Earley

Martin Rau

September 13, 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**
  $natUnion\ f = \bigcup\ \{\ f\ n \mid n.\ True\ \}$

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

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

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

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

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

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

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

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

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

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

lemma *elem-limit-simp*: *x* ∈ *limit f X* = (∃ *n*. *x* ∈ *funpower f n X*)
⟨*proof*⟩

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

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

lemma *limit-elem*: *x* ∈ *funpower f n X* ⟹ *x* ∈ *limit f X*
⟨*proof*⟩

definition *pointbase* :: ('*a set* ⇒ '*b set*) ⇒ '*a set* ⇒ '*b set* **where**
pointbase F I = ⋃ { *F X* | *X*. *finite X* ∧ *X* ⊆ *I* }

definition *pointbased* :: ('*a set* ⇒ '*b set*) ⇒ *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 $'a\ rule = 'a \times 'a\ list$

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

type-synonym $'a\ sentence = 'a\ list$

datatype $'a\ cfg =$
$CFG\ (\mathfrak{N} : 'a\ list)\ (\mathfrak{T} : 'a\ list)\ (\mathfrak{R} : 'a\ rules)\ (\mathfrak{S} : 'a)$

definition *disjunct-symbols* :: $'a\ cfg \Rightarrow bool$ **where**
$disjunct\text{-}symbols\ \mathcal{G} \equiv set\ (\mathfrak{N}\ \mathcal{G}) \cap set\ (\mathfrak{T}\ \mathcal{G}) = \{\}$

definition *valid-startsymbol* :: $'a\ cfg \Rightarrow bool$ **where**
$valid\text{-}startsymbol\ \mathcal{G} \equiv \mathfrak{S}\ \mathcal{G} \in set\ (\mathfrak{N}\ \mathcal{G})$

definition *valid-rules* :: $'a\ cfg \Rightarrow bool$ **where**
$valid\text{-}rules\ \mathcal{G} \equiv \forall (N, \alpha) \in set\ (\mathfrak{R}\ \mathcal{G}).\ N \in set\ (\mathfrak{N}\ \mathcal{G}) \wedge (\forall s \in set\ \alpha.\ s \in set\ (\mathfrak{N}\ \mathcal{G}) \cup set\ (\mathfrak{T}\ \mathcal{G}))$

definition *distinct-rules* :: $'a\ cfg \Rightarrow bool$ **where**
$distinct\text{-}rules\ \mathcal{G} \equiv distinct\ (\mathfrak{R}\ \mathcal{G})$

definition *wf-$\mathcal{G}$* :: $'a\ cfg \Rightarrow bool$ **where**
$wf\text{-}\mathcal{G}\ \mathcal{G} \equiv disjunct\text{-}symbols\ \mathcal{G} \wedge valid\text{-}startsymbol\ \mathcal{G} \wedge valid\text{-}rules\ \mathcal{G} \wedge distinct\text{-}rules\ \mathcal{G}$

lemmas *wf-$\mathcal{G}$-defs = wf-$\mathcal{G}$-def valid-rules-def valid-startsymbol-def disjunct-symbols-def distinct-rules-def*

definition *is-terminal* :: $'a\ cfg \Rightarrow 'a \Rightarrow bool$ **where**
$is\text{-}terminal\ \mathcal{G}\ x \equiv x \in set\ (\mathfrak{T}\ \mathcal{G})$

definition *is-nonterminal* :: $'a\ cfg \Rightarrow 'a \Rightarrow bool$ **where**
$is\text{-}nonterminal\ \mathcal{G}\ x \equiv x \in set\ (\mathfrak{N}\ \mathcal{G})$

definition *is-symbol* :: $'a\ cfg \Rightarrow 'a \Rightarrow bool$ **where**
$is\text{-}symbol\ \mathcal{G}\ x \equiv is\text{-}terminal\ \mathcal{G}\ x \vee is\text{-}nonterminal\ \mathcal{G}\ x$

definition *wf-sentence* :: $'a\ cfg \Rightarrow 'a\ sentence \Rightarrow bool$ **where**
$wf\text{-}sentence\ \mathcal{G}\ \omega \equiv \forall x \in set\ \omega.\ is\text{-}symbol\ \mathcal{G}\ x$

lemma *is-nonterminal-startsymbol*:
$wf\text{-}\mathcal{G}\ \mathcal{G} \Longrightarrow is\text{-}nonterminal\ \mathcal{G}\ (\mathfrak{S}\ \mathcal{G})$

⟨*proof*⟩

definition *is-word* :: *'a cfg* ⇒ *'a sentence* ⇒ *bool* **where**
is-word $\mathcal{G}$ ω ≡ ∀ *x* ∈ *set* ω. *is-terminal* $\mathcal{G}$ *x*

definition *derives1* :: *'a cfg* ⇒ *'a sentence* ⇒ *'a sentence* ⇒ *bool* **where**
derives1 $\mathcal{G}$ *u v* ≡ ∃ *x y N* α.
u = *x* @ [*N*] @ *y* ∧
v = *x* @ α @ *y* ∧
(*N*, α) ∈ *set* ($\mathfrak{R}$ $\mathcal{G}$)

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

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

definition *derives* :: *'a cfg* ⇒ *'a sentence* ⇒ *'a sentence* ⇒ *bool* **where**
derives $\mathcal{G}$ *u v* ≡ ((*u*, *v*) ∈ *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* ⟹ *derives* $\mathcal{G}$ *a b*
⟨*proof*⟩

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

lemma *derives1-eq-derivations1*:
derives1 $\mathcal{G}$ *x y* = ((*x*, *y*) ∈ *derivations1* $\mathcal{G}$)
⟨*proof*⟩

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

⟨*proof*⟩

definition *Derives1* :: *'a cfg* ⇒ *'a 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*: $Derivation\ \mathcal{G}\ a\ (D@E)\ c = (\exists\ b.\ Derivation\ \mathcal{G}\ a\ D\ b \wedge Derivation\ \mathcal{G}\ b\ E\ c)$
⟨*proof*⟩

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

4 Additional derivation lemmas

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

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

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

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

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

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

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

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

lemma *Derivation-concat-split*:

