Earley

Martin Rau

December 9, 2023

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

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

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

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

lemma *subset-setmonotone*: $setmonotone\ f \Longrightarrow X \subseteq f\ X$
⟨*proof*⟩

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

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

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

definition *continuous* :: $('a\ set \Rightarrow 'b\ set) \Rightarrow bool$
where
$continuous\ f = (\forall\ C.\ chain\ C \longrightarrow (chain\ (f\ o\ C) \wedge f\ (natUnion\ C) = natUnion\ (f\ o\ C)))$

lemma *natUnion-upperbound*:
$(\bigwedge n.\ f\ n \subseteq G) \Longrightarrow (natUnion\ f) \subseteq G$
⟨*proof*⟩

lemma *funpower-upperbound*:
$(\bigwedge I.\ I \subseteq G \Longrightarrow f\ I \subseteq G) \Longrightarrow I \subseteq G \Longrightarrow funpower\ f\ n\ I \subseteq G$
⟨*proof*⟩

lemma *limit-upperbound*:
$(\bigwedge I.\ I \subseteq G \Longrightarrow f\ I \subseteq G) \Longrightarrow I \subseteq G \Longrightarrow limit\ f\ I \subseteq G$
⟨*proof*⟩

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

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

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

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

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 = (∃ *F*. *f* = *pointbase F*)

lemma *chain-implies-mono*: *chain C* ⟹ *mono C*
⟨*proof*⟩

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

lemma *natUnion-subset*: (⋀ *n*. ∃ *m*. *f n* ⊆ *g m*) ⟹ *natUnion f* ⊆ *natUnion g*
⟨*proof*⟩

lemma *natUnion-eq*[*case-names Subset Superset*]:
(⋀ *n*. ∃ *m*. *f n* ⊆ *g m*) ⟹ (⋀ *n*. ∃ *m*. *g n* ⊆ *f m*) ⟹ *natUnion f* = *natUnion g*
⟨*proof*⟩

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

definition *regular* :: ('*a set* ⇒ '*a set*) ⇒ *bool*
where
regular f = (*setmonotone f* ∧ *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* ⟹ *limit f* (*limit f I*) = *limit f I*
⟨*proof*⟩

definition *mk-regular1* :: ('*b* ⇒ '*a* ⇒ *bool*) ⇒ ('*b* ⇒ '*a* ⇒ '*a*) ⇒ '*a set* ⇒ '*a set*
where
mk-regular1 P F I = *I* ∪ { *F q x* | *q x*. *x* ∈ *I* ∧ *P q x* }

definition *mk-regular2* :: ('*b* ⇒ '*a* ⇒ '*a* ⇒ *bool*) ⇒ ('*b* ⇒ '*a* ⇒ '*a* ⇒ '*a*) ⇒ '*a set* ⇒ '*a set* **where**
mk-regular2 P F I = *I* ∪ { *F q x y* | *q x y*. *x* ∈ *I* ∧ *y* ∈ *I* ∧ *P q x y* }

end

theory *CFG*
imports *Main*
begin

2 Adjusted content from AFP/LocalLexing

type-synonym *'a rule* = *'a* × *'a list*

type-synonym *'a rules* = *'a rule list*

type-synonym *'a sentence* = *'a list*

datatype *'a cfg* =
CFG (𝔑 : *'a list*) (𝔗 : *'a list*) (ℜ : *'a rules*) (𝔖 : *'a*)

definition *disjunct-symbols* :: *'a cfg* ⇒ *bool* **where**
disjunct-symbols 𝒢 ≡ *set* (𝔑 𝒢) ∩ *set* (𝔗 𝒢) = {}

definition *valid-startsymbol* :: *'a cfg* ⇒ *bool* **where**
valid-startsymbol 𝒢 ≡ 𝔖 𝒢 ∈ *set* (𝔑 𝒢)

definition *valid-rules* :: *'a cfg* ⇒ *bool* **where**
valid-rules 𝒢 ≡ ∀ (*N*, *α*) ∈ *set* (ℜ 𝒢). *N* ∈ *set* (𝔑 𝒢) ∧ (∀ *s* ∈ *set α*. *s* ∈ *set* (𝔑 𝒢) ∪ *set* (𝔗 𝒢))

definition *distinct-rules* :: *'a cfg* ⇒ *bool* **where**
distinct-rules 𝒢 ≡ *distinct* (ℜ 𝒢)

definition *wf-𝒢* :: *'a cfg* ⇒ *bool* **where**
wf-𝒢 𝒢 ≡ *disjunct-symbols* 𝒢 ∧ *valid-startsymbol* 𝒢 ∧ *valid-rules* 𝒢 ∧ *distinct-rules* 𝒢

lemmas *wf-𝒢-defs* = *wf-𝒢-def valid-rules-def valid-startsymbol-def disjunct-symbols-def distinct-rules-def*

definition *is-terminal* :: *'a cfg* ⇒ *'a* ⇒ *bool* **where**
is-terminal 𝒢 *x* ≡ *x* ∈ *set* (𝔗 𝒢)

definition *is-nonterminal* :: *'a cfg* ⇒ *'a* ⇒ *bool* **where**
is-nonterminal 𝒢 *x* ≡ *x* ∈ *set* (𝔑 𝒢)

definition *is-symbol* :: *'a cfg* ⇒ *'a* ⇒ *bool* **where**
is-symbol 𝒢 *x* ≡ *is-terminal* 𝒢 *x* ∨ *is-nonterminal* 𝒢 *x*

definition *wf-sentence* :: *'a cfg* ⇒ *'a sentence* ⇒ *bool* **where**
wf-sentence 𝒢 *ω* ≡ ∀ *x* ∈ *set ω*. *is-symbol* 𝒢 *x*

lemma *is-nonterminal-startsymbol*:
wf-𝒢 𝒢 ⟹ *is-nonterminal* 𝒢 (𝔖 𝒢)

⟨*proof*⟩

definition *is-word* :: *'a cfg* ⇒ *'a sentence* ⇒ *bool* **where**
is-word $\mathcal{G}$ $\omega \equiv \forall x \in set\ \omega.$ *is-terminal* $\mathcal{G}$ x

definition *derives1* :: *'a cfg* ⇒ *'a sentence* ⇒ *'a sentence* ⇒ *bool* **where**
derives1 $\mathcal{G}$ u $v \equiv \exists\ x\ y\ N\ \alpha.$
$u = x$ @ $[N]$ @ $y \wedge$
$v = x$ @ α @ $y \wedge$
$(N, \alpha) \in set\ (\mathfrak{R}\ \mathcal{G})$

definition *derivations1* :: *'a cfg* ⇒ (*'a sentence* × *'a sentence*) *set* **where**
derivations1 $\mathcal{G} \equiv \{\ (u,v) \mid u\ v.$ *derives1* $\mathcal{G}$ u $v\ \}$

definition *derivations* :: *'a cfg* ⇒ (*'a sentence* × *'a sentence*) *set* **where**
derivations $\mathcal{G} \equiv$ (*derivations1* $\mathcal{G}$)$\widehat{}*$

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

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 sentence* ⇒ *nat* ⇒ *'a rule* ⇒ *'a sentence* ⇒ *bool* **where**
Derives1 $\mathcal{G}$ *u i r v* ≡ ∃ *x y N* α.
u = *x* @ [*N*] @ *y* ∧
v = *x* @ α @ *y* ∧
(*N*, α) ∈ *set* ($\mathfrak{R}$ $\mathcal{G}$) ∧ *r* = (*N*, 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* ⟹ ∃ *i r*. *Derives1* $\mathcal{G}$ *u i r v*
⟨*proof*⟩

fun *Derivation* :: *'a cfg* ⇒ *'a sentence* ⇒ *'a derivation* ⇒ *'a sentence* ⇒ *bool* **where**
Derivation - a [] *b* = (*a* = *b*)
| *Derivation* $\mathcal{G}$ *a* (*d*#*D*) *b* = (∃ *x*. *Derives1* $\mathcal{G}$ *a* (*fst d*) (*snd d*) *x* ∧ *Derivation* $\mathcal{G}$ *x D b*)

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

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

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

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

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

lemma *is-terminal-nonterminal*: *wf-*$\mathcal{G}$ $\mathcal{G}$ ⟹ *is-terminal* $\mathcal{G}$ *x* ⟹ *is-nonterminal* $\mathcal{G}$ *x* ⟹ *False*
⟨*proof*⟩

lemma *is-word-is-terminal*: *i* < *length u* ⟹ *is-word* $\mathcal{G}$ *u* ⟹ *is-terminal* $\mathcal{G}$ (*u* ! *i*)

⟨*proof*⟩

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

lemma *Derivation-implies-append*:
Derivation $\mathcal{G}$ *a D b* ⟹ *Derivation* $\mathcal{G}$ *b E c* ⟹ *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*′ ⟹ *Derivation* $\mathcal{G}$ (*a*@*b*) (*map* (λ(*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*′ ⟹ *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 ∃ *F*. *Derivation* $\mathcal{G}$ *a F* (*b* @ *c*′ @ *d*)
⟨*proof*⟩

lemma *derives1-if-valid-rule*:
(*N*, *α*) ∈ *set* (ℜ $\mathcal{G}$) ⟹ *derives1* $\mathcal{G}$ [*N*] *α*
⟨*proof*⟩

lemma *derives-if-valid-rule*:
(*N*, *α*) ∈ *set* (ℜ $\mathcal{G}$) ⟹ *derives* $\mathcal{G}$ [*N*] *α*
⟨*proof*⟩

lemma *Derivation-from-empty*:
Derivation $\mathcal{G}$ [] *D a* ⟹ *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-𝔖1*:
assumes *Derivation* $\mathcal{G}$ $[\mathfrak{S}\ \mathcal{G}]$ D ω *is-word* $\mathcal{G}$ ω *wf-*$\mathcal{G}$ $\mathcal{G}$
shows $\exists \alpha\ E$. *Derivation* $\mathcal{G}$ α E $\omega \wedge$ $(\mathfrak{S}\ \mathcal{G},\alpha) \in$ *set* $(\mathfrak{R}\ \mathcal{G})$
⟨*proof*⟩

end
theory *Earley*
imports
Derivations
begin

5 Slices

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

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

lemma *slice-append-aux*:
Suc b ≤ *c* ⟹ *slice (Suc b) c (x # xs) = slice b (c−1) xs*
⟨*proof*⟩

lemma *slice-concat*:
a ≤ *b* ⟹ *b* ≤ *c* ⟹ *slice a b xs @ slice b c xs = slice a c xs*
⟨*proof*⟩

lemma *slice-concat-Ex*:
a ≤ *c* ⟹ *slice a c xs = ys @ zs* ⟹ ∃ *b. ys = slice a b xs* ∧ *zs = slice b c xs* ∧ *a* ≤ *b* ∧ *b* ≤ *c*
⟨*proof*⟩

