Earley

Martin Rau

March 17, 2025

Abstract

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

Contents

theory *Limit*
imports
Main
begin

1 Slightly adjusted content from AFP/LocalLexing

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

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

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

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

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

lemma *subset-setmonotone*: *setmonotone f* ⟹ *X* ⊆ *f X*
by (*simp add*: *setmonotone-def*)

lemma[*simp*]: *funpower id n* = *id*
by (*rule ext*, *induct n*, *simp-all*)

lemma[*simp*]: *limit id* = *id*
by (*rule ext*, *auto simp add*: *limit-def natUnion-def*)

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*
by (*auto simp add*: *natUnion-def*)

lemma *funpower-upperbound*:
(⋀ *I*. *I* ⊆ *G* ⟹ *f I* ⊆ *G*) ⟹ *I* ⊆ *G* ⟹ *funpower f n I* ⊆ *G*
proof (*induct n*)
case *0* **thus** *?case* **by** *simp*
next
case (*Suc n*) **thus** *?case* **by** *simp*
qed

lemma *limit-upperbound*:
(⋀ *I*. *I* ⊆ *G* ⟹ *f I* ⊆ *G*) ⟹ *I* ⊆ *G* ⟹ *limit f I* ⊆ *G*
by (*simp add*: *funpower-upperbound limit-def natUnion-upperbound*)

lemma *elem-limit-simp*: *x* ∈ *limit f X* = (∃ *n*. *x* ∈ *funpower f n X*)
by (*auto simp add*: *limit-def natUnion-def*)

definition *pointwise* :: (*'a set* ⇒ *'b set*) ⇒ *bool* **where**

pointwise f = (∀ *X*. *f X* = $\bigcup$ { *f* {*x*} | *x*. *x* ∈ *X*})

lemma *natUnion-elem*: *x* ∈ *f n* ⟹ *x* ∈ *natUnion f*
using *natUnion-def* **by** *fastforce*

lemma *limit-elem*: *x* ∈ *funpower f n X* ⟹ *x* ∈ *limit f X*
by (*simp add*: *limit-def natUnion-elem*)

definition *pointbase* :: (′*a set* ⇒ ′*b set*) ⇒ ′*a set* ⇒ ′*b set* **where**
pointbase F I = $\bigcup$ { *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*
by (*simp add*: *chain-def mono-iff-le-Suc*)

lemma *setmonotone-implies-chain-funpower*:
assumes *setmonotone*: *setmonotone f*
shows *chain* (λ *n*. *funpower f n I*)
by (*simp add*: *chain-def setmonotone subset-setmonotone*)

lemma *natUnion-subset*: (⋀ *n*. ∃ *m*. *f n* ⊆ *g m*) ⟹ *natUnion f* ⊆ *natUnion g*
by (*meson natUnion-elem natUnion-upperbound subset-iff*)

lemma *natUnion-eq*[*case-names Subset Superset*]:
(⋀ *n*. ∃ *m*. *f n* ⊆ *g m*) ⟹ (⋀ *n*. ∃ *m*. *g n* ⊆ *f m*) ⟹ *natUnion f* = *natUnion g*
by (*simp add*: *natUnion-subset subset-antisym*)

lemma *natUnion-shift*[*symmetric*]:
assumes *chain*: *chain C*
shows *natUnion C* = *natUnion* (λ *n*. *C* (*n* + *m*))
proof (*induct rule*: *natUnion-eq*)
case (*Subset n*)
show *?case* **using** *chain chain-implies-mono le-add1 mono-def* **by** *blast*
next
case (*Superset n*)
show *?case* **by** *blast*
qed

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 −

have *setmonotone*: *setmonotone f* **using** *regular regular-def* **by** *blast*
have *continuous*: *continuous f* **using** *regular regular-def* **by** *blast*

let *?C* = λ *n. funpower f n I*
have *chain*: *chain ?C*
  by (*simp add*: *setmonotone setmonotone-implies-chain-funpower*)
have *f* (*limit f I*) = *f* (*natUnion ?C*)
  using *limit-def* **by** *metis*
also have *f* (*natUnion ?C*) = *natUnion* (*f o ?C*)
  by (*metis continuous continuous-def chain*)
also have *natUnion* (*f o ?C*) = *natUnion* (λ *n. f*(*funpower f n I*))
  by (*meson comp-apply*)
also have *natUnion* (λ *n. f*(*funpower f n I*)) = *natUnion* (λ *n. ?C* (*n* + *1*))
  by *simp*
also have *natUnion* (λ *n. ?C*(*n* + *1*)) = *natUnion ?C*
  apply (*subst natUnion-shift*)
  using *chain* **by** (*blast*+)
finally show *?thesis* **by** (*simp add*: *limit-def*)
qed

lemma *fix-is-fix-of-limit*:
  assumes *fixpoint*: *f I* = *I*
  shows *limit f I* = *I*
proof −
  have *funpower*: ⋀ *n. funpower f n I* = *I*
  proof −
    fix *n* :: *nat*
    from *fixpoint* **show** *funpower f n I* = *I*
      by (*induct n*, *auto*)
  qed
  show *?thesis* **by** (*simp add*: *limit-def funpower natUnion-def*)
qed

lemma *limit-is-idempotent*: *regular f* ⟹ *limit f* (*limit f I*) = *limit f I*
by (*simp add*: *fix-is-fix-of-limit regular-fixpoint*)