Derivation $\mathcal{G}$ *(a@b) D c* $\Longrightarrow$ $\exists$ *E F a′ b′. Derivation* $\mathcal{G}$ *a E a′* $\wedge$ *Derivation* $\mathcal{G}$ *b F b′* $\wedge$
c = a′ @ b′ $\wedge$ *length E* $\leq$ *length D* $\wedge$ *length F* $\leq$ *length D*
⟨*proof*⟩

lemma *Derivation-𝔖1*:
assumes *Derivation* $\mathcal{G}$ [𝔖 $\mathcal{G}$] *D* ω *is-word* $\mathcal{G}$ ω *wf-*$\mathcal{G}$ $\mathcal{G}$
shows $\exists$ α *E. Derivation* $\mathcal{G}$ α *E* ω $\wedge$ (𝔖 $\mathcal{G}$,α) $\in$ *set* (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 $\leq$ *c* $\Longrightarrow$ *slice* (*Suc b*) *c* (*x # xs*) = *slice b* (*c−1*) *xs*
⟨*proof*⟩

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

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

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

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

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

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

lemma *slice-singleton*:
$b \le length\ xs \Longrightarrow [x] = slice\ a\ b\ xs \Longrightarrow b = a + 1$
⟨*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* ⇒ *'a cfg* ⇒ *'a sentence* ⇒ *bool* **where**
recognizing $I\ \mathcal{G}\ \omega \equiv \exists x \in I.$ *is-finished* $\mathcal{G}\ \omega\ x$

inductive-set *Earley* :: *'a cfg* ⇒ *'a sentence* ⇒ *'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* ⇒ *'a sentence* => *'a item* ⇒ *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* :: $'a\ cfg \Rightarrow 'a\ sentence \Rightarrow 'a\ item \Rightarrow bool$ **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 $\omega!j = a\ j < length\ \omega$ *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* $\mathcal{G}\ [\mathfrak{S}\ \mathcal{G}]\ \omega$

⟨*proof*⟩

6.4 Completeness

definition *partially-completed* :: $nat \Rightarrow {'a}\ cfg \Rightarrow {'a}\ sentence \Rightarrow {'a}\ item\ set \Rightarrow ({'a}\ derivation \Rightarrow bool) \Rightarrow bool$ **where**
partially-completed $k\ \mathcal{G}\ \omega\ E\ P \equiv \forall\, r\ b\ i'\ i\ j\ x\ a\ D.$
$i \leq j \wedge j \leq k \wedge k \leq length\ \omega\ \wedge$
$x = Item\ r\ b\ i'\ i \wedge x \in E \wedge next\text{-}symbol\ x = Some\ a\ \wedge$
$Derivation\ \mathcal{G}\ [a]\ D\ (slice\ i\ j\ \omega) \wedge P\ D \longrightarrow$
$Item\ r\ (b{+}1)\ i'\ j \in E$

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

lemma *partially-completed-Earley-k*:
assumes $wf\text{-}\mathcal{G}\ \mathcal{G}$
shows $partially\text{-}completed\ k\ \mathcal{G}\ \omega\ (Earley\ \mathcal{G}\ \omega)\ (\lambda\text{-}.\ True)$
⟨*proof*⟩

lemma *partially-completed-Earley*:
$wf\text{-}\mathcal{G}\ \mathcal{G} \Longrightarrow partially\text{-}completed\ (length\ \omega)\ \mathcal{G}\ \omega\ (Earley\ \mathcal{G}\ \omega)\ (\lambda\text{-}.\ True)$
⟨*proof*⟩

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

6.5 Correctness

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

6.6 Finiteness

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

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

lemma *finiteness-nonempty*:
  assumes *set* $(\mathfrak{R}\ \mathcal{G}) \neq \{\}$
  shows *finite* $\{\ x \mid x.\ \textit{wf-item}\ \mathcal{G}\ \omega\ x\ \}$
⟨*proof*⟩

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

theorem *finiteness-Earley*:
  finite $(\textit{Earley}\ \mathcal{G}\ \omega)$
  ⟨*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 ≡ $\{\ x\ .\ x \in I \wedge \textit{item-end}\ x = k\ \}$

definition $\textit{Init}_F$ :: *'a cfg* ⇒ *'a item set* **where**
  $\textit{Init}_F\ \mathcal{G} \equiv \{\ \textit{init-item}\ r\ 0 \mid r.\ r \in \textit{set}\ (\mathfrak{R}\ \mathcal{G}) \wedge \textit{fst}\ r = (\mathfrak{S}\ \mathcal{G})\ \}$

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

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

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

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

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

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

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

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

lemma $Predict_F$*-bin-absorb*:

$Predict_F\ k\ \mathcal{G}\ (bin\ I\ k) = Predict_F\ k\ \mathcal{G}\ I$
⟨*proof*⟩

lemma $Scan_F$-*Un*:
$Scan_F\ k\ \omega\ (I \cup J) = Scan_F\ k\ \omega\ I \cup Scan_F\ k\ \omega\ J$
⟨*proof*⟩

lemma $Predict_F$-*Un*:
$Predict_F\ k\ \mathcal{G}\ (I \cup J) = Predict_F\ k\ \mathcal{G}\ I \cup Predict_F\ k\ \mathcal{G}\ J$
⟨*proof*⟩

lemma $Scan_F$-*sub-mono*:
$I \subseteq J \Longrightarrow Scan_F\ k\ \omega\ I \subseteq Scan_F\ k\ \omega\ J$
⟨*proof*⟩

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

lemma $Complete_F$-*sub-mono*:
$I \subseteq J \Longrightarrow Complete_F\ k\ I \subseteq Complete_F\ k\ J$
⟨*proof*⟩

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

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

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

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

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

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

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

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

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

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

7.3 Soundness

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

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

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

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

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

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

lemma *Earley$_F$-bins-sub-Earley*:
shows *Earley$_F$-bins* n $\mathcal{G}$ $\omega \subseteq$ *Earley* $\mathcal{G}$ ω
⟨*proof*⟩