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

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

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

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

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

6 Earley recognizer

6.1 Earley items

definition *rule-head* :: *'a rule* ⇒ *'a* **where**
rule-head ≡ *fst*

definition *rule-body* :: *'a rule* ⇒ *'a list* **where**
rule-body ≡ *snd*

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

definition *item-rule-head* :: *'a item* ⇒ *'a* **where**
item-rule-head x ≡ *rule-head* (*item-rule x*)

definition *item-rule-body* :: *'a item* ⇒ *'a sentence* **where**
item-rule-body x ≡ *rule-body* (*item-rule x*)

definition *item-α* :: *'a item* ⇒ *'a sentence* **where**
item-α x ≡ *take* (*item-dot x*) (*item-rule-body x*)

definition *item-β* :: *'a item* ⇒ *'a sentence* **where**
item-β x ≡ *drop* (*item-dot x*) (*item-rule-body x*)

definition *is-complete* :: *'a item* ⇒ *bool* **where**
is-complete x ≡ *item-dot x* ≥ *length* (*item-rule-body x*)

definition *next-symbol* :: *'a item* ⇒ *'a option* **where**
next-symbol x ≡ *if is-complete x then None else Some* (*item-rule-body x* ! *item-dot x*)

lemmas *item-defs* = *item-rule-head-def item-rule-body-def item-α-def item-β-def rule-head-def rule-body-def*

definition *is-finished* :: *'a cfg* ⇒ *'a sentence* ⇒ *'a item* ⇒ *bool* **where**
is-finished $\mathcal{G}$ ω *x* ≡

item-rule-head $x = \mathfrak{S}\ \mathcal{G} \wedge$
item-origin $x = 0 \wedge$
item-end $x =$ *length* $\omega \wedge$
is-complete x

definition *recognizing* :: $'a$ *item set* $\Rightarrow$ $'a$ *cfg* $\Rightarrow$ $'a$ *sentence* $\Rightarrow$ *bool* **where**
recognizing $I\ \mathcal{G}\ \omega \equiv \exists x \in I.$ *is-finished* $\mathcal{G}\ \omega\ x$

inductive-set *Earley* :: $'a$ *cfg* $\Rightarrow$ $'a$ *sentence* $\Rightarrow$ $'a$ *item set*
for $\mathcal{G}$ :: $'a$ *cfg* **and** ω :: $'a$ *sentence* **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* (*rule-head* 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* (*item-rule-head* y) $\Longrightarrow$
Item $r_x\ (b_x + 1)\ i\ k \in$ *Earley* $\mathcal{G}\ \omega$

6.2 Well-formedness

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

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

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

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

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

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

6.3 Soundness

definition *sound-item* :: *'a cfg* ⇒ *'a sentence* ⇒ *'a item* ⇒ *bool* **where**
sound-item $\mathcal{G}\ \omega\ x \equiv$ *derives* $\mathcal{G}$ [*item-rule-head x*] (*slice* (*item-origin x*) (*item-end x*) ω @ *item-*β *x*)

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

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

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

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

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

theorem *soundness-Earley*:
assumes *recognizing* (*Earley* $\mathcal{G}\ \omega$) $\mathcal{G}\ \omega$
shows *derives* $\mathcal{G}$ [$\mathfrak{S}\ \mathcal{G}$] ω

⟨*proof*⟩

6.4 Completeness

definition *partially-completed* :: *nat* ⇒ *'a cfg* ⇒ *'a sentence* ⇒ *'a item set* ⇒ (*'a derivation* ⇒ *bool*) ⇒ *bool* **where**
partially-completed k 𝒢 *ω E P* ≡ ∀ *r b i' i j x a D*.
i ≤ *j* ∧ *j* ≤ *k* ∧ *k* ≤ *length ω* ∧
x = *Item r b i' i* ∧ *x* ∈ *E* ∧ *next-symbol x* = *Some a* ∧
Derivation 𝒢 [*a*] *D* (*slice i j ω*) ∧ *P D* ⟶
Item r (*b*+*1*) *i' j* ∈ *E*

lemma *partially-completed-upto*:
assumes *j* ≤ *k k* ≤ *length ω*
assumes *x* = *Item* (*N*,*α*) *d i j x* ∈ *I* ∀ *x* ∈ *I*. *wf-item* 𝒢 *ω x*
assumes *Derivation* 𝒢 (*item-β x*) *D* (*slice j k ω*)
assumes *partially-completed k* 𝒢 *ω I* (λ*D'*. *length D'* ≤ *length D*)
shows *Item* (*N*,*α*) (*length α*) *i k* ∈ *I*
⟨*proof*⟩

lemma *partially-completed-Earley-k*:
assumes *wf-*𝒢 𝒢
shows *partially-completed k* 𝒢 *ω* (*Earley* 𝒢 *ω*) (λ-. *True*)
⟨*proof*⟩

lemma *partially-completed-Earley*:
*wf-*𝒢 𝒢 ⟹ *partially-completed* (*length ω*) 𝒢 *ω* (*Earley* 𝒢 *ω*) (λ-. *True*)
⟨*proof*⟩

theorem *completeness-Earley*:
assumes *derives* 𝒢 [𝔖 𝒢] *ω is-word* 𝒢 *ω wf-*𝒢 𝒢
shows *recognizing* (*Earley* 𝒢 *ω*) 𝒢 *ω*
⟨*proof*⟩

6.5 Correctness

theorem *correctness-Earley*:
assumes *wf-*𝒢 𝒢 *is-word* 𝒢 *ω*
shows *recognizing* (*Earley* 𝒢 *ω*) 𝒢 *ω* ⟷ *derives* 𝒢 [𝔖 𝒢] *ω*
⟨*proof*⟩

6.6 Finiteness

lemma *finiteness-empty*:
set (𝔑 𝒢) = {} ⟹ *finite* { *x* | *x*. *wf-item* 𝒢 *ω 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 | x. *wf-item* $\mathcal{G}$ ω x }
⟨*proof*⟩

lemma *finiteness-UNIV-wf-item*:
finite { x | x. *wf-item* $\mathcal{G}$ ω x }
⟨*proof*⟩

theorem *finiteness-Earley*:
finite (*Earley* $\mathcal{G}$ ω)
⟨*proof*⟩

end
theory *Earley-Fixpoint*
imports
Earley
Limit
begin

7 Earley recognizer

7.1 Earley fixpoint

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 ≡ *Item* (*item-rule x*) (*item-dot x* + *1*) (*item-origin x*) *k*

definition *bin* :: *'a item set* ⇒ *nat* ⇒ *'a item set* **where**
bin I k ≡ { x . $x \in I \wedge$ *item-end* $x = k$ }

definition $Init_F$ :: *'a cfg* ⇒ *'a item set* **where**
$Init_F$ $\mathcal{G}$ ≡ { *init-item r 0* | r. $r \in$ *set* ($\mathfrak{R}$ $\mathcal{G}$) $\wedge$ *fst* $r = (\mathfrak{S}\ \mathcal{G})$ }

definition $Scan_F$ :: *nat* ⇒ *'a sentence* ⇒ *'a item set* ⇒ *'a item set* **where**
$Scan_F$ k ω I ≡ { *inc-item x* (*k*+*1*) | x a.
$x \in$ *bin I k* $\wedge$
$\omega!k = a \wedge$
$k <$ *length* $\omega \wedge$
next-symbol x = *Some a* }

definition $Predict_F$ :: *nat* ⇒ *'a cfg* ⇒ *'a item set* ⇒ *'a item set* **where**
$Predict_F$ k $\mathcal{G}$ I ≡ { *init-item r k* | r x.
$r \in$ *set* ($\mathfrak{R}$ $\mathcal{G}$) $\wedge$
$x \in$ *bin I k* $\wedge$
next-symbol x = *Some* (*rule-head r*) }

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

definition $Earley_F\text{-}bin\text{-}step :: nat \Rightarrow 'a\ cfg \Rightarrow 'a\ sentence \Rightarrow 'a\ item\ set \Rightarrow 'a\ item\ set$ **where**
$Earley_F\text{-}bin\text{-}step\ k\ \mathcal{G}\ \omega\ I \equiv I \cup Scan_F\ k\ \omega\ I \cup Complete_F\ k\ I \cup Predict_F\ k\ \mathcal{G}\ I$

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

fun $Earley_F\text{-}bins :: nat \Rightarrow 'a\ cfg \Rightarrow 'a\ sentence \Rightarrow 'a\ item\ set$ **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 $Earley_F :: 'a\ cfg \Rightarrow 'a\ sentence \Rightarrow 'a\ item\ set$ **where**
$Earley_F\ \mathcal{G}\ \omega \equiv Earley_F\text{-}bins\ (length\ \omega)\ \mathcal{G}\ \omega$

7.2 Monotonicity and Absorption

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

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

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

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

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

lemma $Scan_F\text{-}bin\text{-}absorb$:
$Scan_F\ k\ \omega\ (bin\ I\ k) = Scan_F\ k\ \omega\ I$
⟨*proof*⟩

lemma $Predict_F\text{-}bin\text{-}absorb$:

$Predict_F\ k\ \mathcal{G}\ (bin\ I\ k) = Predict_F\ k\ \mathcal{G}\ I$
⟨*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*:
$I \subseteq J \Longrightarrow Complete_F\ k\ I \subseteq Complete_F\ k\ J$
⟨*proof*⟩

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* $k\ \mathcal{G}\ \omega\ J$
⟨*proof*⟩

