* $\mathcal{G}$ ω x

inductive-set *Earley* :: $'a$ *cfg* $\Rightarrow$ $'a$ *list* $\Rightarrow$ $'a$ *item set*
for $\mathcal{G}$ :: $'a$ *cfg* **and** ω :: $'a$ *list* **where**
Init: $r \in$ *set* $(\mathfrak{R}\ \mathcal{G}) \Longrightarrow$ *fst* $r = \mathfrak{S}\ \mathcal{G} \Longrightarrow$
Item $r\ 0\ 0\ 0 \in$ *Earley* $\mathcal{G}\ \omega$
| *Scan*: $x =$ *Item* $r\ b\ i\ j \Longrightarrow x \in$ *Earley* $\mathcal{G}\ \omega \Longrightarrow$
$\omega!j = a \Longrightarrow j <$ *length* $\omega \Longrightarrow$ *next-symbol* $x =$ *Some* $a \Longrightarrow$
Item $r\ (b + 1)\ i\ (j + 1) \in$ *Earley* $\mathcal{G}\ \omega$
| *Predict*: $x =$ *Item* $r\ b\ i\ j \Longrightarrow x \in$ *Earley* $\mathcal{G}\ \omega \Longrightarrow$
$r' \in$ *set* $(\mathfrak{R}\ \mathcal{G}) \Longrightarrow$ *next-symbol* $x =$ *Some* (*lhs-rule* r') $\Longrightarrow$
Item $r'\ 0\ j\ j \in$ *Earley* $\mathcal{G}\ \omega$
| *Complete*: $x =$ *Item* $r_x\ b_x\ i\ j \Longrightarrow x \in$ *Earley* $\mathcal{G}\ \omega \Longrightarrow y =$ *Item* $r_y\ b_y\ j\ k \Longrightarrow$
$y \in$ *Earley* $\mathcal{G}\ \omega \Longrightarrow$
is-complete $y \Longrightarrow$ *next-symbol* $x =$ *Some* (*lhs-item* y) $\Longrightarrow$
Item $r_x\ (b_x + 1)\ i\ k \in$ *Earley* $\mathcal{G}\ \omega$

6.2 Well-formedness

definition *wf-item* :: $'a$ *cfg* $\Rightarrow$ $'a$ *list* => $'a$ *item* $\Rightarrow$ *bool* **where**
wf-item $\mathcal{G}$ ω x $\equiv$
rule-item $x \in$ *set* $(\mathfrak{R}\ \mathcal{G})$ $\wedge$
dot-item $x \leq$ *length* (*rhs-item* x) $\wedge$
start-item $x \leq$ *end-item* x $\wedge$
end-item $x \leq$ *length* ω

lemma *wf-Init*:
assumes $r \in$ *set* $(\mathfrak{R}\ \mathcal{G})$ *fst* $r = \mathfrak{S}\ \mathcal{G}$
shows *wf-item* $\mathcal{G}$ ω (*Item* r *0 0 0*)
⟨*proof*⟩

lemma *wf-Scan*:
assumes x = *Item* r b i j *wf-item* $\mathcal{G}$ ω x $\omega!j = a$ $j <$ *length* ω *next-symbol* x = *Some* a
shows *wf-item* $\mathcal{G}$ ω (*Item* r (b + *1*) i (j+*1*))
⟨*proof*⟩

lemma *wf-Predict*:
assumes x = *Item* r b i j *wf-item* $\mathcal{G}$ ω x $r' \in$ *set* $(\mathfrak{R}\ \mathcal{G})$ *next-symbol* x = *Some* (*lhs-rule* r')
shows *wf-item* $\mathcal{G}$ ω (*Item* r' *0* j j)
⟨*proof*⟩

lemma *wf-Complete*:
assumes x = *Item* r_x b_x i j *wf-item* $\mathcal{G}$ ω x y = *Item* r_y b_y j k *wf-item* $\mathcal{G}$ ω y
assumes *is-complete* y *next-symbol* x = *Some* (*lhs-item* y)
shows *wf-item* $\mathcal{G}$ ω (*Item* r_x (b_x + *1*) i k)
⟨*proof*⟩

lemma *wf-Earley*:
assumes $x \in$ *Earley* $\mathcal{G}$ ω
shows *wf-item* $\mathcal{G}$ ω x
⟨*proof*⟩

6.3 Soundness

definition *sound-item* :: $'a$ *cfg* $\Rightarrow$ $'a$ *list* $\Rightarrow$ $'a$ *item* $\Rightarrow$ *bool* **where**
sound-item $\mathcal{G}$ ω $x \equiv \mathcal{G} \vdash$ [*lhs-item* x] $\Rightarrow^*$ (*slice* ω (*start-item* x) (*end-item* x) @ β-*item* x)

lemma *sound-Init*:
assumes $r \in$ *set* $(\mathfrak{R}\ \mathcal{G})$ *fst* $r = \mathfrak{S}\ \mathcal{G}$
shows *sound-item* $\mathcal{G}$ ω (*Item* r *0 0 0*)
⟨*proof*⟩

lemma *sound-Scan*:
assumes x = *Item* r b i j *wf-item* $\mathcal{G}$ ω x *sound-item* $\mathcal{G}$ ω x

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

lemma *sound-Predict*:
assumes $x =$ *Item* r b i j *wf-item* $\mathcal{G}$ ω x *sound-item* $\mathcal{G}$ ω x
assumes $r' \in$ *set* ($\mathfrak{R}$ $\mathcal{G}$) *next-symbol* $x =$ *Some* (*lhs-rule* r')
shows *sound-item* $\mathcal{G}$ ω (*Item* r' 0 j j)
⟨*proof*⟩

lemma *sound-Complete*:
assumes $x =$ *Item* r_x b_x i j *wf-item* $\mathcal{G}$ ω x *sound-item* $\mathcal{G}$ ω x
assumes $y =$ *Item* r_y b_y j k *wf-item* $\mathcal{G}$ ω y *sound-item* $\mathcal{G}$ ω y
assumes *is-complete* y *next-symbol* $x =$ *Some* (*lhs-item* y)
shows *sound-item* $\mathcal{G}$ ω (*Item* r_x (b_x + 1) i k)
⟨*proof*⟩

lemma *sound-Earley*:
assumes $x \in$ *Earley* $\mathcal{G}$ ω *wf-item* $\mathcal{G}$ ω x
shows *sound-item* $\mathcal{G}$ ω x
⟨*proof*⟩

theorem *soundness-Earley*:
assumes *recognizing* (*Earley* $\mathcal{G}$ ω) $\mathcal{G}$ ω
shows $\mathcal{G} \vdash [\mathfrak{S}\ \mathcal{G}] \Rightarrow^* \omega$
⟨*proof*⟩

6.4 Completeness

definition *partially-completed* :: *nat* ⇒ *'a cfg* ⇒ *'a list* ⇒ *'a item set* ⇒ (*'a derivation* ⇒ *bool*) ⇒ *bool* **where**
partially-completed k $\mathcal{G}$ ω I P ≡ ∀ r b i' i j x a D.
$i \leq j \wedge j \leq k \wedge k \leq$ *length* $\omega \wedge$
$x =$ *Item* r b i' $i \wedge x \in I \wedge$ *next-symbol* $x =$ *Some* $a \wedge$
Derivation $\mathcal{G}$ [a] D (*slice* ω i j) ∧ P D ⟶
Item r (b+1) i' $j \in I$

lemma *partially-completed-upto*:
assumes $j \leq k$ $k \leq$ *length* ω
assumes $x =$ *Item* (N,α) d i j $x \in I$ ∀ $x \in I$. *wf-item* $\mathcal{G}$ ω x
assumes *Derivation* $\mathcal{G}$ (β-*item* x) D (*slice* ω j k)
assumes *partially-completed* k $\mathcal{G}$ ω I ($\lambda D'$. *length* $D' \leq$ *length* D)
shows *Item* (N,α) (*length* α) i $k \in I$
⟨*proof*⟩

lemma *partially-completed-Earley*:
partially-completed k $\mathcal{G}$ ω (*Earley* $\mathcal{G}$ ω) (λ-. *True*)
⟨*proof*⟩

theorem *completeness-Earley*:
  assumes $\mathcal{G} \vdash [\mathfrak{S}\ \mathcal{G}] \Rightarrow^* \omega$ *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* $\mathcal{G}\ \omega$) $\mathcal{G}\ \omega \longleftrightarrow \mathcal{G} \vdash [\mathfrak{S}\ \mathcal{G}] \Rightarrow^* \omega$
  ⟨*proof*⟩