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

theorem *soundness-Earley$_F$*:
assumes *recognizing* (*Earley$_F$* $\mathcal{G}$ ω) $\mathcal{G}$ ω
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 . $x \in I$ ∧ *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* $\mathcal{G}$ ω) $k' \subseteq I$
assumes *base* ω (*Earley* $\mathcal{G}$ ω) $k \subseteq I$
shows *bin* (*Earley* $\mathcal{G}$ ω) $k \subseteq$ *bin* (*Earley$_F$-bin* k $\mathcal{G}$ ω I) k
⟨*proof*⟩

lemma *Earley-base-sub-Earley$_F$-bin*:
assumes *Init$_F$* $\mathcal{G} \subseteq I$
assumes $\forall k' < k.$ *bin* (*Earley* $\mathcal{G}$ ω) $k' \subseteq I$
assumes *base* ω (*Earley* $\mathcal{G}$ ω) $k \subseteq I$
assumes *wf-$\mathcal{G}$* $\mathcal{G}$ *is-word* $\mathcal{G}$ ω
shows *base* ω (*Earley* $\mathcal{G}$ ω) (k+*1*) $\subseteq$ *bin* (*Earley$_F$-bin* k $\mathcal{G}$ ω I) (k+*1*)
⟨*proof*⟩

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

lemma *Earley-sub-Earley$_F$*:

assumes *wf-$\mathcal{G}$ $\mathcal{G}$ is-word $\mathcal{G}$ ω*
shows *Earley* $\mathcal{G}$ $\omega \subseteq$ *Earley*$_F$ $\mathcal{G}$ ω
⟨*proof*⟩

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

7.5 Correctness

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

theorem *correctness-Earley*$_F$:
assumes *wf-*$\mathcal{G}$ $\mathcal{G}$ *is-word* $\mathcal{G}$ ω
shows *recognizing* (*Earley*$_F$ $\mathcal{G}$ ω) $\mathcal{G}$ ω $\longleftrightarrow$ *derives* $\mathcal{G}$ [$\mathfrak{S}$ $\mathcal{G}$] 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 $items :: {}'a\ bin \Rightarrow {}'a\ item\ list$ **where**
$items\ b \equiv map\ item\ b$

definition $pointers :: {}'a\ bin \Rightarrow pointer\ list$ **where**
$pointers\ b \equiv map\ pointer\ b$

definition $bins\text{-}eq\text{-}items :: {}'a\ bins \Rightarrow {}'a\ bins \Rightarrow bool$ **where**
$bins\text{-}eq\text{-}items\ bs0\ bs1 \equiv map\ items\ bs0 = map\ items\ bs1$

definition $bins :: {}'a\ bins \Rightarrow {}'a\ item\ set$ **where**
$bins\ bs \equiv \bigcup\ \{\ set\ (items\ (bs!k))\ |\ k.\ k < length\ bs\ \}$

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

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

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

definition $wf\text{-}bin :: {}'a\ cfg \Rightarrow {}'a\ sentence \Rightarrow nat \Rightarrow {}'a\ bin \Rightarrow bool$ **where**
$wf\text{-}bin\ \mathcal{G}\ \omega\ k\ b \equiv distinct\ (items\ b) \land wf\text{-}bin\text{-}items\ \mathcal{G}\ \omega\ k\ (items\ b)$

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

definition $nonempty\text{-}derives :: {}'a\ cfg \Rightarrow bool$ **where**
$nonempty\text{-}derives\ \mathcal{G} \equiv \forall N.\ N \in set\ (\mathfrak{N}\ \mathcal{G}) \longrightarrow \neg\ derives\ \mathcal{G}\ [N]\ []$

definition $Init_L :: {}'a\ cfg \Rightarrow {}'a\ sentence \Rightarrow {}'a\ bins$ **where**
$Init_L\ \mathcal{G}\ \omega \equiv$
$\quad let\ rs = filter\ (\lambda r.\ rule\text{-}head\ r = \mathfrak{S}\ \mathcal{G})\ (\mathfrak{R}\ \mathcal{G})\ in$
$\quad let\ b0 = map\ (\lambda r.\ (Entry\ (init\text{-}item\ r\ 0)\ Null))\ rs\ in$
$\quad let\ bs = replicate\ (length\ \omega + 1)\ ([])\ in$
$\quad bs[0 := b0]$

definition $Scan_L :: nat \Rightarrow {}'a\ sentence \Rightarrow {}'a \Rightarrow {}'a\ item \Rightarrow nat \Rightarrow {}'a\ entry\ list$
where
$Scan_L\ k\ \omega\ a\ x\ pre \equiv$
$\quad if\ \omega!k = a\ then$
$\quad\quad let\ x' = inc\text{-}item\ x\ (k+1)\ in$
$\quad\quad [Entry\ x'\ (Pre\ pre)]$
$\quad else\ []$

definition $Predict_L :: nat \Rightarrow {}'a\ cfg \Rightarrow {}'a \Rightarrow {}'a\ entry\ list$ **where**

```
  PredictL k 𝒢 X ≡
    let rs = filter (λr. rule-head r = X) (ℜ 𝒢) in
    map (λr. (Entry (init-item r k) Null)) rs

definition CompleteL :: nat ⇒ 'a item ⇒ 'a bins ⇒ nat ⇒ 'a entry list where
  CompleteL 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) EarleyL-bin' :: nat ⇒ 'a cfg ⇒ 'a sentence ⇒ 'a bins
⇒ nat ⇒ 'a bins where
  EarleyL-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) (ScanL k ω a x i)
              else bs
            else bins-upd bs k (PredictL k 𝒢 a)
        | None ⇒ bins-upd bs k (CompleteL k x bs i)
      in EarleyL-bin' k 𝒢 ω bs' (i+1))

declare EarleyL-bin'.simps[code]

definition EarleyL-bin :: nat ⇒ 'a cfg ⇒ 'a sentence ⇒ 'a bins ⇒ 'a bins where
  EarleyL-bin k 𝒢 ω bs ≡ EarleyL-bin' k 𝒢 ω bs 0
```

fun $Earley_L$*-bins* :: *nat* ⇒ *'a cfg* ⇒ *'a sentence* ⇒ *'a bins* **where**
$Earley_L$*-bins 0* $\mathcal{G}$ ω = $Earley_L$*-bin 0* $\mathcal{G}$ ω ($Init_L$ $\mathcal{G}$ ω)
| $Earley_L$*-bins* (*Suc n*) $\mathcal{G}$ ω = $Earley_L$*-bin* (*Suc n*) $\mathcal{G}$ ω ($Earley_L$*-bins n* $\mathcal{G}$ ω)

definition $Earley_L$ :: *'a cfg* ⇒ *'a sentence* ⇒ *'a bins* **where**
$Earley_L$ $\mathcal{G}$ ω ≡ $Earley_L$*-bins* (*length* ω) $\mathcal{G}$ ω