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

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

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

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

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

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

$Earley_F$-bin-step k $\mathcal{G}$ ω I $\subseteq$ $Earley_F$-bin k $\mathcal{G}$ ω I
⟨*proof*⟩

lemma $Scan_F$-$Earley_F$-bin-mono:
$Scan_F$ k ω $I \subseteq Earley_F$-bin k $\mathcal{G}$ ω I
⟨*proof*⟩

lemma $Predict_F$-$Earley_F$-bin-mono:
$Predict_F$ k $\mathcal{G}$ $I \subseteq Earley_F$-bin k $\mathcal{G}$ ω I
⟨*proof*⟩

lemma $Complete_F$-$Earley_F$-bin-mono:
$Complete_F$ k $I \subseteq Earley_F$-bin k $\mathcal{G}$ ω I
⟨*proof*⟩

lemma $Earley_F$-bin-mono:
$I \subseteq Earley_F$-bin k $\mathcal{G}$ ω I
⟨*proof*⟩

lemma $Init_F$-sub-$Earley_F$-bins:
$Init_F$ $\mathcal{G} \subseteq Earley_F$-bins n $\mathcal{G}$ ω
⟨*proof*⟩

7.3 Soundness

lemma $Init_F$-sub-Earley:
$Init_F$ $\mathcal{G} \subseteq Earley$ $\mathcal{G}$ ω
⟨*proof*⟩

lemma $Scan_F$-sub-Earley:
assumes $I \subseteq Earley$ $\mathcal{G}$ ω
shows $Scan_F$ k ω $I \subseteq Earley$ $\mathcal{G}$ ω
⟨*proof*⟩

lemma $Predict_F$-sub-Earley:
assumes $I \subseteq Earley$ $\mathcal{G}$ ω
shows $Predict_F$ k $\mathcal{G}$ $I \subseteq Earley$ $\mathcal{G}$ ω
⟨*proof*⟩

lemma $Complete_F$-sub-Earley:
assumes $I \subseteq Earley$ $\mathcal{G}$ ω
shows $Complete_F$ k $I \subseteq Earley$ $\mathcal{G}$ ω
⟨*proof*⟩

lemma $Earley_F$-bin-step-sub-Earley:
assumes $I \subseteq Earley$ $\mathcal{G}$ ω
shows $Earley_F$-bin-step k $\mathcal{G}$ ω $I \subseteq Earley$ $\mathcal{G}$ ω
⟨*proof*⟩

lemma *Earley$_F$-bin-sub-Earley*:
assumes $I \subseteq$ *Earley* $\mathcal{G}\ \omega$
shows *Earley$_F$-bin* $k\ \mathcal{G}\ \omega\ I \subseteq$ *Earley* $\mathcal{G}\ \omega$
⟨*proof*⟩

lemma *Earley$_F$-bins-sub-Earley*:
shows *Earley$_F$-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 *derives* $\mathcal{G}$ [$\mathfrak{S}\ \mathcal{G}$] ω
⟨*proof*⟩

7.4 Completeness

definition *prev-symbol* :: *'a item* ⇒ *'a option* **where**
prev-symbol x ≡ *if item-dot x* = *0 then None else Some* (*item-rule-body x* ! (*item-dot x* − *1*))

definition *base* :: *'a sentence* ⇒ *'a item set* ⇒ *nat* ⇒ *'a item set* **where**
base ω *I k* ≡ { *x* . *x* ∈ *I* ∧ *item-end x* = *k* ∧ *k* > *0* ∧ *prev-symbol x* = *Some* (ω!(*k*−*1*)) }

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* ω (*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* ω (*Earley* $\mathcal{G}\ \omega$) $k \subseteq I$
assumes *wf-*$\mathcal{G}$ $\mathcal{G}$ *is-word* $\mathcal{G}\ \omega$
shows *base* ω (*Earley* $\mathcal{G}\ \omega$) (*k*+*1*) ⊆ *bin* (*Earley$_F$-bin* $k\ \mathcal{G}\ \omega\ I$) (*k*+*1*)
⟨*proof*⟩

lemma *Earley$_F$-bin-k-sub-Earley$_F$-bins*:
assumes *wf-*$\mathcal{G}$ $\mathcal{G}$ *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 *wf-𝒢 𝒢 is-word 𝒢 ω*
shows *Earley 𝒢 ω ⊆ Earley$_F$ 𝒢 ω*
⟨*proof*⟩

theorem *completeness-Earley$_F$*:
assumes *derives 𝒢 [𝔖 𝒢] ω is-word 𝒢 ω wf-𝒢 𝒢*
shows *recognizing (Earley$_F$ 𝒢 ω) 𝒢 ω*
⟨*proof*⟩

7.5 Correctness

theorem *Earley-eq-Earley$_F$*:
assumes *wf-𝒢 𝒢 is-word 𝒢 ω*
shows *Earley 𝒢 ω = Earley$_F$ 𝒢 ω*
⟨*proof*⟩

theorem *correctness-Earley$_F$*:
assumes *wf-𝒢 𝒢 is-word 𝒢 ω*
shows *recognizing (Earley$_F$ 𝒢 ω) 𝒢 ω ⟷ derives 𝒢 [𝔖 𝒢] ω*
⟨*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 ∈ set xs ⟹ P x ⟹ filter-with-index P xs ≠ []
⟨*proof*⟩

lemma *filter-with-index′-Ex-first*:
(∃ x i xs′. filter-with-index′ n P xs = (x, i)#xs′) ⟷ (∃ x ∈ set xs. P x)
⟨*proof*⟩

lemma *filter-with-index-Ex-first*:
(∃ x i xs′. filter-with-index P xs = (x, i)#xs′) ⟷ (∃ x ∈ 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

datatype *'a entry* =
Entry (item : 'a item) (pointer : pointer)

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

definition *items :: 'a bin ⇒ 'a item list* **where**
items b ≡ map item b

lemma *length-items-eq*:
‹*length (items b) = length b*›
⟨*proof*⟩

definition *pointers :: 'a bin ⇒ pointer list* **where**
pointers b ≡ map pointer 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 ≡ ⋃ { set (items (bs!k)) | k. k < length bs }

definition *bin-upto :: 'a bin ⇒ nat ⇒ 'a item set* **where**
bin-upto b i ≡ { items b ! j | j. j < i ∧ j < length (items b) }

definition *bins-upto :: 'a bins ⇒ nat ⇒ nat ⇒ 'a item set* **where**
bins-upto bs k i ≡ ⋃ { set (items (bs ! l)) | l. l < k } ∪ bin-upto (bs ! k) i

definition *wf-bin-items :: 'a cfg ⇒ 'a sentence ⇒ nat ⇒ 'a item list ⇒ bool* **where**
wf-bin-items 𝒢 ω k xs ≡ ∀ x ∈ set xs. wf-item 𝒢 ω x ∧ item-end x = k

definition *wf-bin :: 'a cfg ⇒ 'a sentence ⇒ nat ⇒ 'a bin ⇒ bool* **where**
wf-bin 𝒢 ω k b ≡ distinct (items b) ∧ wf-bin-items 𝒢 ω k (items b)

definition *wf-bins :: 'a cfg ⇒ 'a list ⇒ 'a bins ⇒ bool* **where**
wf-bins 𝒢 ω bs ≡ ∀ k < length bs. wf-bin 𝒢 ω k (bs!k)

definition *nonempty-derives :: 'a cfg ⇒ bool* **where**
nonempty-derives 𝒢 ≡ ∀ N. N ∈ set (𝔑 𝒢) ⟶ ¬ derives 𝒢 [N] []

definition *Init$_L$ :: 'a cfg ⇒ 'a sentence ⇒ 'a bins* **where**
Init$_L$ 𝒢 ω ≡
let rs = filter (λr. rule-head r = 𝔖 𝒢) (𝔑 𝒢) in
let b0 = map (λr. (Entry (init-item r 0) Null)) rs in
let bs = replicate (length ω + 1) ([]) in
bs[0 := b0]

definition *Scan$_L$ :: nat ⇒ 'a sentence ⇒ 'a ⇒ 'a item ⇒ nat ⇒ 'a entry list* **where**
Scan$_L$ k ω a x pre ≡
if ω!k = a then
let x′ = inc-item x (k+1) in

```
    [Entry x' (Pre pre)]
  else []

definition Predict_L :: nat ⇒ 'a cfg ⇒ 'a ⇒ 'a entry list where
  Predict_L k G X ≡
    let rs = filter (λr. rule-head r = X) (ℜ G) in
    map (λr. (Entry (init-item r k) Null)) rs

definition Complete_L :: nat ⇒ 'a item ⇒ 'a bins ⇒ nat ⇒ 'a entry list where
  Complete_L k y bs red ≡
    let orig = bs ! (item-origin y) in
    let is = filter-with-index (λx. next-symbol x = Some (item-rule-head y)) (items
orig) in
    map (λ(x, pre). (Entry (inc-item x k) (PreRed (item-origin y, pre, red) []))) is

fun bin-upd :: 'a entry ⇒ 'a bin ⇒ 'a bin where
  bin-upd e' [] = [e']
| bin-upd e' (e#es) = (
    case (e', e) of
      (Entry x (PreRed px xs), Entry y (PreRed py ys)) ⇒
        if x = y then Entry x (PreRed py (px#xs@ys)) # es
        else e # bin-upd e' es
      | - ⇒
        if item e' = item e then e # es
        else e # bin-upd e' es)

fun bin-upds :: 'a entry list ⇒ 'a bin ⇒ 'a bin where
  bin-upds [] b = b
| bin-upds (e#es) b = bin-upds es (bin-upd e b)

definition bins-upd :: 'a bins ⇒ nat ⇒ 'a entry list ⇒ 'a bins where
  bins-upd bs k es ≡ bs[k := bin-upds es (bs!k)]

partial-function (tailrec) Earley_L-bin' :: nat ⇒ 'a cfg ⇒ 'a sentence ⇒ 'a bins
⇒ nat ⇒ '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 is-terminal G a then
              if k < length ω then bins-upd bs (k+1) (Scan_L k ω a x i)
              else bs
            else bins-upd bs k (Predict_L k G a)
        | None ⇒ bins-upd bs k (Complete_L k x bs i)
      in Earley_L-bin' k G ω bs' (i+1))
```