6.6 Finiteness

lemma *finiteness-empty*:
  set $(\mathfrak{R}\ \mathcal{G}) = \{\} \Longrightarrow$ *finite* $\{\ x \mid x.$ *wf-item* $\mathcal{G}\ \omega\ x\ \}$
  ⟨*proof*⟩

fun *item-intro* :: *'a rule* × *nat* × *nat* × *nat* ⇒ *'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* $\{\ x.$ *wf-item* $\mathcal{G}\ \omega\ x\ \}$
⟨*proof*⟩

lemma *finiteness-UNIV-wf-item*:
  finite $\{\ x.$ *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* ⇒ *nat* ⇒ *'a item* **where**
  init-item r k ≡ *Item r 0 k k*

definition *inc-item* :: *'a item* ⇒ *nat* ⇒ *'a item* **where**

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

definition *prev-symbol* :: *'a item* $\Rightarrow$ *'a option* **where**
prev-symbol x $\equiv$ *if dot-item x* = *0 then None else Some* (*rhs-item x* ! (*dot-item x* − *1*))

definition *base* :: *'a list* $\Rightarrow$ *'a item set* $\Rightarrow$ *nat* $\Rightarrow$ *'a item set* **where**
base ω *I k* $\equiv$ { *x* . *x* $\in$ *I* $\wedge$ *end-item x* = *k* $\wedge$ *k* > *0* $\wedge$ *prev-symbol x* = *Some* (ω!(*k*−*1*)) }

definition $Init_F$ :: *'a cfg* $\Rightarrow$ *'a item set* **where**
$Init_F$ $\mathcal{G}$ $\equiv$ { *init-item r 0* | *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* ω *I* $\equiv$ { *inc-item x* (*k*+*1*) | *x a*.
x $\in$ *bin I k* $\wedge$
ω!*k* = *a* $\wedge$
k < *length* ω $\wedge$
next-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-item r k* | *r x*.
r $\in$ *set* ($\mathfrak{R}$ $\mathcal{G}$) $\wedge$
x $\in$ *bin I k* $\wedge$
next-symbol x = *Some* (*lhs-rule r*) }

definition $Complete_F$ :: *nat* $\Rightarrow$ *'a item set* $\Rightarrow$ *'a item set* **where**
$Complete_F$ *k I* $\equiv$ { *inc-item x k* | *x y*.
x $\in$ *bin I* (*start-item y*) $\wedge$
y $\in$ *bin I k* $\wedge$
is-complete y $\wedge$
next-symbol x = *Some* (*lhs-item y*) }

definition $Earley_F$*-bin-step* :: *nat* $\Rightarrow$ *'a cfg* $\Rightarrow$ *'a list* $\Rightarrow$ *'a item set* $\Rightarrow$ *'a item set* **where**
$Earley_F$*-bin-step k* $\mathcal{G}$ ω *I* = *I* $\cup$ $Scan_F$ *k* ω *I* $\cup$ $Complete_F$ *k I* $\cup$ $Predict_F$ *k* $\mathcal{G}$ *I*

definition $Earley_F$*-bin* :: *nat* $\Rightarrow$ *'a cfg* $\Rightarrow$ *'a list* $\Rightarrow$ *'a item set* $\Rightarrow$ *'a item set* **where**
$Earley_F$*-bin k* $\mathcal{G}$ ω *I* $\equiv$ *limit* ($Earley_F$*-bin-step k* $\mathcal{G}$ ω) *I*

fun $Earley_F$*-bins* :: *nat* $\Rightarrow$ *'a cfg* $\Rightarrow$ *'a list* $\Rightarrow$ *'a item set* **where**
$Earley_F$*-bins 0* $\mathcal{G}$ ω = $Earley_F$*-bin 0* $\mathcal{G}$ ω ($Init_F$ $\mathcal{G}$)
| $Earley_F$*-bins* (*Suc n*) $\mathcal{G}$ ω = $Earley_F$*-bin* (*Suc n*) $\mathcal{G}$ ω ($Earley_F$*-bins n* $\mathcal{G}$ ω)

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

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

7.2 Monotonicity and Absorption

lemma $Earley_F$*-bin-step-empty*:
$Earley_F\text{-}bin\text{-}step\ k\ \mathcal{G}\ \omega\ \{\} = \{\}$
⟨*proof*⟩

lemma $Earley_F$*-bin-step-setmonotone*:
$setmonotone\ (Earley_F\text{-}bin\text{-}step\ k\ \mathcal{G}\ \omega)$
⟨*proof*⟩

lemma $Earley_F$*-bin-step-continuous*:
$continuous\ (Earley_F\text{-}bin\text{-}step\ k\ \mathcal{G}\ \omega)$
⟨*proof*⟩

lemma $Earley_F$*-bin-step-regular*:
$regular\ (Earley_F\text{-}bin\text{-}step\ k\ \mathcal{G}\ \omega)$
⟨*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*:
$I \subseteq J \Longrightarrow Scan_F\ k\ \omega\ I \subseteq Scan_F\ k\ \omega\ J$
⟨*proof*⟩

lemma $Predict_F$*-sub-mono*:
$I \subseteq J \Longrightarrow Predict_F\ k\ \mathcal{G}\ I \subseteq Predict_F\ k\ \mathcal{G}\ J$
⟨*proof*⟩

lemma $Complete_F$*-sub-mono*:

$I \subseteq J \Longrightarrow Complete_F\ k\ I \subseteq Complete_F\ k\ J$
$\langle proof \rangle$

lemma $Earley_F$-*bin-step-sub-mono*:
$I \subseteq J \Longrightarrow Earley_F$-*bin-step* $k\ \mathcal{G}\ \omega\ I \subseteq Earley_F$-*bin-step* $k\ \mathcal{G}\ \omega\ J$
$\langle proof \rangle$

lemma *funpower-sub-mono*:
$I \subseteq J \Longrightarrow funpower\ (Earley_F\text{-}bin\text{-}step\ k\ \mathcal{G}\ \omega)\ n\ I \subseteq funpower\ (Earley_F\text{-}bin\text{-}step\ k\ \mathcal{G}\ \omega)\ n\ J$
$\langle proof \rangle$

lemma $Earley_F$-*bin-sub-mono*:
$I \subseteq J \Longrightarrow Earley_F$-*bin* $k\ \mathcal{G}\ \omega\ I \subseteq Earley_F$-*bin* $k\ \mathcal{G}\ \omega\ J$
$\langle proof \rangle$

lemma $Scan_F$-$Earley_F$-*bin-step-mono*:
$Scan_F\ k\ \omega\ I \subseteq Earley_F$-*bin-step* $k\ \mathcal{G}\ \omega\ I$
$\langle proof \rangle$

lemma $Predict_F$-$Earley_F$-*bin-step-mono*:
$Predict_F\ k\ \mathcal{G}\ I \subseteq Earley_F$-*bin-step* $k\ \mathcal{G}\ \omega\ I$
$\langle proof \rangle$

lemma $Complete_F$-$Earley_F$-*bin-step-mono*:
$Complete_F\ k\ I \subseteq Earley_F$-*bin-step* $k\ \mathcal{G}\ \omega\ I$
$\langle proof \rangle$

lemma $Earley_F$-*bin-step*-$Earley_F$-*bin-mono*:
$Earley_F$-*bin-step* $k\ \mathcal{G}\ \omega\ I \subseteq Earley_F$-*bin* $k\ \mathcal{G}\ \omega\ I$
$\langle proof \rangle$

lemma $Scan_F$-$Earley_F$-*bin-mono*:
$Scan_F\ k\ \omega\ \ I \subseteq Earley_F$-*bin* $k\ \mathcal{G}\ \omega\ I$
$\langle proof \rangle$

lemma $Predict_F$-$Earley_F$-*bin-mono*:
$Predict_F\ k\ \mathcal{G}\ I \subseteq Earley_F$-*bin* $k\ \mathcal{G}\ \omega\ I$
$\langle proof \rangle$

lemma $Complete_F$-$Earley_F$-*bin-mono*:
$Complete_F\ k\ I \subseteq Earley_F$-*bin* $k\ \mathcal{G}\ \omega\ I$
$\langle proof \rangle$

lemma $Earley_F$-*bin-mono*:
$I \subseteq Earley_F$-*bin* $k\ \mathcal{G}\ \omega\ I$
$\langle proof \rangle$

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 $Earley_F\text{-}bin\text{-}step\ k\ \mathcal{G}\ \omega\ I \subseteq Earley\ \mathcal{G}\ \omega$
⟨*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 $bin\ (Earley\ \mathcal{G}\ \omega)\ k \subseteq bin\ (Earley_F\text{-}bin\ k\ \mathcal{G}\ \omega\ I)\ k$
⟨*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 $is\text{-}word\ \mathcal{G}\ \omega$
shows $base\ \omega\ (Earley\ \mathcal{G}\ \omega)\ (k{+}1) \subseteq bin\ (Earley_F\text{-}bin\ k\ \mathcal{G}\ \omega\ I)\ (k{+}1)$
⟨*proof*⟩

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

lemma *Earley-sub-Earley$_F$*:
assumes $is\text{-}word\ \mathcal{G}\ \omega$
shows $Earley\ \mathcal{G}\ \omega \subseteq Earley_F\ \mathcal{G}\ \omega$
⟨*proof*⟩

theorem *completeness-Earley$_F$*:
assumes $\mathcal{G} \vdash [\mathfrak{S}\ \mathcal{G}] \Rightarrow^* \omega\ is\text{-}word\ \mathcal{G}\ \omega$
shows $recognizing\ (Earley_F\ \mathcal{G}\ \omega)\ \mathcal{G}\ \omega$
⟨*proof*⟩

7.5 Correctness

theorem *Earley-eq-Earley$_F$*:
assumes $is\text{-}word\ \mathcal{G}\ \omega$
shows $Earley\ \mathcal{G}\ \omega = Earley_F\ \mathcal{G}\ \omega$
⟨*proof*⟩

theorem *correctness-Earley$_F$*:
assumes $is\text{-}word\ \mathcal{G}\ \omega$
shows $recognizing\ (Earley_F\ \mathcal{G}\ \omega)\ \mathcal{G}\ \omega \longleftrightarrow \mathcal{G} \vdash [\mathfrak{S}\ \mathcal{G}] \Rightarrow^* \omega$
⟨*proof*⟩

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

8 Earley recognizer

8.1 List auxilaries

fun *filter-with-index′* :: *nat* ⇒ (*′a* ⇒ *bool*) ⇒ *′a list* ⇒ (*′a* × *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* ⇒ *bool*) ⇒ *′a list* ⇒ (*′a* × *nat*) *list* **where**
  filter-with-index P xs = *filter-with-index′ 0 P xs*

lemma *filter-with-index′-P*:
  (*x*, *n*) ∈ *set* (*filter-with-index′ i P xs*) ⟹ *P x*
  ⟨*proof*⟩

lemma *filter-with-index-P*:
  (*x*, *n*) ∈ *set* (*filter-with-index P xs*) ⟹ *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*) ∈ *set* (*filter-with-index′ i P xs*) ⟹ *n* ≥ *i*
  ⟨*proof*⟩

lemma *index-filter-with-index′-lt-length*:
  (*x*, *n*) ∈ *set* (*filter-with-index′ i P xs*) ⟹ *n*−*i* < *length xs*
  ⟨*proof*⟩

lemma *index-filter-with-index-lt-length*:
  (*x*, *n*) ∈ *set* (*filter-with-index P xs*) ⟹ *n* < *length xs*
  ⟨*proof*⟩

lemma *filter-with-index′-nth*:
  (*x*, *n*) ∈ *set* (*filter-with-index′ i P xs*) ⟹ *xs* ! (*n*−*i*) = *x*
⟨*proof*⟩

lemma *filter-with-index-nth*:
  (*x*, *n*) ∈ *set* (*filter-with-index P xs*) ⟹ *xs* ! *n* = *x*
  ⟨*proof*⟩

lemma *filter-with-index-nonempty*:

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

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

lemma *filter-with-index-Ex-first*:
$(\exists\, x\ i\ xs'.\ filter\text{-}with\text{-}index\ P\ xs = (x,\ i)\#xs') \longleftrightarrow (\exists\, x \in set\ xs.\ P\ x)$
⟨*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**
$bins\ bs \equiv \bigcup \{\ set\ (items\ (bs!k))\ |\ k.\ k < length\ bs\ \}$

definition *bin-upto* :: *′a bin* ⇒ *nat* ⇒ *′a item set* **where**
$bin\text{-}upto\ b\ i \equiv \{\ items\ b\ !\ j\ |\ j.\ j < i \wedge j < length\ (items\ b)\ \}$

definition *bins-upto* :: *′a bins* ⇒ *nat* ⇒ *nat* ⇒ *′a item set* **where**
$bins\text{-}upto\ bs\ k\ i \equiv \bigcup \{\ set\ (items\ (bs\ !\ l))\ |\ l.\ l < k\ \} \cup bin\text{-}upto\ (bs\ !\ k)\ i$

definition *wf-bin-items* :: *′a cfg* ⇒ *′a list* ⇒ *nat* ⇒ *′a item list* ⇒ *bool* **where**
$wf\text{-}bin\text{-}items\ \mathcal{G}\ \omega\ k\ xs \equiv \forall\, x \in set\ xs.\ wf\text{-}item\ \mathcal{G}\ \omega\ x \wedge end\text{-}item\ x = k$

definition *wf-bin* :: *′a cfg* ⇒ *′a list* ⇒ *nat* ⇒ *′a bin* ⇒ *bool* **where**
$wf\text{-}bin\ \mathcal{G}\ \omega\ k\ b \equiv distinct\ (items\ b) \wedge wf\text{-}bin\text{-}items\ \mathcal{G}\ \omega\ k\ (items\ b)$

definition *wf-bins* :: *′a cfg* ⇒ *′a list* ⇒ *′a bins* ⇒ *bool* **where**
$wf\text{-}bins\ \mathcal{G}\ \omega\ bs \equiv \forall\, k < length\ bs.\ wf\text{-}bin\ \mathcal{G}\ \omega\ k\ (bs!k)$

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

ε-*free* $\mathcal{G}$ = ($\forall$ *r* $\in$ *set* ($\mathfrak{R}$ $\mathcal{G}$). *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**
$Init_L$ $\mathcal{G}$ ω $\equiv$
let rs = *filter* (λ*r*. *lhs-rule r* = $\mathfrak{S}$ $\mathcal{G}$) (*remdups* ($\mathfrak{R}$ $\mathcal{G}$)) *in*
let b0 = *map* (λ*r*. (*init-item r 0*, *Null*)) *rs in*
let bs = *replicate* (*length* ω + *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 ω!*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* (λ*r*. *lhs-rule r* = *X*) ($\mathfrak{R}$ $\mathcal{G}$) *in*
map (λ*r*. (*init-item r k*, *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* (λ*x*. *next-symbol x* = *Some* (*lhs-item y*)) (*items orig*) *in*
map (λ(*x*, *pre*). (*inc-item x k*, *PreRed* (*start-item y*, *pre*, *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-bins bs k es $\equiv$ *bs*[*k* := *upds-bin es* (*bs*!*k*)]

partial-function (*tailrec*) *Earley*$_L$*-bin′* :: *nat* $\Rightarrow$ *′a cfg* $\Rightarrow$ *′a list* $\Rightarrow$ *′a bins* $\Rightarrow$ *nat* $\Rightarrow$ *′a bins* **where**

```
Earley_L-bin' k 𝒢 ω bs i = (
  if i ≥ length (items (bs ! k)) then bs
  else
    let x = items (bs!k) ! i in
    let bs' =
      case next-symbol x of
        Some a ⇒ (
          if a ∉ nonterminals 𝒢 then
            if k < length ω then upd-bins bs (k+1) (Scan_L k ω a x i)
            else bs
          else upd-bins bs k (Predict_L k 𝒢 a))
      | None ⇒ upd-bins bs k (Complete_L k x bs i)
    in Earley_L-bin' k 𝒢 ω bs' (i+1))
```

declare *Earley*$_L$*-bin′.simps*[*code*]

definition *Earley*$_L$*-bin* :: *nat* $\Rightarrow$ *′a cfg* $\Rightarrow$ *′a list* $\Rightarrow$ *′a bins* $\Rightarrow$ *′a bins* **where**
Earley$_L$*-bin k* $\mathcal{G}$ ω *bs* = *Earley*$_L$*-bin′ k* $\mathcal{G}$ ω *bs 0*

fun *Earley*$_L$*-bins* :: *nat* $\Rightarrow$ *′a cfg* $\Rightarrow$ *′a list* $\Rightarrow$ *′a bins* **where**
Earley$_L$*-bins 0* $\mathcal{G}$ ω = *Earley*$_L$*-bin 0* $\mathcal{G}$ ω (*Init*$_L$ $\mathcal{G}$ ω)
| *Earley*$_L$*-bins* (*Suc n*) $\mathcal{G}$ ω = *Earley*$_L$*-bin* (*Suc n*) $\mathcal{G}$ ω (*Earley*$_L$*-bins n* $\mathcal{G}$ ω)

definition *Earley*$_L$ :: *′a cfg* $\Rightarrow$ *′a list* $\Rightarrow$ *′a bins* **where**
Earley$_L$ $\mathcal{G}$ ω $\equiv$ *Earley*$_L$*-bins* (*length* ω) $\mathcal{G}$ ω

definition *recognizer* :: *′a cfg* $\Rightarrow$ *′a list* $\Rightarrow$ *bool* **where**
recognizer $\mathcal{G}$ ω = ($\exists$ *x* $\in$ *set* (*items* (*Earley*$_L$ $\mathcal{G}$ ω ! *length* ω)). *is-finished* $\mathcal{G}$ ω *x*)

8.3 Epsilon productions

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

lemma *ε-free-impl-nonempty-derives*:
ε-free $\mathcal{G}$ $\Longrightarrow$ *nonempty-derives* $\mathcal{G}$
⟨*proof*⟩

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

lemma *nonempty-derives-iff-ε-free*:

shows *nonempty-derives* $\mathcal{G} \longleftrightarrow$ *ε-free* $\mathcal{G}$
⟨*proof*⟩

8.4 Bin lemmas

lemma *length-upd-bins*[*simp*]:
length (*upd-bins bs k es*) = *length bs*
⟨*proof*⟩

lemma *length-upd-bin*:
length (*upd-bin e b*) ≥ *length b*
⟨*proof*⟩

lemma *length-upds-bin*:
length (*upds-bin es b*) ≥ *length b*
⟨*proof*⟩

lemma *length-nth-upd-bin-bins*:
length (*upd-bins bs k es* ! *n*) ≥ *length* (*bs* ! *n*)
⟨*proof*⟩

lemma *nth-idem-upd-bins*:
k ≠ *n* ⟹ *upd-bins bs k es* ! *n* = *bs* ! *n*
⟨*proof*⟩

lemma *items-nth-idem-upd-bin*:
n < *length b* ⟹ *items* (*upd-bin e b*) ! *n* = *items b* ! *n*
⟨*proof*⟩

lemma *items-nth-idem-upds-bin*:
n < *length b* ⟹ *items* (*upds-bin es b*) ! *n* = *items b* ! *n*
⟨*proof*⟩

lemma *items-nth-idem-upd-bins*:
n < *length* (*bs* ! *k*) ⟹ *items* (*upd-bins bs k es* ! *k*) ! *n* = *items* (*bs* ! *k*) ! *n*
⟨*proof*⟩

lemma *bin-upto-eq-set-items*:
i ≥ *length b* ⟹ *bin-upto b i* = *set* (*items b*)
⟨*proof*⟩

lemma *bins-upto-empty*:
bins-upto bs 0 0 = {}
⟨*proof*⟩

lemma *set-items-upd-bin*:
set (*items* (*upd-bin e b*)) = *set* (*items b*) ∪ {*fst e*}
⟨*proof*⟩

lemma *set-items-upds-bin*:
$set\ (items\ (upds\text{-}bin\ es\ b)) = set\ (items\ b) \cup set\ (items\ es)$
⟨*proof*⟩

lemma *bins-upd-bins*:
assumes $k < length\ bs$
shows $bins\ (upd\text{-}bins\ bs\ k\ es) = bins\ bs \cup set\ (items\ es)$
⟨*proof*⟩

lemma *kth-bin-sub-bins*:
$k < length\ bs \Longrightarrow set\ (items\ (bs\ !\ k)) \subseteq bins\ bs$
⟨*proof*⟩

lemma *bin-upto-Cons-0*:
$bin\text{-}upto\ (e\#es)\ 0 = \{\}$
⟨*proof*⟩

lemma *bin-upto-Cons*:
assumes $0 < n$
shows $bin\text{-}upto\ (e\#es)\ n = \{\ fst\ e\ \} \cup bin\text{-}upto\ es\ (n-1)$
⟨*proof*⟩

lemma *bin-upto-nth-idem-upd-bin*:
$n < length\ b \Longrightarrow bin\text{-}upto\ (upd\text{-}bin\ e\ b)\ n = bin\text{-}upto\ b\ n$
⟨*proof*⟩