8.3 Bin lemmas

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

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

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

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

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

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

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

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

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

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

lemma *set-items-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 < *length* (*bs* ! *k*) ⟹ *bins-upto bs k* (*n*+*1*) = *bins-upto bs k n* ∪ { *items* (*bs* ! *k*) ! *n* }
⟨*proof*⟩

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

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

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

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

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

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

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

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

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

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

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

shows *bins-eq-items* (*bins-upd bs k es*) *bs*
⟨*proof*⟩

lemma *bins-eq-items-imp-eq-bins*:
bins-eq-items bs bs′ $\Longrightarrow$ *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-$\mathcal{G}$ $\mathcal{G}$ $\Longrightarrow$ *distinct* (*items* ($Predict_L$ k $\mathcal{G}$ X))
⟨*proof*⟩

lemma *inj-on-inc-item*:
$\forall x \in A$. *item-end* $x = l$ $\Longrightarrow$ *inj-on* (λx. *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* ∈ *set* (*items* (*bs* ! *k*))
assumes *k* < *length* ω *next-symbol x* ≠ *None y* = *inc-item x* (*k+1*)
shows *wf-item* $\mathcal{G}$ ω *y* ∧ *item-end y* = *k+1*
⟨*proof*⟩

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

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

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

lemma *Ex-wf-bins*:
∃ *n bs* ω $\mathcal{G}$. *n* ≤ *length* ω ∧ *length bs* = *Suc* (*length* ω) ∧ *wf-*$\mathcal{G}$ $\mathcal{G}$ ∧ *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 $(k, \mathcal{G}, \omega, bs) \in$ *wf-earley-input*
shows $k \leq$ *length* $\omega \wedge k <$ *length bs* $\wedge$ *length bs* $=$ *length* $\omega + 1 \wedge$ *wf-*$\mathcal{G}$ $\mathcal{G}$ $\wedge$ *wf-bins* $\mathcal{G}$ ω *bs*
⟨*proof*⟩

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

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

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

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

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

lemma $Earley_L$*-bin'-simps*[*simp*]:
$i \geq$ *length* (*items* (*bs* ! *k*)) $\Longrightarrow$ $Earley_L$*-bin'* k $\mathcal{G}$ ω *bs* $i =$ *bs*
$\neg$ $i \geq$ *length* (*items* (*bs* ! *k*)) $\Longrightarrow$ $x =$ *items* (*bs*!*k*) ! i $\Longrightarrow$ *next-symbol* $x =$ *None* $\Longrightarrow$
$Earley_L$*-bin'* k $\mathcal{G}$ ω *bs* $i =$ $Earley_L$*-bin'* k $\mathcal{G}$ ω (*bins-upd bs k* ($Complete_L$ *k x bs i*)) $(i+1)$
$\neg$ $i \geq$ *length* (*items* (*bs* ! *k*)) $\Longrightarrow$ $x =$ *items* (*bs*!*k*) ! i $\Longrightarrow$ *next-symbol* $x =$ *Some* $a \Longrightarrow$
is-terminal $\mathcal{G}$ $a \Longrightarrow k <$ *length* $\omega \Longrightarrow$ $Earley_L$*-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*)) $(i+1)$
$\neg$ $i \geq$ *length* (*items* (*bs* ! *k*)) $\Longrightarrow$ $x =$ *items* (*bs*!*k*) ! i $\Longrightarrow$ *next-symbol* $x =$ *Some* $a \Longrightarrow$
is-terminal $\mathcal{G}$ $a \Longrightarrow \neg$ $k <$ *length* $\omega \Longrightarrow$ $Earley_L$*-bin'* k $\mathcal{G}$ ω *bs* $i =$ $Earley_L$*-bin'* k $\mathcal{G}$ ω *bs* $(i+1)$
$\neg$ $i \geq$ *length* (*items* (*bs* ! *k*)) $\Longrightarrow$ $x =$ *items* (*bs*!*k*) ! i $\Longrightarrow$ *next-symbol* $x =$ *Some*

$a \Longrightarrow$
  $\neg$ *is-terminal* $\mathcal{G}$ $a \Longrightarrow$ *Earley*$_L$*-bin'* k $\mathcal{G}$ ω *bs* i = *Earley*$_L$*-bin'* k $\mathcal{G}$ ω (*bins-upd* *bs* k (*Predict*$_L$ k $\mathcal{G}$ a)) (i+1)
  ⟨*proof*⟩

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

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

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

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

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

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

shows *wf-bins* $\mathcal{G}$ ω (*Earley*$_L$*-bin′* k $\mathcal{G}$ ω *bs* i)
⟨*proof*⟩

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

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

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

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

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

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

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

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

lemma *length-Earley*$_L$*-bins*[*simp*]:

assumes $k \leq$ *length* ω *wf-*$\mathcal{G}$ $\mathcal{G}$
shows *length* (*Earley*$_L$*-bins* k $\mathcal{G}$ ω) = *length* (*Init*$_L$ $\mathcal{G}$ ω)
⟨*proof*⟩

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

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

8.5 Soundness

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

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

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

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

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

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

$\mathcal{G}\ \omega\ x$
shows $\forall\, x \in set\ (items\ (Predict_L\ k\ \mathcal{G}\ X)).\ sound\text{-}item\ \mathcal{G}\ \omega\ x$
$\langle proof \rangle$

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

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

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

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

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

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

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

theorem *soundness-Earley*$_L$:
assumes $wf\text{-}\mathcal{G}\ \mathcal{G}\ recognizing\ (bins\ (Earley_L\ \mathcal{G}\ \omega))\ \mathcal{G}\ \omega$
shows $derives\ \mathcal{G}\ [\mathfrak{S}\ \mathcal{G}]\ \omega$
$\langle proof \rangle$

8.6 Completeness

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

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

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

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

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

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

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

lemma $Complete_L$*-eq-item-origin*:
  $bs\ !\ item\text{-}origin\ y = bs'\ !\ item\text{-}origin\ y \Longrightarrow Complete_L\ k\ y\ bs\ red = Complete_L\ k\ y\ bs'\ red$
  ⟨*proof*⟩