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

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

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

8.3 Bin lemmas

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

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

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

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

lemma *nth-idem-bins-upd*:
$k \neq n \Longrightarrow bins\text{-}upd\ bs\ k\ es\ !\ n = bs\ !\ n$
⟨*proof*⟩

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

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

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

lemma *bin-upto-eq-set-items*:
$i \geq length\ b \Longrightarrow bin\text{-}upto\ b\ i = set\ (items\ b)$
⟨*proof*⟩

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

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

lemma *set-items-bin-upds*:
set (items (bin-upds es b)) = set (items b) ∪ set (items es)
⟨*proof*⟩

lemma *bins-bins-upd*:
assumes *k < length bs*
shows *bins (bins-upd bs k es) = bins bs ∪ set (items es)*
⟨*proof*⟩

lemma *kth-bin-sub-bins*:
k < length bs ⟹ set (items (bs ! k)) ⊆ bins bs
⟨*proof*⟩

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

lemma *bin-upto-Cons*:
assumes *0 < n*
shows *bin-upto (e#es) n = { item e } ∪ bin-upto es (n−1)*
⟨*proof*⟩

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

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

lemma *bins-upto-kth-nth-idem*:
assumes *l < length bs k ≤ l n < length (bs ! k)*
shows *bins-upto (bins-upd bs l es) k n = bins-upto bs k n*
⟨*proof*⟩

lemma *bins-upto-sub-bins*:
k < length bs ⟹ bins-upto bs k n ⊆ bins bs
⟨*proof*⟩

lemma *bins-upto-Suc-Un*:

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

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

lemma *distinct-bin-upd*:
$\mathit{distinct}\ (\mathit{items}\ b) \Longrightarrow \mathit{distinct}\ (\mathit{items}\ (\mathit{bin\text{-}upd}\ e\ b))$
⟨*proof*⟩

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

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

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

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

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

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

lemma *bin-eq-items-bin-upd*:
$\mathit{item}\ e \in \mathit{set}\ (\mathit{items}\ b) \Longrightarrow \mathit{items}\ (\mathit{bin\text{-}upd}\ e\ b) = \mathit{items}\ b$
⟨*proof*⟩

lemma *bin-eq-items-bin-upds*:
assumes $\mathit{set}\ (\mathit{items}\ es) \subseteq \mathit{set}\ (\mathit{items}\ b)$
shows $\mathit{items}\ (\mathit{bin\text{-}upds}\ es\ b) = \mathit{items}\ b$

⟨*proof*⟩

lemma *bins-eq-items-bins-upd*:
assumes *set* (*items es*) ⊆ *set* (*items* (*bs*!*k*))
shows *bins-eq-items* (*bins-upd bs k es*) *bs*
⟨*proof*⟩

lemma *bins-eq-items-imp-eq-bins*:
bins-eq-items bs bs′ ⟹ *bins bs* = *bins bs*′
⟨*proof*⟩

lemma *bin-eq-items-dist-bin-upd-bin*:
assumes *items a* = *items b*
shows *items* (*bin-upd e a*) = *items* (*bin-upd e b*)
⟨*proof*⟩

lemma *bin-eq-items-dist-bin-upds-bin*:
assumes *items a* = *items b*
shows *items* (*bin-upds es a*) = *items* (*bin-upds es b*)
⟨*proof*⟩

lemma *bin-eq-items-dist-bin-upd-entry*:
assumes *item e* = *item e*′
shows *items* (*bin-upd e b*) = *items* (*bin-upd e*′ *b*)
⟨*proof*⟩

lemma *bin-eq-items-dist-bin-upds-entries*:
assumes *items es* = *items es*′
shows *items* (*bin-upds es b*) = *items* (*bin-upds es*′ *b*)
⟨*proof*⟩

lemma *bins-eq-items-dist-bins-upd*:
assumes *bins-eq-items as bs items aes* = *items bes k* < *length as*
shows *bins-eq-items* (*bins-upd as k aes*) (*bins-upd bs k bes*)
⟨*proof*⟩

8.4 Well-formed bins

lemma *distinct-Scan*$_L$:
distinct (*items* (*Scan*$_L$ *k* ω *a x pre*))
⟨*proof*⟩

lemma *distinct-Predict*$_L$:
wf-$\mathcal{G}$ $\mathcal{G}$ ⟹ *distinct* (*items* (*Predict*$_L$ *k* $\mathcal{G}$ *X*))
⟨*proof*⟩

lemma *inj-on-inc-item*:
∀ *x* ∈ *A*. *item-end x* = *l* ⟹ *inj-on* (λ*x*. *inc-item x k*) *A*
⟨*proof*⟩

lemma *distinct-Complete*$_L$:
assumes *wf-bins* $\mathcal{G}$ ω *bs item-origin* $y <$ *length bs*
shows *distinct* (*items* (*Complete*$_L$ *k y bs red*))
⟨*proof*⟩

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

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

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

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

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

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

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

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

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

lemma *wf-earley-input-Scan*$_L$:
assumes (*k*, $\mathcal{G}$, ω, *bs*) ∈ *wf-earley-input* ¬ *length* (*items* (*bs* ! *k*)) ≤ *i*
assumes *x* = *items* (*bs* ! *k*) ! *i next-symbol x* = *Some a*
assumes *is-terminal* $\mathcal{G}$ *a k* < *length* ω
shows (*k*, $\mathcal{G}$, ω, *bins-upd bs* (*k*+*1*) (*Scan*$_L$ *k* ω *a x pre*)) ∈ *wf-earley-input*
⟨*proof*⟩

lemma *wf-earley-input-Predict*$_L$:
assumes (*k*, $\mathcal{G}$, ω, *bs*) ∈ *wf-earley-input* ¬ *length* (*items* (*bs* ! *k*)) ≤ *i*
assumes *x* = *items* (*bs* ! *k*) ! *i next-symbol x* = *Some a* ¬ *is-terminal* $\mathcal{G}$ *a*
shows (*k*, $\mathcal{G}$, ω, *bins-upd bs k* (*Predict*$_L$ *k* $\mathcal{G}$ *a*)) ∈ *wf-earley-input*
⟨*proof*⟩

fun *earley-measure* :: *nat* × *'a cfg* × *'a sentence* × *'a bins* ⇒ *nat* ⇒ *nat* **where**
earley-measure (*k*, $\mathcal{G}$, ω, *bs*) *i* = *card* { *x* | *x*. *wf-item* $\mathcal{G}$ ω *x* ∧ *item-end x* = *k* } − *i*

lemma *Earley*$_L$*-bin'-simps*[*simp*]:
i ≥ *length* (*items* (*bs* ! *k*)) ⟹ *Earley*$_L$*-bin' k* $\mathcal{G}$ ω *bs i* = *bs*
¬ *i* ≥ *length* (*items* (*bs* ! *k*)) ⟹ *x* = *items* (*bs*!*k*) ! *i* ⟹ *next-symbol x* = *None* ⟹
Earley$_L$*-bin' k* $\mathcal{G}$ ω *bs i* = *Earley*$_L$*-bin' k* $\mathcal{G}$ ω (*bins-upd bs k* (*Complete*$_L$ *k x bs i*)) (*i*+*1*)
¬ *i* ≥ *length* (*items* (*bs* ! *k*)) ⟹ *x* = *items* (*bs*!*k*) ! *i* ⟹ *next-symbol x* = *Some a* ⟹
is-terminal $\mathcal{G}$ *a* ⟹ *k* < *length* ω ⟹ *Earley*$_L$*-bin' k* $\mathcal{G}$ ω *bs i* = *Earley*$_L$*-bin' k* $\mathcal{G}$ ω (*bins-upd bs* (*k*+*1*) (*Scan*$_L$ *k* ω *a x i*)) (*i*+*1*)
¬ *i* ≥ *length* (*items* (*bs* ! *k*)) ⟹ *x* = *items* (*bs*!*k*) ! *i* ⟹ *next-symbol x* = *Some*

$a \Longrightarrow$
$\mathit{is\text{-}terminal}\ \mathcal{G}\ a \Longrightarrow \neg\ k < \mathit{length}\ \omega \Longrightarrow \mathit{Earley}_L\text{-}\mathit{bin}'\ k\ \mathcal{G}\ \omega\ \mathit{bs}\ i = \mathit{Earley}_L\text{-}\mathit{bin}'\ k\ \mathcal{G}\ \omega\ \mathit{bs}\ (i{+}1)$
$\neg\ i \geq \mathit{length}\ (\mathit{items}\ (\mathit{bs}\ !\ k)) \Longrightarrow x = \mathit{items}\ (\mathit{bs}!k)\ !\ i \Longrightarrow \mathit{next\text{-}symbol}\ x = \mathit{Some}\ a \Longrightarrow$
$\neg\ \mathit{is\text{-}terminal}\ \mathcal{G}\ a \Longrightarrow \mathit{Earley}_L\text{-}\mathit{bin}'\ k\ \mathcal{G}\ \omega\ \mathit{bs}\ i = \mathit{Earley}_L\text{-}\mathit{bin}'\ k\ \mathcal{G}\ \omega\ (\mathit{bins\text{-}upd}\ \mathit{bs}\ k\ (\mathit{Predict}_L\ k\ \mathcal{G}\ a))\ (i{+}1)$
⟨*proof*⟩

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

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

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

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

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

⟨*proof*⟩

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

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

lemma *kth-Earley$_L$-bin′-bins*:
assumes $(k, \mathcal{G}, \omega, bs) \in$ *wf-earley-input*
assumes $j <$ *length* (*items* (bs ! l))
shows *items* (*Earley$_L$-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-earley-input*
shows *set* (*items* (bs ! l)) $\subseteq$ *set* (*items* (*Earley$_L$-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 *wf-*$\mathcal{G}$ $\mathcal{G}$
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* ω + *1*
⟨*proof*⟩

lemma *wf-earley-input-Earley$_L$-bins*[*simp*]:
assumes $k \leq$ *length* ω *wf-*$\mathcal{G}$ $\mathcal{G}$

shows $(k, \mathcal{G}, \omega, Earley_L\text{-}bins\ k\ \mathcal{G}\ \omega) \in wf\text{-}earley\text{-}input$
⟨*proof*⟩

lemma $length\text{-}Earley_L\text{-}bins[simp]$:
assumes $k \leq length\ \omega\ wf\text{-}\mathcal{G}\ \mathcal{G}$
shows $length\ (Earley_L\text{-}bins\ k\ \mathcal{G}\ \omega) = length\ (Init_L\ \mathcal{G}\ \omega)$
⟨*proof*⟩