lemma *bin-upto-nth-idem-upds-bin*:
$n < length\ b \Longrightarrow bin\text{-}upto\ (upds\text{-}bin\ es\ b)\ n = bin\text{-}upto\ b\ n$
⟨*proof*⟩

lemma *bins-upto-kth-nth-idem*:
assumes $l < length\ bs\ k \leq l\ n < length\ (bs\ !\ k)$
shows $bins\text{-}upto\ (upd\text{-}bins\ bs\ l\ es)\ k\ n = bins\text{-}upto\ bs\ k\ n$
⟨*proof*⟩

lemma *bins-upto-sub-bins*:
$k < length\ bs \Longrightarrow bins\text{-}upto\ bs\ k\ n \subseteq bins\ bs$
⟨*proof*⟩

lemma *bins-upto-Suc-Un*:
$n < length\ (bs\ !\ k) \Longrightarrow bins\text{-}upto\ bs\ k\ (n+1) = bins\text{-}upto\ bs\ k\ n \cup \{\ items\ (bs\ !\ k)\ !\ n\ \}$
⟨*proof*⟩

lemma *bins-bin-exists*:
$x \in bins\ bs \Longrightarrow \exists\, k < length\ bs.\ x \in set\ (items\ (bs\ !\ k))$
⟨*proof*⟩

lemma *distinct-upd-bin*:

distinct (*items b*) $\Longrightarrow$ *distinct* (*items* (*upd-bin e b*))
⟨*proof*⟩

lemma *distinct-upds-bin*:
distinct (*items b*) $\Longrightarrow$ *distinct* (*items* (*upds-bin es b*))
⟨*proof*⟩

lemma *wf-bins-kth-bin*:
wf-bins $\mathcal{G}$ ω *bs* $\Longrightarrow$ $k <$ *length bs* $\Longrightarrow$ $x \in$ *set* (*items* (*bs* ! *k*)) $\Longrightarrow$ *wf-item* $\mathcal{G}$ ω *x* $\wedge$ *end-item x* = *k*
⟨*proof*⟩

lemma *wf-bin-upd-bin*:
assumes *wf-bin* $\mathcal{G}$ ω *k b wf-item* $\mathcal{G}$ ω (*fst e*) $\wedge$ *end-item* (*fst e*) = *k*
shows *wf-bin* $\mathcal{G}$ ω *k* (*upd-bin e b*)
⟨*proof*⟩

lemma *wf-upd-bins-bin*:
assumes *wf-bin* $\mathcal{G}$ ω *k b*
assumes $\forall x \in$ *set* (*items es*). *wf-item* $\mathcal{G}$ ω *x* $\wedge$ *end-item x* = *k*
shows *wf-bin* $\mathcal{G}$ ω *k* (*upds-bin es b*)
⟨*proof*⟩

lemma *wf-bins-upd-bins*:
assumes *wf-bins* $\mathcal{G}$ ω *bs*
assumes $\forall x \in$ *set* (*items es*). *wf-item* $\mathcal{G}$ ω *x* $\wedge$ *end-item x* = *k*
shows *wf-bins* $\mathcal{G}$ ω (*upd-bins bs k es*)
⟨*proof*⟩

lemma *wf-bins-impl-wf-items*:
wf-bins $\mathcal{G}$ ω *bs* $\Longrightarrow$ $\forall x \in$ (*bins bs*). *wf-item* $\mathcal{G}$ ω *x*
⟨*proof*⟩

lemma *upds-bin-eq-items*:
set (*items es*) $\subseteq$ *set* (*items b*) $\Longrightarrow$ *set* (*items* (*upds-bin es b*)) = *set* (*items b*)
⟨*proof*⟩

lemma *bin-eq-items-upd-bin*:
fst e $\in$ *set* (*items b*) $\Longrightarrow$ *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′ $\Longrightarrow$ *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* ∈ *set* (*items* (*bs* ! *k*))
assumes *k* < *length* ω *next-symbol x* ≠ *None y* = *inc-item x* (*k*+*1*)
shows *wf-item* $\mathcal{G}$ ω *y* ∧ *end-item y* = *k*+*1*
⟨*proof*⟩

lemma *wf-bins-Scan*$_L$:
assumes *wf-bins* $\mathcal{G}$ ω *bs k* < *length bs x* ∈ *set* (*items* (*bs* ! *k*)) *k* < *length* ω *next-symbol x* ≠ *None*
shows ∀ *y* ∈ *set* (*items* (*Scan*$_L$ *k* ω *a x pre*)). *wf-item* $\mathcal{G}$ ω *y* ∧ *end-item y* = (*k*+*1*)
⟨*proof*⟩

lemma *wf-bins-Predict*$_L$:
assumes *wf-bins* $\mathcal{G}$ ω *bs k* < *length bs k* ≤ *length* ω
shows ∀ *y* ∈ *set* (*items* (*Predict*$_L$ *k* $\mathcal{G}$ *X*)). *wf-item* $\mathcal{G}$ ω *y* ∧ *end-item y* = *k*

⟨*proof*⟩

lemma *wf-item-inc-item*:
assumes *wf-item* $\mathcal{G}$ ω x *next-symbol* x = *Some a start-item* $x \leq k$ $k \leq$ *length* ω
shows *wf-item* $\mathcal{G}$ ω (*inc-item* x k) $\wedge$ *end-item* (*inc-item* x k) = k
⟨*proof*⟩

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

lemma *Ex-wf-bins*:
$\exists$ n *bs* ω $\mathcal{G}$. $n \leq$ *length* ω $\wedge$ *length bs* = *Suc* (*length* ω) $\wedge$ *wf-bins* $\mathcal{G}$ ω *bs*
⟨*proof*⟩

definition *wf-earley-input* :: (*nat* × *'a cfg* × *'a list* × *'a bins*) *set* **where**
wf-earley-input = {
(k, $\mathcal{G}$, ω, *bs*) | k $\mathcal{G}$ ω *bs*.
$k \leq$ *length* ω $\wedge$
length bs = *length* ω + *1* $\wedge$
wf-bins $\mathcal{G}$ ω *bs*
}

typedef *'a wf-bins* = *wf-earley-input*::(*nat* × *'a cfg* × *'a list* × *'a bins*) *set*
morphisms *from-wf-bins to-wf-bins*
⟨*proof*⟩

lemma *wf-earley-input-elim*:
assumes (k, $\mathcal{G}$, ω, *bs*) $\in$ *wf-earley-input*
shows $k \leq$ *length* ω $\wedge$ $k <$ *length bs* $\wedge$ *length bs* = *length* ω + *1* $\wedge$ *wf-bins* $\mathcal{G}$ ω *bs*
⟨*proof*⟩

lemma *wf-earley-input-intro*:
assumes $k \leq$ *length* ω *length bs* = *length* ω + *1* *wf-bins* $\mathcal{G}$ ω *bs*
shows (k, $\mathcal{G}$, ω, *bs*) $\in$ *wf-earley-input*
⟨*proof*⟩

lemma *wf-earley-input-Complete*$_L$:
assumes (k, $\mathcal{G}$, ω, *bs*) $\in$ *wf-earley-input* $\neg$ *length* (*items* (*bs* ! k)) $\leq i$
assumes x = *items* (*bs* ! k) ! i *next-symbol* x = *None*
shows (k, $\mathcal{G}$, ω, *upd-bins bs* k (*Complete*$_L$ k x *bs red*)) $\in$ *wf-earley-input*
⟨*proof*⟩

lemma *wf-earley-input-Scan*$_L$:
assumes (k, $\mathcal{G}$, ω, *bs*) $\in$ *wf-earley-input* $\neg$ *length* (*items* (*bs* ! k)) $\leq i$
assumes x = *items* (*bs* ! k) ! i *next-symbol* x = *Some a*

assumes $k < length\ \omega$
shows $(k, \mathcal{G}, \omega, upd\text{-}bins\ bs\ (k+1)\ (Scan_L\ k\ \omega\ a\ x\ pre)) \in wf\text{-}earley\text{-}input$
⟨*proof*⟩

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

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

lemma *Earley*$_L$*-bin′-simps*[*simp*]:
$i \geq length\ (items\ (bs\ !\ k)) \Longrightarrow Earley_L\text{-}bin'\ k\ \mathcal{G}\ \omega\ bs\ i = bs$
$\neg\ i \geq length\ (items\ (bs\ !\ k)) \Longrightarrow x = items\ (bs!k)\ !\ i \Longrightarrow next\text{-}symbol\ x = None \Longrightarrow$
$Earley_L\text{-}bin'\ k\ \mathcal{G}\ \omega\ bs\ i = Earley_L\text{-}bin'\ k\ \mathcal{G}\ \omega\ (upd\text{-}bins\ bs\ k\ (Complete_L\ k\ x\ bs\ i))\ (i+1)$
$\neg\ i \geq length\ (items\ (bs\ !\ k)) \Longrightarrow x = items\ (bs!k)\ !\ i \Longrightarrow next\text{-}symbol\ x = Some\ a \Longrightarrow$
$a \notin nonterminals\ \mathcal{G} \Longrightarrow k < length\ \omega \Longrightarrow Earley_L\text{-}bin'\ k\ \mathcal{G}\ \omega\ bs\ i = Earley_L\text{-}bin'\ k\ \mathcal{G}\ \omega\ (upd\text{-}bins\ bs\ (k+1)\ (Scan_L\ k\ \omega\ a\ x\ i))\ (i+1)$
$\neg\ i \geq length\ (items\ (bs\ !\ k)) \Longrightarrow x = items\ (bs!k)\ !\ i \Longrightarrow next\text{-}symbol\ x = Some\ a \Longrightarrow$
$a \notin nonterminals\ \mathcal{G} \Longrightarrow \neg\ k < length\ \omega \Longrightarrow Earley_L\text{-}bin'\ k\ \mathcal{G}\ \omega\ bs\ i = Earley_L\text{-}bin'\ k\ \mathcal{G}\ \omega\ bs\ (i+1)$
$\neg\ i \geq length\ (items\ (bs\ !\ k)) \Longrightarrow x = items\ (bs!k)\ !\ i \Longrightarrow next\text{-}symbol\ x = Some\ a \Longrightarrow$
$a \in nonterminals\ \mathcal{G} \Longrightarrow Earley_L\text{-}bin'\ k\ \mathcal{G}\ \omega\ bs\ i = Earley_L\text{-}bin'\ k\ \mathcal{G}\ \omega\ (upd\text{-}bins\ bs\ k\ (Predict_L\ k\ \mathcal{G}\ a))\ (i+1)$
⟨*proof*⟩