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

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

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

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

lemma *$Earley_L$-bin′-idem*:
assumes $(k, \mathcal{G}, \omega, bs) \in$ *wf-earley-input*
assumes $i \le j\ \forall x \in$ *bins bs. sound-item* $\mathcal{G}\ \omega\ x$ *nonempty-derives* $\mathcal{G}$
shows *bins* $($*$Earley_L$-bin′* $k\ \mathcal{G}\ \omega$ $($*$Earley_L$-bin′* $k\ \mathcal{G}\ \omega\ bs\ i)\ j) =$ *bins* $($*$Earley_L$-bin′* $k\ \mathcal{G}\ \omega\ bs\ i)$
⟨*proof*⟩

lemma *$Earley_L$-bin-idem*:
assumes $(k, \mathcal{G}, \omega, bs) \in$ *wf-earley-input*
assumes $\forall x \in$ *bins bs. sound-item* $\mathcal{G}\ \omega\ x$ *nonempty-derives* $\mathcal{G}$
shows *bins* $($*$Earley_L$-bin* $k\ \mathcal{G}\ \omega$ $($*$Earley_L$-bin* $k\ \mathcal{G}\ \omega\ bs)) =$ *bins* $($*$Earley_L$-bin* $k\ \mathcal{G}\ \omega\ bs)$
⟨*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* $\mathcal{G}$ ω *nonempty-derives* $\mathcal{G}$
shows *funpower* (*Earley*$_F$*-bin-step* *k* $\mathcal{G}$ ω) *n* (*bins bs*) $\subseteq$ *bins* (*Earley*$_L$*-bin* *k* $\mathcal{G}$ ω *bs*)
⟨*proof*⟩

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

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

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

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

8.7 Correctness

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

theorem *correctness-Earley*$_L$:
assumes *wf-*$\mathcal{G}$ $\mathcal{G}$ *is-word* $\mathcal{G}$ ω *nonempty-derives* $\mathcal{G}$
shows *recognizing* (*bins* (*Earley*$_L$ $\mathcal{G}$ ω)) $\mathcal{G}$ ω $\longleftrightarrow$ *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 $item\ e = item\ (es!i)$ $pointer\ e = PreRed\ p\ ps$ $\nexists p\ ps.\ pointer\ (es!i) = PreRed\ p\ ps$
shows $bin\text{-}upd\ e\ es = es$
⟨*proof*⟩

lemma *bin-upd-prered-upd*:
assumes $distinct\ (items\ es)$ $i < length\ es$
assumes $item\ e = item\ (es!i)$ $pointer\ e = PreRed\ p\ rs$ $pointer\ (es!i) = PreRed\ p'\ rs'$ $bin\text{-}upd\ e\ es = es'$
shows $pointer\ (es'!i) = PreRed\ p'\ (p\#rs@rs') \wedge (\forall j < length\ es'.\ i{\neq}j \longrightarrow es'!j = es!j) \wedge length\ (bin\text{-}upd\ e\ es) = length\ es$
⟨*proof*⟩

lemma *sound-ptrs-bin-upd*:
assumes $sound\text{-}ptrs\ \omega\ bs$ $k < length\ bs$ $es = bs!k$ $distinct\ (items\ es)$
assumes $sound\text{-}null\text{-}ptr\ e$ $sound\text{-}pre\text{-}ptr\ \omega\ bs\ k\ e$ $sound\text{-}prered\text{-}ptr\ bs\ k\ e$
shows $sound\text{-}ptrs\ \omega\ (bs[k := bin\text{-}upd\ e\ es])$
⟨*proof*⟩

lemma *mono-red-ptr-bin-upd*:
assumes $mono\text{-}red\text{-}ptr\ bs$ $k < length\ bs$ $es = bs!k$ $distinct\ (items\ es)$
assumes $\forall k'\ pre\ red\ ps.\ pointer\ e = PreRed\ (k',\ pre,\ red)\ ps \longrightarrow red < length\ es$
shows $mono\text{-}red\text{-}ptr\ (bs[k := bin\text{-}upd\ e\ es])$
⟨*proof*⟩

lemma *sound-mono-ptrs-bin-upds*:
assumes $sound\text{-}ptrs\ \omega\ bs$ $mono\text{-}red\text{-}ptr\ bs$ $k < length\ bs$ $b = bs!k$ $distinct\ (items\ b)$ $distinct\ (items\ es)$
assumes $\forall e \in set\ es.\ sound\text{-}null\text{-}ptr\ e \wedge sound\text{-}pre\text{-}ptr\ \omega\ bs\ k\ e \wedge sound\text{-}prered\text{-}ptr\ bs\ k\ e$
assumes $\forall e \in set\ es.\ \forall k'\ pre\ red\ ps.\ pointer\ e = PreRed\ (k',\ pre,\ red)\ ps \longrightarrow red < length\ b$
shows $sound\text{-}ptrs\ \omega\ (bs[k := bin\text{-}upds\ es\ b]) \wedge mono\text{-}red\text{-}ptr\ (bs[k := bin\text{-}upds\ es\ b])$
⟨*proof*⟩

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

lemma *sound-mono-ptrs-Earley$_L$-bin*:

assumes $(k, \mathcal{G}, \omega, bs) \in$ *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*) ⟷ (
($\exists r \in$ *set* ($\mathfrak{R}$ $\mathcal{G}$). *N* = *rule-head r* $\wedge$ *map root-tree ts* = *rule-body r*) $\wedge$
($\forall t \in$ *set ts. wf-rule-tree* $\mathcal{G}$ *t*))

fun *wf-item-tree* :: *'a cfg* ⇒ *'a item* ⇒ *'a tree* ⇒ *bool* **where**
wf-item-tree $\mathcal{G}$ - (*Leaf a*) ⟷ *True*
| *wf-item-tree* $\mathcal{G}$ *x* (*Branch N ts*) ⟷ (

N = *item-rule-head x* ∧ *map root-tree ts* = *take* (*item-dot x*) (*item-rule-body x*) ∧
(∀ *t* ∈ *set ts*. *wf-rule-tree* $\mathcal{G}$ *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* (λ*fs*. *concat* (*map* (λ*f*. *trees f*) *fs*)) *fss*) *in*
map (λ*ts*. *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* <− *combinations yss*]
⟨*proof*⟩