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

lemma $wf\text{-}bins\text{-}Earley_L$:
$wf\text{-}\mathcal{G}\ \mathcal{G} \Longrightarrow wf\text{-}bins\ \mathcal{G}\ \omega\ (Earley_L\ \mathcal{G}\ \omega)$
⟨*proof*⟩

8.5 Soundness

lemma $Init_L\text{-}eq\text{-}Init_F$:
$bins\ (Init_L\ \mathcal{G}\ \omega) = Init_F\ \mathcal{G}$
⟨*proof*⟩

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

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

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

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

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

theorem *soundness-Earley*$_L$:

assumes *wf-$\mathcal{G}$ $\mathcal{G}$ recognizing (bins (Earley$_L$ $\mathcal{G}$ ω)) $\mathcal{G}$ ω*
shows *derives $\mathcal{G}$ [$\mathfrak{S}$ $\mathcal{G}$] ω*
⟨*proof*⟩

8.6 Completeness

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

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

lemma *Complete$_F$-Un-eq-terminal*:
assumes *next-symbol z = Some a is-terminal $\mathcal{G}$ a ∀ x ∈ I. wf-item $\mathcal{G}$ ω x wf-item $\mathcal{G}$ ω z wf-$\mathcal{G}$ $\mathcal{G}$*
shows *Complete$_F$ k (I ∪ {z}) = Complete$_F$ k I*
⟨*proof*⟩

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

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

lemma *Complete$_F$-bins-upto-eq-bins*:
assumes *wf-bins $\mathcal{G}$ ω bs k < length bs i ≥ 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 ⊆ bins bs I ⊆ bins bs is-complete z wf-bins $\mathcal{G}$ ω bs wf-item $\mathcal{G}$ ω z*
shows *Complete$_F$ k (I ∪ {z}) ⊆ bins bs ∪ set (items (Complete$_L$ k z bs red))*
⟨*proof*⟩

lemma *Complete$_L$-eq-item-origin*:
bs ! item-origin y = bs′ ! item-origin y ⟹ 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, $\mathcal{G}$, ω, *bs*) ∈ *wf-earley-input*
shows *bins bs* ⊆ *bins* (*Earley$_L$-bin′ k* $\mathcal{G}$ ω *bs i*)
⟨*proof*⟩

lemma *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 i*) ⊆ *bins bs*
assumes ∀ *x* ∈ *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*) ⊆ *bins* (*Earley$_L$-bin′ k* $\mathcal{G}$ ω *bs i*)
⟨*proof*⟩

lemma *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*
assumes ∀ *x* ∈ *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*) ⊆ *bins* (*Earley$_L$-bin k* $\mathcal{G}$ ω *bs*)
⟨*proof*⟩

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

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

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

lemma *Earley$_F$-sub-Earley$_L$*:
assumes *wf-*$\mathcal{G}$ $\mathcal{G}$ *is-word* $\mathcal{G}\ \omega$ *nonempty-derives* $\mathcal{G}$
shows *Earley$_F$* $\mathcal{G}\ \omega \subseteq$ *bins* (*Earley$_L$* $\mathcal{G}\ \omega$)
⟨*proof*⟩

theorem *completeness-Earley$_L$*:
assumes *derives* $\mathcal{G}$ [$\mathfrak{S}\ \mathcal{G}$] ω *is-word* $\mathcal{G}\ \omega$ *wf-*$\mathcal{G}$ $\mathcal{G}$ *nonempty-derives* $\mathcal{G}$
shows *recognizing* (*bins* (*Earley$_L$* $\mathcal{G}\ \omega$)) $\mathcal{G}\ \omega$
⟨*proof*⟩

8.7 Correctness

theorem *Earley-eq-Earley$_L$*:
assumes *wf-*$\mathcal{G}$ $\mathcal{G}$ *is-word* $\mathcal{G}\ \omega$ *nonempty-derives* $\mathcal{G}$
shows *Earley* $\mathcal{G}\ \omega =$ *bins* (*Earley$_L$* $\mathcal{G}\ \omega$)
⟨*proof*⟩

theorem *correctness-Earley$_L$*:
assumes *wf-*$\mathcal{G}$ $\mathcal{G}$ *is-word* $\mathcal{G}\ \omega$ *nonempty-derives* $\mathcal{G}$
shows *recognizing* (*bins* (*Earley$_L$* $\mathcal{G}\ \omega$)) $\mathcal{G}\ \omega \longleftrightarrow$ *derives* $\mathcal{G}$ [$\mathfrak{S}\ \mathcal{G}$] ω
⟨*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 ≡ *item-origin x* = *item-end x* ∧ *item-dot x* = *0*

definition *scans* :: *'a sentence* ⇒ *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* ∧ *item-origin z* = *item-end x* ∧
(∃ *N*. *next-symbol x* = *Some N* ∧ *N* = *item-rule-head z*)

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

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

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

definition *sound-ptrs* :: *'a sentence* ⇒ *'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*. *pointer* (*bs*!*k*!*i*) = *PreRed* (*k'*, *pre*, *red*) *ps* ⟶ *red* < *i*

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

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

lemma *bin-upd-null-pre*:
item e ∈ set (items es) ⟹ pointer e = Null ∨ pointer e = Pre pre ⟹ bin-upd e es = es
⟨*proof*⟩

lemma *bin-upd-prered-nop*:
assumes *distinct (items es) i < length es*
assumes *item e = item (es!i) pointer e = PreRed p ps ∄p ps. pointer (es!i) = PreRed p ps*
shows *bin-upd e es = es*
⟨*proof*⟩

lemma *bin-upd-prered-upd*:
assumes *distinct (items es) i < length es*
assumes *item e = item (es!i) pointer e = PreRed p rs pointer (es!i) = PreRed p′ rs′ bin-upd e es = es′*
shows *pointer (es′!i) = PreRed p′ (p#rs@rs′) ∧ (∀ j < length es′. i≠j ⟶ es′!j = es!j) ∧ length (bin-upd e es) = length es*
⟨*proof*⟩

lemma *sound-ptrs-bin-upd*:
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 := bin-upd e es])*
⟨*proof*⟩

lemma *mono-red-ptr-bin-upd*:
assumes *mono-red-ptr bs k < length bs es = bs!k distinct (items es)*
assumes *∀ k′ pre red ps. pointer e = PreRed (k′, pre, red) ps ⟶ red < length es*
shows *mono-red-ptr (bs[k := bin-upd e es])*
⟨*proof*⟩

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

lemma *sound-mono-ptrs-Earley$_L$-bin′*:
assumes *(k, 𝒢, ω, bs) ∈ wf-earley-input*
assumes *sound-ptrs ω bs ∀ x ∈ bins bs. sound-item 𝒢 ω x*
assumes *mono-red-ptr bs*
assumes *nonempty-derives 𝒢 wf-𝒢 𝒢*

shows *sound-ptrs* ω (*Earley$_L$-bin'* k $\mathcal{G}$ ω *bs* i) $\wedge$ *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*) $\in$ *wf-earley-input*
assumes *sound-ptrs* ω *bs* $\forall\, x \in$ *bins bs*. *sound-item* $\mathcal{G}$ ω x
assumes *mono-red-ptr bs*
assumes *nonempty-derives* $\mathcal{G}$ *wf-*$\mathcal{G}$ $\mathcal{G}$
shows *sound-ptrs* ω (*Earley$_L$-bin* k $\mathcal{G}$ ω *bs*) $\wedge$ *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 $k \leq$ *length* ω *wf-*$\mathcal{G}$ $\mathcal{G}$ *nonempty-derives* $\mathcal{G}$ *wf-*$\mathcal{G}$ $\mathcal{G}$
shows *sound-ptrs* ω (*Earley$_L$-bins* k $\mathcal{G}$ ω) $\wedge$ *mono-red-ptr* (*Earley$_L$-bins* k $\mathcal{G}$ ω)
⟨*proof*⟩

lemma *sound-mono-ptrs-Earley$_L$*:
assumes *wf-*$\mathcal{G}$ $\mathcal{G}$ *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-tree* :: *'a tree* ⇒ *'a sentence* **where**
yield-tree (*Leaf a*) = [*a*]
| *yield-tree* (*Branch - ts*) = *concat* (*map yield-tree ts*)

fun *root-tree* :: *'a tree* ⇒ *'a* **where**
root-tree (*Leaf a*) = *a*
| *root-tree* (*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* (ℜ $\mathcal{G}$). *N* = *rule-head r* ∧ *map root-tree ts* = *rule-body r*) ∧

```
    (∀ t ∈ set ts. wf-rule-tree G t))

fun wf-item-tree :: 'a cfg ⇒ 'a item ⇒ 'a tree ⇒ bool where
  wf-item-tree G - (Leaf a) ⟷ True
| wf-item-tree G x (Branch N ts) ⟷ (
    N = item-rule-head x ∧ map root-tree ts = take (item-dot x) (item-rule-body x)
∧
    (∀ t ∈ set ts. wf-rule-tree G t))

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

datatype 'a forest =
  FLeaf 'a
  | FBranch 'a 'a forest list list

fun combinations :: 'a list list ⇒ 'a list list where
  combinations [] = [[]]
| combinations (xs#xss) = [ x#cs . x <− xs, cs <− combinations xss ]

fun trees :: 'a forest ⇒ 'a tree list where
  trees (FLeaf a) = [Leaf a]
| trees (FBranch N fss) = (
    let tss = (map (λfs. concat (map (λf. trees f) fs)) fss) in
    map (λts. Branch N ts) (combinations tss)
  )

lemma list-comp-flatten:
  [ f xs . xs <− [ g xs ys . xs <− as, ys <− bs ] ] = [ f (g xs ys) . xs <− as, ys
<− bs ]
  ⟨proof⟩

lemma list-comp-flatten-Cons:
  [ x#xs . x <− as, xs <− [ xs @ ys. xs <− bs, ys <− cs ] ] = [ x#xs@ys. x <−
as, xs <− bs, ys <− cs ]
  ⟨proof⟩

lemma list-comp-flatten-append:
  [ xs@ys . xs <− [ x#xs . x <− as, xs <− bs ], ys <− cs ] = [ x#xs@ys . x <−
as, xs <− bs, ys <− cs ]
  ⟨proof⟩

lemma combinations-append:
  combinations (xss @ yss) = [ xs @ ys . xs <− combinations xss, ys <− combi-
nations yss ]
  ⟨proof⟩

lemma trees-append:
  trees (FBranch N (xss @ yss)) = (
```