lemma *Earley*$_L$*-bin′-induct*[*case-names Base Complete*$_F$ *Scan*$_F$ *Pass Predict*$_F$]:
assumes $(k, \mathcal{G}, \omega, bs) \in wf\text{-}earley\text{-}input$
assumes *base*: $\bigwedge k\ \mathcal{G}\ \omega\ bs\ i.\ i \geq length\ (items\ (bs\ !\ k)) \Longrightarrow P\ k\ \mathcal{G}\ \omega\ bs\ i$
assumes *complete*: $\bigwedge k\ \mathcal{G}\ \omega\ bs\ i\ x.\ \neg\ i \geq length\ (items\ (bs\ !\ k)) \Longrightarrow x = items\ (bs\ !\ k)\ !\ i \Longrightarrow$
$next\text{-}symbol\ x = None \Longrightarrow P\ k\ \mathcal{G}\ \omega\ (upd\text{-}bins\ bs\ k\ (Complete_L\ k\ x\ bs\ i))\ (i+1) \Longrightarrow P\ k\ \mathcal{G}\ \omega\ bs\ i$
assumes *scan*: $\bigwedge k\ \mathcal{G}\ \omega\ bs\ i\ x\ a.\ \neg\ i \geq length\ (items\ (bs\ !\ k)) \Longrightarrow x = items\ (bs\ !\ k)\ !\ i \Longrightarrow$
$next\text{-}symbol\ x = Some\ a \Longrightarrow a \notin nonterminals\ \mathcal{G} \Longrightarrow k < length\ \omega \Longrightarrow$
$P\ k\ \mathcal{G}\ \omega\ (upd\text{-}bins\ bs\ (k+1)\ (Scan_L\ k\ \omega\ a\ x\ i))\ (i+1) \Longrightarrow P\ k\ \mathcal{G}\ \omega\ bs\ i$
assumes *pass*: $\bigwedge k\ \mathcal{G}\ \omega\ bs\ i\ x\ a.\ \neg\ i \geq length\ (items\ (bs\ !\ k)) \Longrightarrow x = items\ (bs\ !\ k)\ !\ i \Longrightarrow$
$next\text{-}symbol\ x = Some\ a \Longrightarrow a \notin nonterminals\ \mathcal{G} \Longrightarrow \neg\ k < length\ \omega \Longrightarrow$

$P\ k\ \mathcal{G}\ \omega\ bs\ (i{+}1) \Longrightarrow P\ k\ \mathcal{G}\ \omega\ bs\ i$
assumes *predict*: $\bigwedge k\ \mathcal{G}\ \omega\ bs\ i\ x\ a.\ \neg\ i \geq length\ (items\ (bs\ !\ k)) \Longrightarrow x = items\ (bs\ !\ k)\ !\ i \Longrightarrow$
$next\text{-}symbol\ x = Some\ a \Longrightarrow a \in nonterminals\ \mathcal{G} \Longrightarrow$
$P\ k\ \mathcal{G}\ \omega\ (upd\text{-}bins\ bs\ k\ (Predict_L\ k\ \mathcal{G}\ a))\ (i{+}1) \Longrightarrow P\ k\ \mathcal{G}\ \omega\ bs\ i$
shows $P\ k\ \mathcal{G}\ \omega\ bs\ i$
⟨*proof*⟩

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

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

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

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

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

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

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

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

lemma *nth-Earley$_L$-bin′-eq*:
assumes $(k, \mathcal{G}, \omega, bs) \in$ *wf-earley-input*
shows $l < k \Longrightarrow$ *Earley$_L$-bin′* $k\ \mathcal{G}\ \omega\ bs\ i\ !\ l = bs\ !\ l$
⟨*proof*⟩

lemma *set-items-Earley$_L$-bin′-eq*:
assumes $(k, \mathcal{G}, \omega, bs) \in$ *wf-earley-input*
shows $l < k \Longrightarrow$ *set* (*items* (*Earley$_L$-bin′* $k\ \mathcal{G}\ \omega\ bs\ i\ !\ l$)) = *set* (*items* ($bs\ !\ l$))
⟨*proof*⟩

lemma *bins-upto-k0-Earley$_L$-bin′-eq*:
assumes $(k, \mathcal{G}, \omega, bs) \in$ *wf-earley-input*
shows *bins-upto* (*Earley$_L$-bin* $k\ \mathcal{G}\ \omega\ bs$) $k\ 0$ = *bins-upto* $bs\ k\ 0$
⟨*proof*⟩

lemma *wf-earley-input-Init$_L$*:
assumes $k \leq$ *length* ω
shows $(k, \mathcal{G}, \omega,$ *Init$_L$* $\mathcal{G}\ \omega) \in$ *wf-earley-input*
⟨*proof*⟩

lemma *length-bins-Init$_L$*[*simp*]:
length (*Init$_L$* $\mathcal{G}\ \omega$) = *length* $\omega + 1$
⟨*proof*⟩

lemma *wf-earley-input-Earley$_L$-bins*[*simp*]:
assumes $k \leq$ *length* ω
shows $(k, \mathcal{G}, \omega,$ *Earley$_L$-bins* $k\ \mathcal{G}\ \omega) \in$ *wf-earley-input*
⟨*proof*⟩

lemma *length-Earley$_L$-bins*[*simp*]:
assumes $k \leq$ *length* ω
shows *length* (*Earley$_L$-bins* $k\ \mathcal{G}\ \omega$) = *length* (*Init$_L$* $\mathcal{G}\ \omega$)
⟨*proof*⟩

lemma *wf-bins-Earley$_L$-bins*[*simp*]:
assumes $k \leq$ *length* ω
shows *wf-bins* $\mathcal{G}\ \omega$ (*Earley$_L$-bins* $k\ \mathcal{G}\ \omega$)
⟨*proof*⟩

lemma *wf-bins-Earley$_L$*:
wf-bins $\mathcal{G}\ \omega$ (*Earley$_L$* $\mathcal{G}\ \omega$)
⟨*proof*⟩

8.6 Soundness

lemma *Init$_L$-eq-Init$_F$*:
bins (*Init$_L$* $\mathcal{G}\ \omega$) = *Init$_F$* $\mathcal{G}$
⟨*proof*⟩

lemma *Scan$_L$-sub-Scan$_F$*:
assumes *wf-bins* $\mathcal{G}$ ω *bs bins bs* $\subseteq$ *I x* $\in$ *set* (*items* (*bs* ! *k*)) *k* < *length bs k* < *length* ω
assumes *next-symbol x* = *Some a*
shows *set* (*items* (*Scan$_L$ k* ω *a x pre*)) $\subseteq$ *Scan$_F$ k* ω *I*
⟨*proof*⟩

lemma *Predict$_L$-sub-Predict$_F$*:
assumes *wf-bins* $\mathcal{G}$ ω *bs bins bs* $\subseteq$ *I x* $\in$ *set* (*items* (*bs* ! *k*)) *k* < *length bs*
assumes *next-symbol x* = *Some X*
shows *set* (*items* (*Predict$_L$ k* $\mathcal{G}$ *X*)) $\subseteq$ *Predict$_F$ k* $\mathcal{G}$ *I*
⟨*proof*⟩

lemma *Complete$_L$-sub-Complete$_F$*:
assumes *wf-bins* $\mathcal{G}$ ω *bs bins bs* $\subseteq$ *I y* $\in$ *set* (*items* (*bs* ! *k*)) *k* < *length bs*
assumes *next-symbol y* = *None*
shows *set* (*items* (*Complete$_L$ k y bs red*)) $\subseteq$ *Complete$_F$ k I*
⟨*proof*⟩

lemma *sound-Scan$_L$*:
assumes *wf-bins* $\mathcal{G}$ ω *bs bins bs* $\subseteq$ *I x* $\in$ *set* (*items* (*bs*!*k*)) *k* < *length bs k* < *length* ω
assumes *next-symbol x* = *Some a* $\forall$ *x* $\in$ *I*. *wf-item* $\mathcal{G}$ ω *x* $\forall$ *x* $\in$ *I*. *sound-item* $\mathcal{G}$ ω *x*
shows $\forall$ *x* $\in$ *set* (*items* (*Scan$_L$ k* ω *a x i*)). *sound-item* $\mathcal{G}$ ω *x*
⟨*proof*⟩

lemma *sound-Predict$_L$*:
assumes *wf-bins* $\mathcal{G}$ ω *bs bins bs* $\subseteq$ *I x* $\in$ *set* (*items* (*bs*!*k*)) *k* < *length bs*
assumes *next-symbol x* = *Some X* $\forall$ *x* $\in$ *I*. *wf-item* $\mathcal{G}$ ω *x* $\forall$ *x* $\in$ *I*. *sound-item* $\mathcal{G}$ ω *x*
shows $\forall$ *x* $\in$ *set* (*items* (*Predict$_L$ k* $\mathcal{G}$ *X*)). *sound-item* $\mathcal{G}$ ω *x*
⟨*proof*⟩

lemma *sound-Complete$_L$*:
assumes *wf-bins* $\mathcal{G}$ ω *bs bins bs* $\subseteq$ *I y* $\in$ *set* (*items* (*bs*!*k*)) *k* < *length bs*
assumes *next-symbol y* = *None* $\forall$ *x* $\in$ *I*. *wf-item* $\mathcal{G}$ ω *x* $\forall$ *x* $\in$ *I*. *sound-item* $\mathcal{G}$ ω *x*
shows $\forall$ *x* $\in$ *set* (*items* (*Complete$_L$ k y bs i*)). *sound-item* $\mathcal{G}$ ω *x*
⟨*proof*⟩