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

lemma *trees-append-singleton*:
trees (*FBranch N* (*xss* @ [*ys*])) = (
let xtss = (*map* (λ*xs*. *concat* (*map* (λ*f*. *trees f*) *xs*)) *xss*) *in*
let ytss = [*concat* (*map trees ys*)] *in*
map (λ*ts*. *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* (λ*xs*. *concat* (*map* (λ*f*. *trees f*) *xs*)) *xss*) *in*
map (λ*ts*. *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 𝒢 ω bs = (
let k = length bs − 1 in (
case filter-with-index (λx. is-finished 𝒢 ω x) (items (bs!k)) of
[] ⇒ None
| *(-, i)#- ⇒ 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 ⟹
build-tree′ bs ω k i = None
e = bs!k!i ⟹ pointer e = PreRed (k′, pre, red) reds ⟹ build-tree′ bs ω k′ pre = Some (Branch N ts) ⟹

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

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

fun $build\text{-}tree'\text{-}measure :: ('a\ bins \times 'a\ sentence \times nat \times nat) \Rightarrow nat$ **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 $\bigwedge bs\ \omega\ k\ i.$
$(\bigwedge e\ pre.\ e = bs!k!i \Longrightarrow pointer\ e = Pre\ pre \Longrightarrow P\ bs\ \omega\ (k-1)\ pre) \Longrightarrow$
$(\bigwedge e\ k'\ pre\ red\ ps.\ e = bs!k!i \Longrightarrow pointer\ e = PreRed\ (k', pre, red)\ ps \Longrightarrow P\ bs\ \omega\ k'\ pre) \Longrightarrow$
$(\bigwedge e\ k'\ pre\ red\ ps.\ e = bs!k!i \Longrightarrow pointer\ e = PreRed\ (k', pre, red)\ ps \Longrightarrow P\ bs\ \omega\ k\ red) \Longrightarrow$
$P\ bs\ \omega\ k\ i$

shows *P bs* ω *k i*
⟨*proof*⟩

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

lemma *wf-item-tree-build-tree′*:
assumes $(bs, \omega, k, i) \in$ *wf-tree-input*
assumes *wf-bins* $\mathcal{G}$ 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 $(bs, \omega, k, i) \in$ *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* $\wedge$ *root-tree t* = $\mathfrak{S}$ $\mathcal{G}$ $\wedge$ *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* $\wedge$ *root-tree t* = $\mathfrak{S}$ $\mathcal{G}$ $\wedge$ *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 ($\exists$ *t. build-tree* $\mathcal{G}$ ω (*Earley*$_L$ $\mathcal{G}$ ω) = *Some t*) $\longleftrightarrow$ *derives* $\mathcal{G}$ [$\mathfrak{S}$ $\mathcal{G}$] ω (**is** *?L* $\longleftrightarrow$ *?R*)
⟨*proof*⟩

9.5 those, map, map option lemmas

lemma *those-map-exists*:
Some ys = *those* (*map f xs*) $\Longrightarrow$ $y \in$ *set ys* $\Longrightarrow$ $\exists x.\ x \in$ *set xs* $\wedge$ *Some y* $\in$ *set* (*map f xs*)
⟨*proof*⟩

lemma *those-Some*:
$(\forall x \in \mathit{set}\ \mathit{xs}.\ \exists a.\ x = \mathit{Some}\ a) \longleftrightarrow (\exists \mathit{ys}.\ \mathit{those}\ \mathit{xs} = \mathit{Some}\ \mathit{ys})$
⟨*proof*⟩

lemma *those-Some-P*:
assumes $\forall x \in \mathit{set}\ \mathit{xs}.\ \exists \mathit{ys}.\ x = \mathit{Some}\ \mathit{ys} \wedge (\forall y \in \mathit{set}\ \mathit{ys}.\ P\ y)$
shows $\exists \mathit{yss}.\ \mathit{those}\ \mathit{xs} = \mathit{Some}\ \mathit{yss} \wedge (\forall \mathit{ys} \in \mathit{set}\ \mathit{yss}.\ \forall y \in \mathit{set}\ \mathit{ys}.\ P\ y)$
⟨*proof*⟩

lemma *map-Some-P*:
assumes $z \in \mathit{set}\ (\mathit{map}\ f\ \mathit{xs})$
assumes $\forall x \in \mathit{set}\ \mathit{xs}.\ \exists \mathit{ys}.\ f\ x = \mathit{Some}\ \mathit{ys} \wedge (\forall y \in \mathit{set}\ \mathit{ys}.\ P\ y)$
shows $\exists \mathit{ys}.\ z = \mathit{Some}\ \mathit{ys} \wedge (\forall y \in \mathit{set}\ \mathit{ys}.\ P\ y)$
⟨*proof*⟩

lemma *those-map-FBranch-only*:
assumes $g = (\lambda f.\ \mathit{case}\ f\ \mathit{of}\ \mathit{FBranch}\ N\ \mathit{fss} \Rightarrow \mathit{Some}\ (\mathit{FBranch}\ N\ (\mathit{fss}\ @\ [[\mathit{FLeaf}\ (\omega!(k-1))]]))\ |\ \text{-} \Rightarrow \mathit{None})$
assumes $\mathit{Some}\ \mathit{fs} = \mathit{those}\ (\mathit{map}\ g\ \mathit{pres})\ f \in \mathit{set}\ \mathit{fs}$
assumes $\forall f \in \mathit{set}\ \mathit{pres}.\ \exists N\ \mathit{fss}.\ f = \mathit{FBranch}\ N\ \mathit{fss}$
shows $\exists \mathit{f\text{-}pre}\ N\ \mathit{fss}.\ f = \mathit{FBranch}\ N\ (\mathit{fss}\ @\ [[\mathit{FLeaf}\ (\omega!(k-1))]]) \wedge \mathit{f\text{-}pre} = \mathit{FBranch}\ N\ \mathit{fss} \wedge \mathit{f\text{-}pre} \in \mathit{set}\ \mathit{pres}$
⟨*proof*⟩

lemma *those-map-Some-concat-exists*:
assumes $y \in \mathit{set}\ (\mathit{concat}\ \mathit{ys})$
assumes $\mathit{Some}\ \mathit{ys} = \mathit{those}\ (\mathit{map}\ f\ \mathit{xs})$
shows $\exists \mathit{ys}\ x.\ \mathit{Some}\ \mathit{ys} = f\ x \wedge y \in \mathit{set}\ \mathit{ys} \wedge x \in \mathit{set}\ \mathit{xs}$
⟨*proof*⟩