let xtss = (map (λxs. concat (map (λf. trees f) xs)) xss) in
let ytss = (map (λys. concat (map (λf. trees f) ys)) yss) in
map (λts. Branch N ts) [xs @ ys . xs <− combinations xtss, ys <− combinations ytss])
⟨*proof*⟩

lemma *trees-append-singleton*:
trees (FBranch N (xss @ [ys])) = (
let xtss = (map (λxs. concat (map (λf. trees f) xs)) xss) in
let ytss = [concat (map trees ys)] in
map (λts. Branch N ts) [xs @ ys . xs <− combinations xtss, ys <− combinations ytss])
⟨*proof*⟩

lemma *trees-append-single-singleton*:
trees (FBranch N (xss @ [[y]])) = (
let xtss = (map (λxs. concat (map (λf. trees f) xs)) xss) in
map (λts. Branch N ts) [xs @ ys . xs <− combinations xtss, ys <− [[t] . t <− trees y]])
⟨*proof*⟩

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 ≤ j ⟹ j < length xs ⟹ foldl (+) z (map length (take i xs)) + length (xs!j) ≤ foldl (+) z (map length (take (j+1) xs))
⟨*proof*⟩

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

lemma *foldl-take-mono*:
i ≤ j ⟹ foldl (+) z (map length (take i xs)) ≤ foldl (+) z (map length (take j xs))
⟨*proof*⟩

9.4 Parse tree

partial-function (*option*) *build-tree′* :: *′a bins ⇒ ′a sentence ⇒ nat ⇒ nat ⇒ ′a tree option* **where**
build-tree′ bs ω k i = (
let e = *bs*!*k*!*i in* (
case pointer e of
Null ⇒ *Some* (*Branch* (*item-rule-head* (*item 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 sentence ⇒ ′a bins ⇒ ′a tree option* **where**
build-tree $\mathcal{G}$ *ω bs* = (
let k = *length bs* − *1 in* (
case filter-with-index (*λx. is-finished* $\mathcal{G}$ *ω x*) (*items* (*bs*!*k*)) *of*
[] ⇒ *None*
| (-, *i*)#- ⇒ *build-tree′ bs ω k i*
))

lemma *build-tree′-simps*[*simp*]:
e = *bs*!*k*!*i* ⟹ *pointer e* = *Null* ⟹ *build-tree′ bs ω k i* = *Some* (*Branch* (*item-rule-head* (*item e*)) [])
e = *bs*!*k*!*i* ⟹ *pointer e* = *Pre pre* ⟹ *build-tree′ bs ω* (*k*−*1*) *pre* = *None* ⟹
build-tree′ bs ω k i = *None*
e = *bs*!*k*!*i* ⟹ *pointer e* = *Pre pre* ⟹ *build-tree′ bs ω* (*k*−*1*) *pre* = *Some* (*Branch N ts*) ⟹
build-tree′ bs ω k i = *Some* (*Branch N* (*ts* @ [*Leaf* (*ω*!(*k*−*1*))]))
e = *bs*!*k*!*i* ⟹ *pointer e* = *Pre pre* ⟹ *build-tree′ bs ω* (*k*−*1*) *pre* = *Some* (*Leaf a*) ⟹
build-tree′ bs ω k i = *undefined*
e = *bs*!*k*!*i* ⟹ *pointer e* = *PreRed* (*k′*, *pre*, *red*) *reds* ⟹ *build-tree′ bs ω k′ pre*

$= None \Longrightarrow$
$build\text{-}tree'\ bs\ \omega\ k\ i = None$
$e = bs!k!i \Longrightarrow pointer\ e = PreRed\ (k',\ pre,\ red)\ reds \Longrightarrow build\text{-}tree'\ bs\ \omega\ k'\ pre$
$= Some\ (Branch\ N\ ts) \Longrightarrow$
$build\text{-}tree'\ bs\ \omega\ k\ red = None \Longrightarrow build\text{-}tree'\ bs\ \omega\ k\ i = None$
$e = bs!k!i \Longrightarrow pointer\ e = PreRed\ (k',\ pre,\ red)\ reds \Longrightarrow build\text{-}tree'\ bs\ \omega\ k'\ pre$
$= Some\ (Leaf\ a) \Longrightarrow$
$build\text{-}tree'\ bs\ \omega\ k\ i = undefined$
$e = bs!k!i \Longrightarrow pointer\ e = PreRed\ (k',\ pre,\ red)\ reds \Longrightarrow build\text{-}tree'\ bs\ \omega\ k'\ pre$
$= Some\ (Branch\ N\ ts) \Longrightarrow$
$build\text{-}tree'\ bs\ \omega\ k\ red = Some\ t \Longrightarrow$
$build\text{-}tree'\ bs\ \omega\ k\ i = Some\ (Branch\ N\ (ts\ @\ [t]))$
⟨*proof*⟩

definition *wf-tree-input* :: $('a\ bins \times 'a\ sentence \times nat \times nat)\ set$ **where**
wf-tree-input = {
$(bs,\ \omega,\ k,\ i) \mid bs\ \omega\ k\ i.$
sound-ptrs $\omega\ bs \wedge$
mono-red-ptr $bs \wedge$
$k < length\ bs \wedge$
$i < length\ (bs!k)$
}

fun *build-tree'-measure* :: $('a\ bins \times 'a\ sentence \times nat \times nat) \Rightarrow nat$ **where**
$build\text{-}tree'\text{-}measure\ (bs,\ \omega,\ k,\ i) = foldl\ (+)\ 0\ (map\ length\ (take\ k\ bs)) + i$

lemma *wf-tree-input-pre*:
assumes $(bs,\ \omega,\ k,\ i) \in wf\text{-}tree\text{-}input$
assumes $e = bs!k!i\ pointer\ e = Pre\ pre$
shows $(bs,\ \omega,\ (k-1),\ pre) \in wf\text{-}tree\text{-}input$
⟨*proof*⟩

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

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

lemma *build-tree'-induct*:
assumes $(bs,\ \omega,\ k,\ i) \in wf\text{-}tree\text{-}input$
assumes $\bigwedge bs\ \omega\ k\ i.$
$(\bigwedge e\ pre.\ e = bs!k!i \Longrightarrow pointer\ e = Pre\ pre \Longrightarrow P\ bs\ \omega\ (k-1)\ pre) \Longrightarrow$
$(\bigwedge e\ k'\ pre\ red\ ps.\ e = bs!k!i \Longrightarrow pointer\ e = PreRed\ (k',\ pre,\ red)\ ps \Longrightarrow P\ bs$

ω k' *pre*) $\Longrightarrow$
(⋀*e* k' *pre red ps*. $e = bs!k!i \Longrightarrow$ *pointer* e = *PreRed* (k', *pre*, *red*) *ps* $\Longrightarrow$ *P bs* ω *k red*) $\Longrightarrow$
P bs ω *k i*
shows *P bs* ω *k i*
⟨*proof*⟩

lemma *build-tree′-termination*:
assumes (*bs*, ω, *k*, *i*) ∈ *wf-tree-input*
shows ∃ *N ts*. *build-tree′ bs* ω *k i* = *Some* (*Branch N ts*)
⟨*proof*⟩

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

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

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

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

theorem *correctness-build-tree-Earley*$_L$:
assumes *wf-*$\mathcal{G}$ $\mathcal{G}$ *is-word* $\mathcal{G}$ ω *nonempty-derives* $\mathcal{G}$
shows (∃ *t*. *build-tree* $\mathcal{G}$ ω (*Earley*$_L$ $\mathcal{G}$ ω) = *Some t*) ⟷ *derives* $\mathcal{G}$ [$\mathfrak{S}$ $\mathcal{G}$] ω (**is** *?L* ⟷ *?R*)
⟨*proof*⟩

9.5 those, map, map option lemmas

lemma *those-map-exists*:
Some ys = those (map f xs) ⟹ y ∈ set ys ⟹ ∃ x. x ∈ set xs ∧ Some y ∈ set (map f xs)
⟨*proof*⟩

lemma *those-Some*:
(∀ x ∈ set xs. ∃ a. x = Some a) ⟷ (∃ ys. those xs = Some ys)
⟨*proof*⟩

lemma *those-Some-P*:
assumes *∀ x ∈ set xs. ∃ ys. x = Some ys ∧ (∀ y ∈ set ys. P y)*
shows *∃ yss. those xs = Some yss ∧ (∀ ys ∈ set yss. ∀ y ∈ set ys. P y)*
⟨*proof*⟩

lemma *map-Some-P*:
assumes *z ∈ set (map f xs)*
assumes *∀ x ∈ set xs. ∃ ys. f x = Some ys ∧ (∀ y ∈ set ys. P y)*
shows *∃ ys. z = Some ys ∧ (∀ y ∈ set ys. P y)*
⟨*proof*⟩

lemma *those-map-FBranch-only*:
assumes *g = (λf. case f of FBranch N fss ⇒ Some (FBranch N (fss @ [[FLeaf (ω!(k−1))]])) | - ⇒ None)*
assumes *Some fs = those (map g pres) f ∈ set fs*
assumes *∀ f ∈ set pres. ∃ N fss. f = FBranch N fss*
shows *∃ f-pre N fss. f = FBranch N (fss @ [[FLeaf (ω!(k−1))]]) ∧ f-pre = FBranch N fss ∧ f-pre ∈ set pres*
⟨*proof*⟩

lemma *those-map-Some-concat-exists*:
assumes *y ∈ set (concat ys)*
assumes *Some ys = those (map f xs)*
shows *∃ ys x. Some ys = f x ∧ y ∈ set ys ∧ x ∈ set xs*
⟨*proof*⟩