lemma *sound-Earley$_L$-bin′*:
assumes (*k*, $\mathcal{G}$, ω, *bs*) $\in$ *wf-earley-input*
assumes $\forall$ *x* $\in$ *bins bs*. *sound-item* $\mathcal{G}$ ω *x*
shows $\forall$ *x* $\in$ *bins* (*Earley$_L$-bin′ k* $\mathcal{G}$ ω *bs i*). *sound-item* $\mathcal{G}$ ω *x*
⟨*proof*⟩

lemma *sound-Earley$_L$-bin*:
assumes (*k*, $\mathcal{G}$, ω, *bs*) $\in$ *wf-earley-input*

assumes $\forall x \in$ *bins bs. sound-item* $\mathcal{G}$ ω x
shows $\forall x \in$ *bins* (*Earley*$_L$*-bin* k $\mathcal{G}$ ω *bs*). *sound-item* $\mathcal{G}$ ω x
⟨*proof*⟩

lemma *Earley*$_L$*-bin′-sub-Earley*$_F$*-bin*:
assumes (k, $\mathcal{G}$, ω, *bs*) $\in$ *wf-earley-input*
assumes *bins bs* $\subseteq I$
shows *bins* (*Earley*$_L$*-bin′* k $\mathcal{G}$ ω *bs i*) $\subseteq$ *Earley*$_F$*-bin* k $\mathcal{G}$ ω I
⟨*proof*⟩

lemma *Earley*$_L$*-bin-sub-Earley*$_F$*-bin*:
assumes (k, $\mathcal{G}$, ω, *bs*) $\in$ *wf-earley-input*
assumes *bins bs* $\subseteq I$
shows *bins* (*Earley*$_L$*-bin* k $\mathcal{G}$ ω *bs*) $\subseteq$ *Earley*$_F$*-bin* k $\mathcal{G}$ ω I
⟨*proof*⟩

lemma *Earley*$_L$*-bins-sub-Earley*$_F$*-bins*:
assumes $k \leq$ *length* ω
shows *bins* (*Earley*$_L$*-bins* k $\mathcal{G}$ ω) $\subseteq$ *Earley*$_F$*-bins* k $\mathcal{G}$ ω
⟨*proof*⟩

lemma *Earley*$_L$*-sub-Earley*$_F$:
bins (*Earley*$_L$ $\mathcal{G}$ ω) $\subseteq$ *Earley*$_F$ $\mathcal{G}$ ω
⟨*proof*⟩

theorem *soundness-Earley*$_L$:
assumes *recognizing* (*bins* (*Earley*$_L$ $\mathcal{G}$ ω)) $\mathcal{G}$ ω
shows $\mathcal{G} \vdash [\mathfrak{S}\ \mathcal{G}] \Rightarrow^* \omega$
⟨*proof*⟩

8.7 Completeness

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

lemma *impossible-complete-item*:
assumes *sound-item* $\mathcal{G}$ ω x *is-complete* x *start-item* $x = k$ *end-item* $x = k$ *nonempty-derives* $\mathcal{G}$
shows *False*
⟨*proof*⟩

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

lemma *Complete$_F$-Un-eq-nonterminal*:
assumes $\forall x \in I.$ *wf-item* $\mathcal{G}$ ω x $\forall x \in I.$ *sound-item* $\mathcal{G}$ ω x
assumes *nonempty-derives* $\mathcal{G}$ *wf-item* $\mathcal{G}$ ω z
assumes *end-item* $z = k$ *next-symbol* $z \neq$ *None*
shows *Complete$_F$* k $(I \cup \{z\})$ = *Complete$_F$* k I
⟨*proof*⟩

lemma *wf-item-in-kth-bin*:
wf-bins $\mathcal{G}$ ω *bs* $\Longrightarrow$ $x \in$ *bins bs* $\Longrightarrow$ *end-item* $x = k \Longrightarrow x \in$ *set* (*items* (*bs* ! k))
⟨*proof*⟩

lemma *Complete$_F$-bins-upto-eq-bins*:
assumes *wf-bins* $\mathcal{G}$ ω *bs* $k <$ *length bs* $i \geq$ *length* (*items* (*bs* ! k))
shows *Complete$_F$* k (*bins-upto bs* k i) = *Complete$_F$* k (*bins bs*)
⟨*proof*⟩

lemma *Complete$_F$-sub-bins-Un-Complete$_L$*:
assumes *Complete$_F$* k $I \subseteq$ *bins bs* $I \subseteq$ *bins bs* *is-complete* z *wf-bins* $\mathcal{G}$ ω *bs* *wf-item* $\mathcal{G}$ ω z
shows *Complete$_F$* k $(I \cup \{z\}) \subseteq$ *bins bs* $\cup$ *set* (*items* (*Complete$_L$* k z *bs red*))
⟨*proof*⟩

lemma *Complete$_L$-eq-start-item*:
bs ! *start-item* y = *bs′* ! *start-item* $y \Longrightarrow$ *Complete$_L$* k y *bs red* = *Complete$_L$* k y *bs′ red*
⟨*proof*⟩

lemma *kth-bin-bins-upto-empty*:
assumes *wf-bins* $\mathcal{G}$ ω *bs* $k <$ *length bs*
shows *bin* (*bins-upto bs k 0*) k = {}
⟨*proof*⟩

lemma *Earley$_L$-bin′-mono*:
assumes $(k, \mathcal{G}, \omega, bs) \in$ *wf-earley-input*
shows *bins bs* $\subseteq$ *bins* (*Earley$_L$-bin′* k $\mathcal{G}$ ω *bs* i)
⟨*proof*⟩

lemma *Earley$_F$-bin-step-sub-Earley$_L$-bin′*:
assumes $(k, \mathcal{G}, \omega, bs) \in$ *wf-earley-input*
assumes *Earley$_F$-bin-step* k $\mathcal{G}$ ω (*bins-upto bs* k i) $\subseteq$ *bins bs*
assumes $\forall x \in$ *bins bs.* *sound-item* $\mathcal{G}$ ω x *is-word* $\mathcal{G}$ ω *nonempty-derives* $\mathcal{G}$
shows *Earley$_F$-bin-step* k $\mathcal{G}$ ω (*bins bs*) $\subseteq$ *bins* (*Earley$_L$-bin′* k $\mathcal{G}$ ω *bs* i)
⟨*proof*⟩

lemma *Earley$_F$-bin-step-sub-Earley$_L$-bin*:
assumes $(k, \mathcal{G}, \omega, bs) \in$ *wf-earley-input*
assumes *Earley$_F$-bin-step* k $\mathcal{G}$ ω (*bins-upto bs k 0*) $\subseteq$ *bins bs*
assumes $\forall x \in$ *bins bs.* *sound-item* $\mathcal{G}$ ω x *is-word* $\mathcal{G}$ ω *nonempty-derives* $\mathcal{G}$
shows *Earley$_F$-bin-step* k $\mathcal{G}$ ω (*bins bs*) $\subseteq$ *bins* (*Earley$_L$-bin* k $\mathcal{G}$ ω *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*, $\mathcal{G}$, ω, *as*) ∈ *wf-earley-input*
assumes *bins-eq-items as bs wf-bins* $\mathcal{G}$ ω *as*
shows *bins-eq-items* (*Earley*$_L$*-bin′ k* $\mathcal{G}$ ω *as i*) (*Earley*$_L$*-bin′ k* $\mathcal{G}$ ω *bs i*)
⟨*proof*⟩

lemma *Earley*$_L$*-bin′-idem*:
assumes (*k*, $\mathcal{G}$, ω, *bs*) ∈ *wf-earley-input*
assumes *i* ≤ *j* ∀ *x* ∈ *bins bs. sound-item* $\mathcal{G}$ ω *x nonempty-derives* $\mathcal{G}$
shows *bins* (*Earley*$_L$*-bin′ k* $\mathcal{G}$ ω (*Earley*$_L$*-bin′ k* $\mathcal{G}$ ω *bs i*) *j*) = *bins* (*Earley*$_L$*-bin′ k* $\mathcal{G}$ ω *bs i*)
⟨*proof*⟩

lemma *Earley*$_L$*-bin-idem*:
assumes (*k*, $\mathcal{G}$, ω, *bs*) ∈ *wf-earley-input*
assumes ∀ *x* ∈ *bins bs. sound-item* $\mathcal{G}$ ω *x nonempty-derives* $\mathcal{G}$
shows *bins* (*Earley*$_L$*-bin k* $\mathcal{G}$ ω (*Earley*$_L$*-bin k* $\mathcal{G}$ ω *bs*)) = *bins* (*Earley*$_L$*-bin k* $\mathcal{G}$ ω *bs*)
⟨*proof*⟩

lemma *funpower-Earley*$_F$*-bin-step-sub-Earley*$_L$*-bin*:
assumes (*k*, $\mathcal{G}$, ω, *bs*) ∈ *wf-earley-input*
assumes *Earley*$_F$*-bin-step k* $\mathcal{G}$ ω (*bins-upto bs k 0*) ⊆ *bins bs* ∀ *x* ∈ *bins bs. sound-item* $\mathcal{G}$ ω *x*
assumes *is-word* $\mathcal{G}$ ω *nonempty-derives* $\mathcal{G}$
shows *funpower* (*Earley*$_F$*-bin-step k* $\mathcal{G}$ ω) *n* (*bins bs*) ⊆ *bins* (*Earley*$_L$*-bin k* $\mathcal{G}$ ω *bs*)
⟨*proof*⟩

lemma *Earley*$_F$*-bin-sub-Earley*$_L$*-bin*:
assumes (*k*, $\mathcal{G}$, ω, *bs*) ∈ *wf-earley-input*
assumes *Earley*$_F$*-bin-step k* $\mathcal{G}$ ω (*bins-upto bs k 0*) ⊆ *bins bs* ∀ *x* ∈ *bins bs. sound-item* $\mathcal{G}$ ω *x*
assumes *is-word* $\mathcal{G}$ ω *nonempty-derives* $\mathcal{G}$
shows *Earley*$_F$*-bin k* $\mathcal{G}$ ω (*bins bs*) ⊆ *bins* (*Earley*$_L$*-bin k* $\mathcal{G}$ ω *bs*)
⟨*proof*⟩

lemma *Earley*$_F$*-bins-sub-Earley*$_L$*-bins*:
assumes *k* ≤ *length* ω
assumes *is-word* $\mathcal{G}$ ω *nonempty-derives* $\mathcal{G}$
shows *Earley*$_F$*-bins k* $\mathcal{G}$ ω ⊆ *bins* (*Earley*$_L$*-bins k* $\mathcal{G}$ ω)
⟨*proof*⟩

lemma *Earley$_F$-sub-Earley$_L$*:
assumes *is-word* $\mathcal{G}$ ω ε*-free* $\mathcal{G}$
shows *Earley$_F$* $\mathcal{G}$ ω $\subseteq$ *bins* (*Earley$_L$* $\mathcal{G}$ ω)
⟨*proof*⟩