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

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

end
theory *CFG*
  imports *Main*
begin

2 Adjusted content from AFP/LocalLexing

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

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

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

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

definition *is-word* :: *'a cfg* ⇒ *'a list* ⇒ *bool* **where**
  is-word $\mathcal{G}$ ω = (*nonterminals* $\mathcal{G}$ ∩ *set* ω = {})

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

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

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

definition *derives* :: *'a cfg* ⇒ *'a list* ⇒ *'a list* ⇒ *bool* **where**
  derives $\mathcal{G}$ *u v* ≡ ((*u*, *v*) ∈ *derivations* $\mathcal{G}$)

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

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

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

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

end
theory *Derivations*
  imports
    CFG
begin

3 Adjusted content from AFP/LocalLexing

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

lemma *is-word-empty*: *is-word* $\mathcal{G}$ [] **by** (*auto simp add*: *is-word-def*)

lemma *derives1-implies-derives*[*simp*]:
  derives1 $\mathcal{G}$ *a b* $\Longrightarrow$ *derives* $\mathcal{G}$ *a b*
  by (*auto simp add*: *derives-def derivations-def derivations1-def*)

lemma *derives-trans*:
  derives $\mathcal{G}$ *a b* $\Longrightarrow$ *derives* $\mathcal{G}$ *b c* $\Longrightarrow$ *derives* $\mathcal{G}$ *a c*
  by (*auto simp add*: *derives-def derivations-def*)

lemma *derives1-eq-derivations1*:
  derives1 $\mathcal{G}$ *x y* = ((*x*, *y*) ∈ *derivations1* $\mathcal{G}$)
  by (*simp add*: *derivations1-def*)

lemma *derives-induct*[*consumes 1*, *case-names Base Step*]:
  assumes *derives*: *derives* $\mathcal{G}$ *a b*
  assumes *Pa*: *P a*
  assumes *induct*: $\bigwedge$*y z*. *derives* $\mathcal{G}$ *a y* $\Longrightarrow$ *derives1* $\mathcal{G}$ *y z* $\Longrightarrow$ *P y* $\Longrightarrow$ *P z*
  shows *P b*
proof −
  note *rtrancl-lemma* = *rtrancl-induct*[**where** *a* = *a* **and** *b* = *b* **and** *r* = *derivations1* $\mathcal{G}$ **and** *P*=*P*]
  from *derives Pa induct rtrancl-lemma* **show** *P b*
    by (*metis derives-def derivations-def derives1-eq-derivations1*)
qed

definition *Derives1* :: *'a cfg* ⇒ *'a list* ⇒ *nat* ⇒ *'a rule* ⇒ *'a list* ⇒ *bool* **where**
  Derives1 $\mathcal{G}$ *u i r v* ≡ ∃ *x y A α*.
    u = *x* @ [*A*] @ *y* ∧
    v = *x* @ *α* @ *y* ∧
    (*A*, *α*) ∈ *set* ($\mathfrak{R}$ $\mathcal{G}$) ∧ *r* = (*A*, *α*) ∧ *i* = *length x*

lemma *Derives1-split*:
  Derives1 $\mathcal{G}$ *u i r v* $\Longrightarrow$ ∃ *x y*. *u* = *x* @ [*fst r*] @ *y* ∧ *v* = *x* @ (*snd r*) @ *y* ∧ *length x* = *i*
  by (*metis Derives1-def fst-conv snd-conv*)

lemma *Derives1-implies-derives1*: *Derives1* $\mathcal{G}$ *u i r v* $\Longrightarrow$ *derives1* $\mathcal{G}$ *u v*
  by (*auto simp add*: *Derives1-def derives1-def*)

lemma *derives1-implies-Derives1*: *derives1* $\mathcal{G}$ *u v* $\Longrightarrow$ ∃ *i r*. *Derives1* $\mathcal{G}$ *u i r v*
  by (*auto simp add*: *Derives1-def derives1-def*)

fun *Derivation* :: *'a cfg* ⇒ *'a list* ⇒ *'a derivation* ⇒ *'a list* ⇒ *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 (*induct D arbitrary*: *a b*)
  case *Nil* **thus** *?case*
    by (*simp add*: *derives-def derivations-def*)
next
  case (*Cons d D*)
  note *ihyps* = *this*
  from *ihyps* **have** ∃ *x*. *Derives1* $\mathcal{G}$ *a* (*fst d*) (*snd d*) *x* ∧ *Derivation* $\mathcal{G}$ *x D b* **by** *auto*
  then obtain *x* **where** *Derives1* $\mathcal{G}$ *a* (*fst d*) (*snd d*) *x* **and** *xb*: *Derivation* $\mathcal{G}$ *x D b* **by** *blast*
  with *Derives1-implies-derives1* **have** *d1*: *derives* $\mathcal{G}$ *a x* **by** *fastforce*
  from *ihyps xb* **have** *d2*:*derives* $\mathcal{G}$ *x b* **by** *simp*
  show *derives* $\mathcal{G}$ *a b* **by** (*rule derives-trans*[*OF d1 d2*])
qed