lemma *map-option-concat-those-map-exists*:
assumes *Some fs = map-option concat (those (map F xs))*
assumes *f ∈ set fs*
shows *∃ fss fs'. Some fss = those (map F xs) ∧ fs' ∈ set fss ∧ f ∈ set fs'*
⟨*proof*⟩

lemma [*partial-function-mono*]:
monotone option.le-fun option-ord
(λf. map-option concat (those (map (λ((k', pre), reds).
f ((((r, s), k'), pre), {pre}) ⟜
(λpres. those (map (λred. f ((((r, s), t), red), b ∪ {red})) reds) ⟜
(λrss. those (map (λf. case f of FBranch N fss ⇒ Some (FBranch N (fss @ [concat rss])) | - ⇒ None) pres))))

xs)))
⟨*proof*⟩

9.6 Parse trees

fun *insert-group* :: ('*a* ⇒ '*k*) ⇒ ('*a* ⇒ '*v*) ⇒ '*a* ⇒ ('*k* × '*v list*) *list* ⇒ ('*k* × '*v list*) *list* **where**
insert-group K V a [] = [(*K a*, [*V a*])]
| *insert-group K V a* ((*k*, *vs*)#*xs*) = (
if K a = *k then* (*k*, *V a* # *vs*) # *xs*
else (*k*, *vs*) # *insert-group K V a xs*
)

fun *group-by* :: ('*a* ⇒ '*k*) ⇒ ('*a* ⇒ '*v*) ⇒ '*a list* ⇒ ('*k* × '*v list*) *list* **where**
group-by K V [] = []
| *group-by K V* (*x*#*xs*) = *insert-group K V x* (*group-by K V xs*)

lemma *insert-group-cases*:
assumes (*k*, *vs*) ∈ *set* (*insert-group K V a xs*)
shows (*k* = *K a* ∧ *vs* = [*V a*]) ∨ (*k*, *vs*) ∈ *set xs* ∨ (∃(*k'*, *vs'*) ∈ *set xs*. *k'* = *k* ∧ *k* = *K a* ∧ *vs* = *V a* # *vs'*)
⟨*proof*⟩

lemma *group-by-exists-kv*:
(*k*, *vs*) ∈ *set* (*group-by K V xs*) ⟹ ∃ *x* ∈ *set xs*. *k* = *K x* ∧ (∃ *v* ∈ *set vs*. *v* = *V x*)
⟨*proof*⟩

lemma *group-by-forall-v-exists-k*:
(*k*, *vs*) ∈ *set* (*group-by K V xs*) ⟹ *v* ∈ *set vs* ⟹ ∃ *x* ∈ *set xs*. *k* = *K x* ∧ *v* = *V x*
⟨*proof*⟩

partial-function (*option*) *build-trees'* :: '*a bins* ⇒ '*a sentence* ⇒ *nat* ⇒ *nat* ⇒ *nat set* ⇒ '*a forest list option* **where**
build-trees' bs ω k i I = (
let e = *bs*!*k*!*i in* (
case pointer e of
Null ⇒ *Some* ([*FBranch* (*item-rule-head* (*item e*)) []]) — start building sub-trees
| *Pre pre* ⇒ (— add sub-trees starting from terminal
do {
pres ← *build-trees' bs ω* (*k*−*1*) *pre* {*pre*};
those (*map* (λ*f*.
case f of
FBranch N fss ⇒ *Some* (*FBranch N* (*fss* @ [[*FLeaf* (*ω*!(*k*−*1*))]]))
| - ⇒ *None* — impossible case
) *pres*)
})

```
  | PreRed p ps ⇒ ( — add sub-trees starting from non-terminal
      let ps′ = filter (λ(k′, pre, red). red ∉ I) (p#ps) in
      let gs = group-by (λ(k′, pre, red). (k′, pre)) (λ(k′, pre, red). red) ps′ in
      map-option concat (those (map (λ((k′, pre), reds).
        do {
          pres ← build-trees′ bs ω k′ pre {pre};
          rss ← those (map (λred. build-trees′ bs ω k red (I ∪ {red})) reds);
          those (map (λf.
            case f of
              FBranch N fss ⇒ Some (FBranch N (fss @ [concat rss]))
            | - ⇒ None — impossible case
          ) pres)
        }
      ) gs))
    )
))

declare build-trees′.simps [code]

definition build-trees :: 'a cfg ⇒ 'a sentence ⇒ 'a bins ⇒ 'a forest list option
where
  build-trees G ω bs = (
    let k = length bs − 1 in
    let finished = filter-with-index (λx. is-finished G ω x) (items (bs!k)) in
    map-option concat (those (map (λ(-, i). build-trees′ bs ω k i {i}) finished))
  )

lemma build-forest′-simps[simp]:
  e = bs!k!i ⟹ pointer e = Null ⟹ build-trees′ bs ω k i I = Some ([FBranch
(item-rule-head (item e)) []])
  e = bs!k!i ⟹ pointer e = Pre pre ⟹ build-trees′ bs ω (k−1) pre {pre} = None
⟹ build-trees′ bs ω k i I = None
  e = bs!k!i ⟹ pointer e = Pre pre ⟹ build-trees′ bs ω (k−1) pre {pre} = Some
pres ⟹
    build-trees′ bs ω k i I = those (map (λf. case f of FBranch N fss ⇒ Some
(FBranch N (fss @ [[FLeaf (ω!(k−1))]])) | - ⇒ None) pres)
  ⟨proof⟩

definition wf-trees-input :: ('a bins × 'a sentence × nat × nat × nat set) set
where
  wf-trees-input = {
    (bs, ω, k, i, I) | bs ω k i I.
      sound-ptrs ω bs ∧
      k < length bs ∧
      i < length (bs!k) ∧
      I ⊆ {0..<length (bs!k)} ∧
      i ∈ I
  }
```

fun *build-forest′-measure* :: (*′a bins* × *′a sentence* × *nat* × *nat* × *nat set*) ⇒ *nat*
where
build-forest′-measure (*bs*, *ω*, *k*, *i*, *I*) = *foldl* (+) *0* (*map length* (*take* (*k*+*1*) *bs*)) − *card I*

lemma *wf-trees-input-pre*:
assumes (*bs*, *ω*, *k*, *i*, *I*) ∈ *wf-trees-input*
assumes *e* = *bs*!*k*!*i pointer e* = *Pre pre*
shows (*bs*, *ω*, (*k*−*1*), *pre*, {*pre*}) ∈ *wf-trees-input*
⟨*proof*⟩

lemma *wf-trees-input-prered-pre*:
assumes (*bs*, *ω*, *k*, *i*, *I*) ∈ *wf-trees-input*
assumes *e* = *bs*!*k*!*i pointer e* = *PreRed p ps*
assumes *ps′* = *filter* (λ(*k′*, *pre*, *red*). *red* ∉ *I*) (*p*#*ps*)
assumes *gs* = *group-by* (λ(*k′*, *pre*, *red*). (*k′*, *pre*)) (λ(*k′*, *pre*, *red*). *red*) *ps′*
assumes ((*k′*, *pre*), *reds*) ∈ *set gs*
shows (*bs*, *ω*, *k′*, *pre*, {*pre*}) ∈ *wf-trees-input*
⟨*proof*⟩

lemma *wf-trees-input-prered-red*:
assumes (*bs*, *ω*, *k*, *i*, *I*) ∈ *wf-trees-input*
assumes *e* = *bs*!*k*!*i pointer e* = *PreRed p ps*
assumes *ps′* = *filter* (λ(*k′*, *pre*, *red*). *red* ∉ *I*) (*p*#*ps*)
assumes *gs* = *group-by* (λ(*k′*, *pre*, *red*). (*k′*, *pre*)) (λ(*k′*, *pre*, *red*). *red*) *ps′*
assumes ((*k′*, *pre*), *reds*) ∈ *set gs red* ∈ *set reds*
shows (*bs*, *ω*, *k*, *red*, *I* ∪ {*red*}) ∈ *wf-trees-input*
⟨*proof*⟩

lemma *build-trees′-induct*:
assumes (*bs*, *ω*, *k*, *i*, *I*) ∈ *wf-trees-input*
assumes ⋀*bs ω k i I*.
(⋀*e pre*. *e* = *bs*!*k*!*i* ⟹ *pointer e* = *Pre pre* ⟹ *P bs ω* (*k*−*1*) *pre* {*pre*}) ⟹
(⋀*e p ps ps′ gs k′ pre reds*. *e* = *bs*!*k*!*i* ⟹ *pointer e* = *PreRed p ps* ⟹
ps′ = *filter* (λ(*k′*, *pre*, *red*). *red* ∉ *I*) (*p*#*ps*) ⟹
gs = *group-by* (λ(*k′*, *pre*, *red*). (*k′*, *pre*)) (λ(*k′*, *pre*, *red*). *red*) *ps′* ⟹
((*k′*, *pre*), *reds*) ∈ *set gs* ⟹ *P bs ω k′ pre* {*pre*}) ⟹
(⋀*e p ps ps′ gs k′ pre red reds reds′*. *e* = *bs*!*k*!*i* ⟹ *pointer e* = *PreRed p ps* ⟹
ps′ = *filter* (λ(*k′*, *pre*, *red*). *red* ∉ *I*) (*p*#*ps*) ⟹
gs = *group-by* (λ(*k′*, *pre*, *red*). (*k′*, *pre*)) (λ(*k′*, *pre*, *red*). *red*) *ps′* ⟹
((*k′*, *pre*), *reds*) ∈ *set gs* ⟹ *red* ∈ *set reds* ⟹ *P bs ω k red* (*I* ∪ {*red*})) ⟹
P bs ω k i I
shows *P bs ω k i I*
⟨*proof*⟩

lemma *build-trees′-termination*:
assumes (*bs*, *ω*, *k*, *i*, *I*) ∈ *wf-trees-input*
shows ∃ *fs*. *build-trees′ bs ω k i I* = *Some fs* ∧ (∀ *f* ∈ *set fs*. ∃ *N fss*. *f* = *FBranch*