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

8.8 Correctness

theorem *Earley-eq-Earley$_L$*:
assumes *is-word* $\mathcal{G}$ ω ε*-free* $\mathcal{G}$
shows *Earley* $\mathcal{G}$ ω = *bins* (*Earley$_L$* $\mathcal{G}$ ω)
⟨*proof*⟩

lemma *correctness-recognizer*:
assumes *is-word* $\mathcal{G}$ ω ε*-free* $\mathcal{G}$
shows *recognizer* $\mathcal{G}$ ω $\longleftrightarrow$ $\mathcal{G} \vdash [\mathfrak{S}\ \mathcal{G}] \Rightarrow^* \omega$ (**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* ⇒ *bool* **where**
predicts x ≡ *start-item x* = *end-item x* ∧ *dot-item x* = *0*

definition *scans* :: *'a list* ⇒ *nat* ⇒ *'a item* ⇒ *'a item* ⇒ *bool* **where**
scans ω *k x y* ≡ *y* = *inc-item x k* ∧ (∃ *a*. *next-symbol x* = *Some a* ∧ ω!(*k*−*1*) = *a*)

definition *completes* :: *nat* ⇒ *'a item* ⇒ *'a item* ⇒ *'a item* ⇒ *bool* **where**
completes k x y z ≡ *y* = *inc-item x k* ∧ *is-complete z* ∧ *start-item z* = *end-item x* ∧
(∃ *N*. *next-symbol x* = *Some N* ∧ *N* = *lhs-item z*)

definition *sound-null-ptr* :: *'a item* × *pointer* ⇒ *bool* **where**
sound-null-ptr e ≡ (*snd e* = *Null* ⟶ *predicts* (*fst e*))

definition *sound-pre-ptr* :: *'a list* ⇒ *'a bins* ⇒ *nat* ⇒ *'a item* × *pointer* ⇒ *bool* **where**
sound-pre-ptr ω bs k e ≡ ∀ *pre. snd e* = *Pre pre* ⟶
k > *0* ∧ *pre* < *length* (*bs*!(*k*−*1*)) ∧ *scans ω k* (*fst* (*bs*!(*k*−*1*)!*pre*)) (*fst e*)

definition *sound-prered-ptr* :: *'a bins* ⇒ *nat* ⇒ *'a item* × *pointer* ⇒ *bool* **where**
sound-prered-ptr bs k e ≡ ∀ *p ps k' pre red. snd e* = *PreRed p ps* ∧ (*k'*, *pre*, *red*) ∈ *set* (*p*#*ps*) ⟶
k' < *k* ∧ *pre* < *length* (*bs*!*k'*) ∧ *red* < *length* (*bs*!*k*) ∧ *completes k* (*fst* (*bs*!*k'*!*pre*)) (*fst e*) (*fst* (*bs*!*k*!*red*))

definition *sound-ptrs* :: *'a list* ⇒ *'a bins* ⇒ *bool* **where**
sound-ptrs ω bs ≡ ∀ *k* < *length bs*. ∀ *e* ∈ *set* (*bs*!*k*).
sound-null-ptr e ∧ *sound-pre-ptr ω bs k e* ∧ *sound-prered-ptr bs k e*

definition *mono-red-ptr* :: *'a bins* ⇒ *bool* **where**
mono-red-ptr bs ≡ ∀ *k* < *length bs*. ∀ *i* < *length* (*bs*!*k*).
∀ *k' pre red ps. snd* (*bs*!*k*!*i*) = *PreRed* (*k'*, *pre*, *red*) *ps* ⟶ *red* < *i*

lemma *nth-item-upd-bin*:
n < *length es* ⟹ *fst* (*upd-bin e es* ! *n*) = *fst* (*es*!*n*)
⟨*proof*⟩

lemma *upd-bin-append*:
fst e ∉ *set* (*items es*) ⟹ *upd-bin e es* = *es* @ [*e*]
⟨*proof*⟩

lemma *upd-bin-null-pre*:
fst e ∈ *set* (*items es*) ⟹ *snd e* = *Null* ∨ *snd e* = *Pre pre* ⟹ *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* ∄ *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'*) ∧ (∀ *j* < *length es'*. *i*≠*j* ⟶ *es'*!*j* = *es*!*j*) ∧ *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* < *length bs es* = *bs*!*k distinct* (*items es*)
assumes ∀ *k′ pre red ps. snd e* = *PreRed* (*k′*, *pre*, *red*) *ps* ⟶ *red* < *length es*
shows *mono-red-ptr* (*bs*[*k* := *upd-bin e es*])
⟨*proof*⟩

lemma *sound-mono-ptrs-upds-bin*:
assumes *sound-ptrs* ω *bs mono-red-ptr bs k* < *length bs b* = *bs*!*k distinct* (*items b*)
assumes ∀ *e* ∈ *set es. sound-null-ptr e* ∧ *sound-pre-ptr* ω *bs k e* ∧ *sound-prered-ptr bs k e*
assumes ∀ *e* ∈ *set es.* ∀ *k′ pre red ps. snd e* = *PreRed* (*k′*, *pre*, *red*) *ps* ⟶ *red* < *length b*
shows *sound-ptrs* ω (*bs*[*k* := *upds-bin es b*]) ∧ *mono-red-ptr* (*bs*[*k* := *upds-bin es b*])
⟨*proof*⟩

lemma *sound-mono-ptrs-Earley*$_L$*-bin′*:
assumes (*k*, $\mathcal{G}$, ω, *bs*) ∈ *wf-earley-input*
assumes *sound-ptrs* ω *bs* ∀ *x* ∈ *bins bs. sound-item* $\mathcal{G}$ ω *x*
assumes *mono-red-ptr bs*
assumes *nonempty-derives* $\mathcal{G}$
shows *sound-ptrs* ω (*Earley*$_L$*-bin′ k* $\mathcal{G}$ ω *bs i*) ∧ *mono-red-ptr* (*Earley*$_L$*-bin′ k* $\mathcal{G}$ ω *bs i*)
⟨*proof*⟩

lemma *sound-mono-ptrs-Earley*$_L$*-bin*:
assumes (*k*, $\mathcal{G}$, ω, *bs*) ∈ *wf-earley-input*
assumes *sound-ptrs* ω *bs* ∀ *x* ∈ *bins bs. sound-item* $\mathcal{G}$ ω *x*
assumes *mono-red-ptr bs*
assumes *nonempty-derives* $\mathcal{G}$
shows *sound-ptrs* ω (*Earley*$_L$*-bin k* $\mathcal{G}$ ω *bs*) ∧ *mono-red-ptr* (*Earley*$_L$*-bin k* $\mathcal{G}$ ω *bs*)
⟨*proof*⟩

lemma *sound-ptrs-Init*$_L$:
sound-ptrs ω (*Init*$_L$ $\mathcal{G}$ ω)
⟨*proof*⟩

lemma *mono-red-ptr-Init*$_L$:
mono-red-ptr (*Init*$_L$ $\mathcal{G}$ ω)
⟨*proof*⟩

lemma *sound-mono-ptrs-Earley*$_L$*-bins*:
assumes *k* ≤ *length* ω *nonempty-derives* $\mathcal{G}$
shows *sound-ptrs* ω (*Earley*$_L$*-bins k* $\mathcal{G}$ ω) ∧ *mono-red-ptr* (*Earley*$_L$*-bins k* $\mathcal{G}$ ω)
⟨*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)
⟨*proof*⟩

9.2 Common Definitions

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

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

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

fun *wf-rule-tree* :: *'a cfg* ⇒ *'a tree* ⇒ *bool* **where**
wf-rule-tree - (Leaf a) ⟷ *True*
| *wf-rule-tree* $\mathcal{G}$ (*Branch N ts*) ⟷ (
(∃ *r* ∈ *set* ($\mathfrak{R}$ $\mathcal{G}$). *N* = *lhs-rule r* ∧ *map root ts* = *rhs-rule r*) ∧
(∀ *t* ∈ *set ts*. *wf-rule-tree* $\mathcal{G}$ *t*))

fun *wf-item-tree* :: *'a cfg* ⇒ *'a item* ⇒ *'a tree* ⇒ *bool* **where**
wf-item-tree $\mathcal{G}$ - (*Leaf a*) ⟷ *True*
| *wf-item-tree* $\mathcal{G}$ *x* (*Branch N ts*) ⟷ (
N = *lhs-item x* ∧ *map root ts* = *take* (*dot-item x*) (*rhs-item x*) ∧
(∀ *t* ∈ *set ts*. *wf-rule-tree* $\mathcal{G}$ *t*))