lemma *map-option-concat-those-map-exists*:
assumes $\mathit{Some}\ \mathit{fs} = \mathit{map\text{-}option}\ \mathit{concat}\ (\mathit{those}\ (\mathit{map}\ F\ \mathit{xs}))$
assumes $f \in \mathit{set}\ \mathit{fs}$
shows $\exists \mathit{fss}\ \mathit{fs}'.\ \mathit{Some}\ \mathit{fss} = \mathit{those}\ (\mathit{map}\ F\ \mathit{xs}) \wedge \mathit{fs}' \in \mathit{set}\ \mathit{fss} \wedge f \in \mathit{set}\ \mathit{fs}'$
⟨*proof*⟩

lemma [*partial-function-mono*]:
monotone option.le-fun option-ord
$(\lambda f.\ \mathit{map\text{-}option}\ \mathit{concat}\ (\mathit{those}\ (\mathit{map}\ (\lambda((k',\ \mathit{pre}),\ \mathit{reds}).$
$f\ ((((r,\ s),\ k'),\ \mathit{pre}),\ \{\mathit{pre}\}) \gg=$
$(\lambda \mathit{pres}.\ \mathit{those}\ (\mathit{map}\ (\lambda \mathit{red}.\ f\ ((((r,\ s),\ t),\ \mathit{red}),\ b \cup \{\mathit{red}\}))\ \mathit{reds}) \gg=$
$(\lambda \mathit{rss}.\ \mathit{those}\ (\mathit{map}\ (\lambda f.\ \mathit{case}\ f\ \mathit{of}\ \mathit{FBranch}\ N\ \mathit{fss} \Rightarrow \mathit{Some}\ (\mathit{FBranch}\ N\ (\mathit{fss}\ @\ [\mathit{concat}\ \mathit{rss}]))\ |\ \text{-} \Rightarrow \mathit{None})\ \mathit{pres}))))$
$\mathit{xs})))$
⟨*proof*⟩

9.6 Parse trees

fun *insert-group* :: $('a \Rightarrow 'k) \Rightarrow ('a \Rightarrow 'v) \Rightarrow 'a \Rightarrow ('k \times 'v\ list)\ list \Rightarrow ('k \times 'v\ list)\ list$ **where**
  $insert\text{-}group\ K\ V\ a\ [] = [(K\ a,\ [V\ a])]$
| $insert\text{-}group\ K\ V\ a\ ((k,\ vs)\#xs) = ($
    $if\ K\ a = k\ then\ (k,\ V\ a\ \#\ vs)\ \#\ xs$
    $else\ (k,\ vs)\ \#\ insert\text{-}group\ K\ V\ a\ xs$
)

fun *group-by* :: $('a \Rightarrow 'k) \Rightarrow ('a \Rightarrow 'v) \Rightarrow 'a\ list \Rightarrow ('k \times 'v\ list)\ list$ **where**
  $group\text{-}by\ K\ V\ [] = []$
| $group\text{-}by\ K\ V\ (x\#xs) = insert\text{-}group\ K\ V\ x\ (group\text{-}by\ K\ V\ xs)$

lemma *insert-group-cases*:
  assumes $(k,\ vs) \in set\ (insert\text{-}group\ K\ V\ a\ xs)$
  shows $(k = K\ a \wedge vs = [V\ a]) \vee (k,\ vs) \in set\ xs \vee (\exists (k',\ vs') \in set\ xs.\ k' = k \wedge k = K\ a \wedge vs = V\ a\ \#\ vs')$
  ⟨*proof*⟩

lemma *group-by-exists-kv*:
  $(k,\ vs) \in set\ (group\text{-}by\ K\ V\ xs) \Longrightarrow \exists x \in set\ xs.\ k = K\ x \wedge (\exists v \in set\ vs.\ v = V\ x)$
  ⟨*proof*⟩

lemma *group-by-forall-v-exists-k*:
  $(k,\ vs) \in set\ (group\text{-}by\ K\ V\ xs) \Longrightarrow v \in set\ vs \Longrightarrow \exists x \in set\ xs.\ k = K\ x \wedge v = V\ x$
⟨*proof*⟩

partial-function (*option*) *build-trees'* :: $'a\ bins \Rightarrow 'a\ sentence \Rightarrow nat \Rightarrow nat \Rightarrow nat\ set \Rightarrow 'a\ forest\ list\ option$ **where**
  $build\text{-}trees'\ bs\ \omega\ k\ i\ I = ($
    $let\ e = bs!k!i\ in\ ($
    $case\ pointer\ e\ of$
      $Null \Rightarrow Some\ ([FBranch\ (item\text{-}rule\text{-}head\ (item\ e))\ []])$ — start building sub-trees
    | $Pre\ pre \Rightarrow ($ — add sub-trees starting from terminal
        $do\ \{$
          $pres \leftarrow build\text{-}trees'\ bs\ \omega\ (k-1)\ pre\ \{pre\};$
          $those\ (map\ (\lambda f.$
            $case\ f\ of$
              $FBranch\ N\ fss \Rightarrow Some\ (FBranch\ N\ (fss\ @\ [[FLeaf\ (\omega!(k-1))]]))$
            | $- \Rightarrow None$ — impossible case
          $)\ pres)$
        $\})$
    | $PreRed\ p\ ps \Rightarrow ($ — add sub-trees starting from non-terminal
        $let\ ps' = filter\ (\lambda(k',\ pre,\ red).\ red \notin I)\ (p\#ps)\ in$
        $let\ gs = group\text{-}by\ (\lambda(k',\ pre,\ red).\ (k',\ pre))\ (\lambda(k',\ pre,\ red).\ red)\ ps'\ in$
        $map\text{-}option\ concat\ (those\ (map\ (\lambda((k',\ pre),\ reds).$