N fss)
⟨*proof*⟩

lemma *wf-item-tree-build-trees'*:
assumes (*bs*, *ω*, *k*, *i*, *I*) ∈ *wf-trees-input*
assumes *wf-bins* 𝒢 *ω bs*
assumes *k* < *length bs i* < *length* (*bs*!*k*)
assumes *build-trees' bs ω k i I* = *Some fs*
assumes *f* ∈ *set fs*
assumes *t* ∈ *set* (*trees f*)
shows *wf-item-tree* 𝒢 (*item* (*bs*!*k*!*i*)) *t*
⟨*proof*⟩

lemma *wf-yield-tree-build-trees'*:
assumes (*bs*, *ω*, *k*, *i*, *I*) ∈ *wf-trees-input*
assumes *wf-bins* 𝒢 *ω bs*
assumes *k* < *length bs i* < *length* (*bs*!*k*) *k* ≤ *length ω*
assumes *build-trees' bs ω k i I* = *Some fs*
assumes *f* ∈ *set fs*
assumes *t* ∈ *set* (*trees f*)
shows *wf-yield-tree ω* (*item* (*bs*!*k*!*i*)) *t*
⟨*proof*⟩

theorem *wf-rule-root-yield-tree-build-trees*:
assumes *wf-bins* 𝒢 *ω bs sound-ptrs ω bs length bs* = *length ω* + *1*
assumes *build-trees* 𝒢 *ω bs* = *Some fs f* ∈ *set fs t* ∈ *set* (*trees f*)
shows *wf-rule-tree* 𝒢 *t* ∧ *root-tree t* = 𝔖 𝒢 ∧ *yield-tree t* = *ω*
⟨*proof*⟩

corollary *wf-rule-root-yield-tree-build-trees-Earley*$_L$:
assumes *wf-*𝒢 𝒢 *nonempty-derives* 𝒢
assumes *build-trees* 𝒢 *ω* (*Earley*$_L$ 𝒢 *ω*) = *Some fs f* ∈ *set fs t* ∈ *set* (*trees f*)
shows *wf-rule-tree* 𝒢 *t* ∧ *root-tree t* = 𝔖 𝒢 ∧ *yield-tree t* = *ω*
⟨*proof*⟩

theorem *soundness-build-trees-Earley*$_L$:
assumes *wf-*𝒢 𝒢 *is-word* 𝒢 *ω nonempty-derives* 𝒢
assumes *build-trees* 𝒢 *ω* (*Earley*$_L$ 𝒢 *ω*) = *Some fs f* ∈ *set fs t* ∈ *set* (*trees f*)
shows *derives* 𝒢 [𝔖 𝒢] *ω*
⟨*proof*⟩

theorem *termination-build-tree-Earley*$_L$:
assumes *wf-*𝒢 𝒢 *nonempty-derives* 𝒢 *derives* 𝒢 [𝔖 𝒢] *ω*
shows ∃*fs*. *build-trees* 𝒢 *ω* (*Earley*$_L$ 𝒢 *ω*) = *Some fs*
⟨*proof*⟩

end
theory *Examples*
imports *Earley-Parser*

begin

10 Epsilon productions

definition *ε-free* :: *'a cfg* ⇒ *bool* **where**
ε-free $\mathcal{G}$ ⟷ (∀ *r* ∈ *set* (ℜ $\mathcal{G}$). *rule-body r* ≠ [])

lemma *ε-free-impl-non-empty-sentence-deriv*:
ε-free $\mathcal{G}$ ⟹ *a* ≠ [] ⟹ ¬ *Derivation* $\mathcal{G}$ *a D* []
⟨*proof*⟩

lemma *ε-free-impl-non-empty-deriv*:
ε-free $\mathcal{G}$ ⟹ ∀ *N* ∈ *set* (𝔑 $\mathcal{G}$). ¬ *derives* $\mathcal{G}$ [*N*] []
⟨*proof*⟩

lemma *nonempty-deriv-impl-ε-free*:
assumes ∀ *N* ∈ *set* (𝔑 $\mathcal{G}$). ¬ *derives* $\mathcal{G}$ [*N*] [] ∀ (*N*, *α*) ∈ *set* (ℜ $\mathcal{G}$). *N* ∈ *set* (𝔑 $\mathcal{G}$)
shows *ε-free* $\mathcal{G}$
⟨*proof*⟩

lemma *nonempty-deriv-iff-ε-free*:
assumes ∀ (*N*, *α*) ∈ *set* (ℜ $\mathcal{G}$). *N* ∈ *set* (𝔑 $\mathcal{G}$)
shows (∀ *N* ∈ *set* (𝔑 $\mathcal{G}$). ¬ *derives* $\mathcal{G}$ [*N*] []) ⟷ *ε-free* $\mathcal{G}$
⟨*proof*⟩

11 Example 1: Addition

datatype *t1* = *x* | *plus*
datatype *n1* = *S*
datatype *s1* = *Terminal t1* | *Nonterminal n1*

definition *nonterminals1* :: *s1 list* **where**
nonterminals1 = [*Nonterminal S*]

definition *terminals1* :: *s1 list* **where**
terminals1 = [*Terminal x*, *Terminal plus*]

definition *rules1* :: *s1 rule list* **where**
rules1 = [
(*Nonterminal S*, [*Terminal x*]),
(*Nonterminal S*, [*Nonterminal S*, *Terminal plus*, *Nonterminal S*])
]

definition *start-symbol1* :: *s1* **where**
start-symbol1 = *Nonterminal S*

definition *cfg1* :: *s1 cfg* **where**

cfg1 = *CFG nonterminals1 terminals1 rules1 start-symbol1*

definition *inp1* :: *s1 list* **where**
inp1 = [*Terminal x*, *Terminal plus*, *Terminal x*, *Terminal plus*, *Terminal x*]

lemmas *cfg1-defs* = *cfg1-def nonterminals1-def terminals1-def rules1-def start-symbol1-def*

lemma *wf-𝒢1*:
wf-𝒢 cfg1
⟨*proof*⟩

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

lemma *nonempty-derives1*:
nonempty-derives cfg1
⟨*proof*⟩

lemma *correctness1*:
recognizing (*bins* (*Earley*$_L$ *cfg1 inp1*)) *cfg1 inp1* ⟷ *derives cfg1* [𝔖 *cfg1*] *inp1*
⟨*proof*⟩

lemma *wf-tree1*:
assumes *build-tree cfg1 inp1* (*Earley*$_L$ *cfg1 inp1*) = *Some t*
shows *wf-rule-tree cfg1 t* ∧ *root-tree t* = 𝔖 *cfg1* ∧ *yield-tree t* = *inp1*
⟨*proof*⟩

lemma *correctness-tree1*:
(∃ *t*. *build-tree cfg1 inp1* (*Earley*$_L$ *cfg1 inp1*) = *Some t*) ⟷ *derives cfg1* [𝔖 *cfg1*] *inp1*
⟨*proof*⟩

lemma *wf-trees1*:
assumes *build-trees cfg1 inp1* (*Earley*$_L$ *cfg1 inp1*) = *Some fs f* ∈ *set fs t* ∈ *set* (*trees f*)
shows *wf-rule-tree cfg1 t* ∧ *root-tree t* = 𝔖 *cfg1* ∧ *yield-tree t* = *inp1*
⟨*proof*⟩

lemma *soundness-trees1*:
assumes *build-trees cfg1 inp1* (*Earley*$_L$ *cfg1 inp1*) = *Some fs f* ∈ *set fs t* ∈ *set* (*trees f*)
shows *derives cfg1* [𝔖 *cfg1*] *inp1*
⟨*proof*⟩

12 Example 2: Cyclic reduction pointers

datatype *t2* = *x*
datatype *n2* = *A* | *B*

datatype *s2* = *Terminal t2* | *Nonterminal n2*

definition *nonterminals2* :: *s2 list* **where**
nonterminals2 = [*Nonterminal A*, *Nonterminal B*]

definition *terminals2* :: *s2 list* **where**
terminals2 = [*Terminal x*]

definition *rules2* :: *s2 rule list* **where**
rules2 = [
(*Nonterminal B*, [*Nonterminal A*]),
(*Nonterminal A*, [*Nonterminal B*]),
(*Nonterminal A*, [*Terminal x*])
]

definition *start-symbol2* :: *s2* **where**
start-symbol2 = *Nonterminal A*

definition *cfg2* :: *s2 cfg* **where**
cfg2 = *CFG nonterminals2 terminals2 rules2 start-symbol2*

definition *inp2* :: *s2 list* **where**
inp2 = [*Terminal x*]

lemmas *cfg2-defs* = *cfg2-def nonterminals2-def terminals2-def rules2-def start-symbol2-def*

lemma *wf-𝒢2*:
wf-𝒢 cfg2
⟨*proof*⟩

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

lemma *nonempty-derives2*:
nonempty-derives cfg2
⟨*proof*⟩

lemma *correctness2*:
recognizing (*bins* (*Earley*$_L$ *cfg2 inp2*)) *cfg2 inp2* ⟷ *derives cfg2* [𝔖 *cfg2*] *inp2*
⟨*proof*⟩

lemma *wf-tree2*:
assumes *build-tree cfg2 inp2* (*Earley*$_L$ *cfg2 inp2*) = *Some t*
shows *wf-rule-tree cfg2 t* ∧ *root-tree t* = 𝔖 *cfg2* ∧ *yield-tree t* = *inp2*
⟨*proof*⟩

lemma *correctness-tree2*:
(∃ *t*. *build-tree cfg2 inp2* (*Earley*$_L$ *cfg2 inp2*) = *Some t*) ⟷ *derives cfg2* [𝔖

cfg2] *inp2*
⟨*proof*⟩

lemma *wf-trees2*:
assumes *build-trees cfg2 inp2* (*Earley*$_L$ *cfg2 inp2*) = *Some fs f* ∈ *set fs t* ∈ *set* (*trees f*)
shows *wf-rule-tree cfg2 t* ∧ *root-tree t* = 𝔖 *cfg2* ∧ *yield-tree t* = *inp2*
⟨*proof*⟩

lemma *soundness-trees2*:
assumes *build-trees cfg2 inp2* (*Earley*$_L$ *cfg2 inp2*) = *Some fs f* ∈ *set fs t* ∈ *set* (*trees f*)
shows *derives cfg2* [𝔖 *cfg2*] *inp2*
⟨*proof*⟩

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).