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

9.3 foldl lemmas

lemma *foldl-add-nth*:
k < *length xs* ⟹ *foldl* (+) *z* (*map length* (*take k xs*)) + *length* (*xs*!*k*) = *foldl* (+) *z* (*map length* (*take* (*k*+*1*) *xs*))
⟨*proof*⟩

lemma *foldl-acc-mono*:
a ≤ *b* ⟹ *foldl* (+) *a xs* ≤ *foldl* (+) *b xs* **for** *a* :: *nat*
⟨*proof*⟩

lemma *foldl-ge-z-nth*:
j < *length xs* ⟹ *z* + *length* (*xs*!*j*) ≤ *foldl* (+) *z* (*map length* (*take* (*j*+*1*) *xs*))
⟨*proof*⟩

lemma *foldl-add-nth-ge*:
$i \leq j \Longrightarrow j < length\ xs \Longrightarrow foldl\ (+)\ z\ (map\ length\ (take\ i\ xs)) + length\ (xs!j) \leq foldl\ (+)\ z\ (map\ length\ (take\ (j+1)\ xs))$
⟨*proof*⟩

lemma *foldl-ge-acc*:
$foldl\ (+)\ z\ (map\ length\ xs) \geq z$
⟨*proof*⟩

lemma *foldl-take-mono*:
$i \leq j \Longrightarrow foldl\ (+)\ z\ (map\ length\ (take\ i\ xs)) \leq foldl\ (+)\ z\ (map\ length\ (take\ j\ xs))$
⟨*proof*⟩

9.4 Parse tree

```
partial-function (option) build-tree' :: 'a bins ⇒ 'a list ⇒ nat ⇒ nat ⇒ 'a tree
option where
  build-tree' bs ω k i = (
    let e = bs!k!i in (
    case snd e of
      Null ⇒ Some (Branch (lhs-item (fst e)) []) — start building sub-tree
    | Pre pre ⇒ ( — add sub-tree starting from terminal
        do {
          t ← build-tree' bs ω (k−1) pre;
          case t of
            Branch N ts ⇒ Some (Branch N (ts @ [Leaf (ω!(k−1))]))
          | - ⇒ undefined — impossible case
        })
    | PreRed (k', pre, red) - ⇒ ( — add sub-tree starting from non-terminal
        do {
          t ← build-tree' bs ω k' pre;
          case t of
            Branch N ts ⇒
              do {
                t ← build-tree' bs ω k red;
                Some (Branch N (ts @ [t]))
              }
          | - ⇒ undefined — impossible case
        })
  ))
```

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

```
definition build-tree :: 'a cfg ⇒ 'a list ⇒ 'a bins ⇒ 'a tree option where
  build-tree 𝒢 ω bs = (
    let k = length bs − 1 in (
    case filter-with-index (λx. is-finished 𝒢 ω x) (items (bs!k)) of
      [] ⇒ None
```

| (-, i)#- $\Rightarrow$ *build-tree′ bs ω k i*
))

lemma *build-tree′-simps*[*simp*]:
$e = bs!k!i \Longrightarrow snd\ e = Null \Longrightarrow$ *build-tree′ bs ω k i* = *Some* (*Branch* (*lhs-item* (*fst e*)) [])
$e = bs!k!i \Longrightarrow snd\ e = Pre\ pre \Longrightarrow$ *build-tree′ bs ω* $(k-1)$ *pre* = *None* $\Longrightarrow$
build-tree′ bs ω k i = *None*
$e = bs!k!i \Longrightarrow snd\ e = Pre\ pre \Longrightarrow$ *build-tree′ bs ω* $(k-1)$ *pre* = *Some* (*Branch N ts*) $\Longrightarrow$
build-tree′ bs ω k i = *Some* (*Branch N* (*ts* @ [*Leaf* ($\omega!(k-1)$)]))
$e = bs!k!i \Longrightarrow snd\ e = Pre\ pre \Longrightarrow$ *build-tree′ bs ω* $(k-1)$ *pre* = *Some* (*Leaf a*) $\Longrightarrow$
build-tree′ bs ω k i = *undefined*
$e = bs!k!i \Longrightarrow snd\ e = PreRed\ (k', pre, red)\ reds \Longrightarrow$ *build-tree′ bs ω k′ pre* = *None* $\Longrightarrow$
build-tree′ bs ω k i = *None*
$e = bs!k!i \Longrightarrow snd\ e = PreRed\ (k', pre, red)\ reds \Longrightarrow$ *build-tree′ bs ω k′ pre* = *Some* (*Branch N ts*) $\Longrightarrow$
build-tree′ bs ω k red = *None* $\Longrightarrow$ *build-tree′ bs ω k i* = *None*
$e = bs!k!i \Longrightarrow snd\ e = PreRed\ (k', pre, red)\ reds \Longrightarrow$ *build-tree′ bs ω k′ pre* = *Some* (*Leaf a*) $\Longrightarrow$
build-tree′ bs ω k i = *undefined*
$e = bs!k!i \Longrightarrow snd\ e = PreRed\ (k', pre, red)\ reds \Longrightarrow$ *build-tree′ bs ω k′ pre* = *Some* (*Branch N ts*) $\Longrightarrow$
build-tree′ bs ω k red = *Some t* $\Longrightarrow$
build-tree′ bs ω 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 ∧
$k < length\ bs$ ∧
$k \leq length\ \omega$ ∧
$i < length\ (bs!k)$
}

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-tree-input*
assumes $e = bs!k!i$ *snd* $e =$ *PreRed* (k', pre, red) *ps*
shows $(bs, \omega, k', pre) \in$ *wf-tree-input*
⟨*proof*⟩

lemma *wf-tree-input-prered-red*:
assumes $(bs, \omega, k, i) \in$ *wf-tree-input*
assumes $e = bs!k!i$ *snd* $e =$ *PreRed* (k', pre, red) *ps*
shows $(bs, \omega, k, red) \in$ *wf-tree-input*
⟨*proof*⟩

lemma *build-tree′-induct*:
assumes $(bs, \omega, k, i) \in$ *wf-tree-input*
assumes $\bigwedge bs\ \omega\ k\ i.$
$(\bigwedge e\ pre.\ e = bs!k!i \Longrightarrow snd\ e = Pre\ pre \Longrightarrow P\ bs\ \omega\ (k-1)\ pre) \Longrightarrow$
$(\bigwedge e\ k'\ pre\ red\ ps.\ e = bs!k!i \Longrightarrow snd\ e = PreRed\ (k', pre, red)\ ps \Longrightarrow P\ bs\ \omega\ k'\ pre) \Longrightarrow$
$(\bigwedge e\ k'\ pre\ red\ ps.\ e = bs!k!i \Longrightarrow snd\ e = PreRed\ (k', pre, red)\ ps \Longrightarrow P\ bs\ \omega\ k\ red) \Longrightarrow$
$P\ bs\ \omega\ k\ i$
shows $P\ bs\ \omega\ k\ i$
⟨*proof*⟩

lemma *build-tree′-termination*:
assumes $(bs, \omega, k, i) \in$ *wf-tree-input*
shows $\exists N\ ts.$ *build-tree′* $bs\ \omega\ k\ i = Some\ (Branch\ N\ ts)$
⟨*proof*⟩

lemma *wf-item-tree-build-tree′*:
assumes $(bs, \omega, k, i) \in$ *wf-tree-input*
assumes *wf-bins* $\mathcal{G}\ \omega\ bs$
assumes *build-tree′* $bs\ \omega\ k\ i = Some\ t$
shows *wf-item-tree* $\mathcal{G}$ $(fst\ (bs!k!i))\ t$
⟨*proof*⟩

lemma *wf-yield-build-tree′*:
assumes $(bs, \omega, k, i) \in$ *wf-tree-input*
assumes *wf-bins* $\mathcal{G}\ \omega\ bs$
assumes *build-tree′* $bs\ \omega\ k\ i = Some\ t$
shows *wf-yield* ω $(fst\ (bs!k!i))\ t$
⟨*proof*⟩

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

corollary *wf-rule-root-yield-build-tree-Earley$_L$*:
assumes *ε-free* $\mathcal{G}$
assumes *build-tree* $\mathcal{G}$ ω (*Earley$_L$* $\mathcal{G}$ ω) = *Some t*
shows *wf-rule-tree* $\mathcal{G}$ *t* $\wedge$ *root t* = $\mathfrak{S}$ $\mathcal{G}$ $\wedge$ *yield t* = ω
⟨*proof*⟩

theorem *correctness-build-tree-Earley$_L$*:
assumes *is-word* $\mathcal{G}$ ω *ε-free* $\mathcal{G}$
shows ($\exists$ *t*. *build-tree* $\mathcal{G}$ ω (*Earley$_L$* $\mathcal{G}$ ω) = *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* | *S* | *X* | *Y* | *Z*

10.2 $O(n^3)$ ambiguous grammars

10.2.1 S -> SS | 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 -> aS | 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 -> aSa | 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 -> Sa | 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 -> SX, X -> Y | Z, Y -> a, Z -> 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*, *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* (λ*b*. *fold* (+) (*map* (λ*e*. *size-pointer e*) *b*) *0*) *bs*) *0*

export-code *Earley*$_L$ *build-tree rules1 cfg1 rules2 cfg2 rules3 cfg3 rules4 cfg4 rules5 cfg5 inp size-bins size-pointers* **in** *Scala*

value *size-bins* (*Earley*$_L$ *cfg1 inp*)
value *size-pointers* (*Earley*$_L$ *cfg1 inp*)

value *size-bins* (*Earley*$_L$ *cfg2 inp*)
value *size-pointers* (*Earley*$_L$ *cfg2 inp*)

value *size-bins* (*Earley*$_L$ *cfg3 inp*)
value *size-pointers* (*Earley*$_L$ *cfg3 inp*)

value *size-bins* (*Earley*$_L$ *cfg4 inp*)
value *size-pointers* (*Earley*$_L$ *cfg4 inp*)

value *size-bins* (*Earley*$_L$ *cfg5 inp*)
value *size-pointers* (*Earley*$_L$ *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).