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

lemma *wf-trees-input-prered-pre*:
  assumes $(bs, \omega, k, i, I) \in$ *wf-trees-input*
  assumes $e = bs!k!i$ *pointer* $e =$ *PreRed p ps*
  assumes $ps' =$ *filter* $(\lambda(k', pre, red).\ red \notin I)\ (p\#ps)$
  assumes $gs =$ *group-by* $(\lambda(k', pre, red).\ (k', pre))\ (\lambda(k', pre, red).\ red)\ ps'$
  assumes $((k', pre), reds) \in$ *set gs*
  shows $(bs, \omega, k', pre, \{pre\}) \in$ *wf-trees-input*
⟨*proof*⟩

lemma *wf-trees-input-prered-red*:
  assumes $(bs, \omega, k, i, I) \in$ *wf-trees-input*
  assumes $e = bs!k!i$ *pointer* $e =$ *PreRed p ps*
  assumes $ps' =$ *filter* $(\lambda(k', pre, red).\ red \notin I)\ (p\#ps)$
  assumes $gs =$ *group-by* $(\lambda(k', pre, red).\ (k', pre))\ (\lambda(k', pre, red).\ red)\ ps'$
  assumes $((k', pre), reds) \in$ *set gs* $red \in$ *set reds*
  shows $(bs, \omega, k, red, I \cup \{red\}) \in$ *wf-trees-input*
⟨*proof*⟩

lemma *build-trees′-induct*:
  assumes $(bs, \omega, k, i, I) \in$ *wf-trees-input*
  assumes $\bigwedge bs\ \omega\ k\ i\ I.$
    $(\bigwedge e\ pre.\ e = bs!k!i \Longrightarrow pointer\ e = Pre\ pre \Longrightarrow P\ bs\ \omega\ (k-1)\ pre\ \{pre\}) \Longrightarrow$
    $(\bigwedge e\ p\ ps\ ps'\ gs\ k'\ pre\ reds.\ e = bs!k!i \Longrightarrow pointer\ e = PreRed\ p\ ps \Longrightarrow$
      $ps' = filter\ (\lambda(k', pre, red).\ red \notin I)\ (p\#ps) \Longrightarrow$
      $gs = group\text{-}by\ (\lambda(k', pre, red).\ (k', pre))\ (\lambda(k', pre, red).\ red)\ ps' \Longrightarrow$
      $((k', pre), reds) \in set\ gs \Longrightarrow P\ bs\ \omega\ k'\ pre\ \{pre\}) \Longrightarrow$
    $(\bigwedge e\ p\ ps\ ps'\ gs\ k'\ pre\ red\ reds\ reds'.\ e = bs!k!i \Longrightarrow pointer\ e = PreRed\ p\ ps$
$\Longrightarrow$
      $ps' = filter\ (\lambda(k', pre, red).\ red \notin I)\ (p\#ps) \Longrightarrow$
      $gs = group\text{-}by\ (\lambda(k', pre, red).\ (k', pre))\ (\lambda(k', pre, red).\ red)\ ps' \Longrightarrow$
      $((k', pre), reds) \in set\ gs \Longrightarrow red \in set\ reds \Longrightarrow P\ bs\ \omega\ k\ red\ (I \cup \{red\})) \Longrightarrow$
    $P\ bs\ \omega\ k\ i\ I$
  shows $P\ bs\ \omega\ k\ i\ I$
  ⟨*proof*⟩

lemma *build-trees′-termination*:
  assumes $(bs, \omega, k, i, I) \in$ *wf-trees-input*
  shows $\exists fs.\ build\text{-}trees'\ bs\ \omega\ k\ i\ I = Some\ fs \wedge (\forall f \in set\ fs.\ \exists N\ fss.\ f = FBranch$
$N\ fss)$
⟨*proof*⟩

lemma *wf-item-tree-build-trees′*:

assumes $(bs, \omega, k, i, I) \in$ *wf-trees-input*
assumes *wf-bins* $\mathcal{G}$ ω *bs*
assumes $k <$ *length bs* $i <$ *length* $(bs!k)$
assumes *build-trees'* $bs\ \omega\ k\ i\ I =$ *Some fs*
assumes $f \in$ *set fs*
assumes $t \in$ *set* (*trees* f)
shows *wf-item-tree* $\mathcal{G}$ (*item* $(bs!k!i)$) t
⟨*proof*⟩

lemma *wf-yield-tree-build-trees'*:
assumes $(bs, \omega, k, i, I) \in$ *wf-trees-input*
assumes *wf-bins* $\mathcal{G}$ ω *bs*
assumes $k <$ *length bs* $i <$ *length* $(bs!k)$ $k \leq$ *length* ω
assumes *build-trees'* $bs\ \omega\ k\ i\ I =$ *Some fs*
assumes $f \in$ *set fs*
assumes $t \in$ *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* $\mathcal{G}$ ω *bs sound-ptrs* ω *bs length bs* $=$ *length* $\omega + 1$
assumes *build-trees* $\mathcal{G}$ ω *bs* $=$ *Some fs* $f \in$ *set fs* $t \in$ *set* (*trees* f)
shows *wf-rule-tree* $\mathcal{G}$ $t \wedge$ *root-tree* $t = \mathfrak{S}\ \mathcal{G} \wedge$ *yield-tree* $t = \omega$
⟨*proof*⟩

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

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

theorem *termination-build-tree-Earley*$_L$:
assumes *wf-*$\mathcal{G}$ $\mathcal{G}$ *nonempty-derives* $\mathcal{G}$ *derives* $\mathcal{G}$ $[\mathfrak{S}\ \mathcal{G}]$ ω
shows $\exists fs.$ *build-trees* $\mathcal{G}$ ω (*Earley*$_L$ $\mathcal{G}$ 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* ($\mathfrak{R}$ $\mathcal{G}$). *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* ($\mathfrak{N}$ $\mathcal{G}$). ¬ *derives* $\mathcal{G}$ [*N*] []
⟨*proof*⟩

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

lemma *nonempty-deriv-iff-ε-free*:
assumes ∀ (*N*, *α*) ∈ *set* ($\mathfrak{R}$ $\mathcal{G}$). *N* ∈ *set* ($\mathfrak{N}$ $\mathcal{G}$)
shows (∀ *N* ∈ *set* ($\mathfrak{N}$ $\mathcal{G}$). ¬ *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-$\mathcal{G}$2*:
wf-$\mathcal{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* $\longleftrightarrow$ *derives cfg2* [$\mathfrak{S}$ *cfg2*] *inp2*
⟨*proof*⟩

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

lemma *correctness-tree2*:
($\exists$ *t*. *build-tree cfg2 inp2* (*Earley*$_L$ *cfg2 inp2*) = *Some t*) $\longleftrightarrow$ *derives cfg2* [$\mathfrak{S}$ *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).