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 (*induct S arbitrary*: *a y z i r*)
  case *Nil* **thus** *?case* **by** *simp*
next
  case (*Cons s S*) **thus** *?case*
    by (*metis Derivation.simps*(*2*) *append-Cons*)
qed

lemma *derives-implies-Derivation*: *derives* $\mathcal{G}$ *a b* ⟹ ∃ *D*. *Derivation* $\mathcal{G}$ *a D b*
proof (*induct rule*: *derives-induct*)
  case *Base*
  show *?case* **by** (*rule exI*[**where** *x*=[]], *simp*)
next
  case (*Step y z*)
  note *ihyps* = *this*
  from *ihyps* **obtain** *D* **where** *ay*: *Derivation* $\mathcal{G}$ *a D y* **by** *blast*
  from *ihyps derives1-implies-Derives1* **obtain** *i r* **where** *yz*: *Derives1* $\mathcal{G}$ *y i r z* **by** *blast*
  from *Derivation-Derives1*[*OF ay yz*] **show** *?case* **by** *auto*
qed

lemma *Derives1-rule* [*elim*]: *Derives1* $\mathcal{G}$ *a i r b* ⟹ *r* ∈ *set* ($\mathfrak{R}$ $\mathcal{G}$)
  using *Derives1-def* **by** *metis*

lemma *Derivation-append*: *Derivation* $\mathcal{G}$ *a* (*D*@*E*) *c* = (∃ *b*. *Derivation* $\mathcal{G}$ *a D b* ∧ *Derivation* $\mathcal{G}$ *b E c*)
  by (*induct D arbitrary*: *a c E*) *auto*

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*
using *Derivation-append* **by** *blast*

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 −
obtain *x y A* α **where** ∗:
u = *x* @ [*A*] @ *y* *v* = *x* @ α @ *y*
(*A*, α) ∈ *set* ($\mathfrak{R}$ $\mathcal{G}$) *r* = (*A*, α) *i* = *length x*
using *assms Derives1-def* **by** (*smt* (*verit*))
hence *w*@*u* = *w* @ *x* @ [*A*] @ *y* *w*@*v* = *w* @ *x* @ α @ *y*
by *auto*
thus *?thesis*
unfolding *Derives1-def* **using** ∗
apply (*rule-tac exI*[**where** *x*=*w*@*x*])
apply (*rule-tac exI*[**where** *x*=*y*])
by *simp*
qed

lemma *Derivation-prepend*:
Derivation $\mathcal{G}$ *b D b'* $\Longrightarrow$ *Derivation* $\mathcal{G}$ (*a*@*b*) (*map* (λ(*i*, *r*). (*i* + *length a*, *r*)) *D*) (*a*@*b'*)
using *Derives1-prepend* **by** (*induction D arbitrary*: *b b'*) (*auto*, *fast*)

lemma *Derives1-append*:
assumes *Derives1* $\mathcal{G}$ *u i r v*
shows *Derives1* $\mathcal{G}$ (*u*@*w*) *i r* (*v*@*w*)
proof −
obtain *x y A* α **where** ∗:
u = *x* @ [*A*] @ *y* *v* = *x* @ α @ *y*
(*A*, α) ∈ *set* ($\mathfrak{R}$ $\mathcal{G}$) *r* = (*A*, α) *i* = *length x*
using *assms Derives1-def* **by** (*smt* (*verit*))
hence *u*@*w* = *x* @ [*A*] @ *y* @ *w* *v*@*w* = *x* @ α @ *y* @ *w*
by *auto*
thus *?thesis*
unfolding *Derives1-def* **using** ∗
apply (*rule-tac exI*[**where** *x*=*x*])
apply (*rule-tac exI*[**where** *x*=*y*@*w*])
by *blast*
qed

lemma *Derivation-append'*:
Derivation $\mathcal{G}$ *a D a'* $\Longrightarrow$ *Derivation* $\mathcal{G}$ (*a*@*b*) *D* (*a'*@*b*)
using *Derives1-append* **by** (*induction D arbitrary*: *a a'*) (*auto*, *fast*)

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*)
using *assms Derivation-append' Derivation-prepend Derivation-implies-append*
by *fast*

lemma *derives1-if-valid-rule*:
(A, α) $\in$ *set* ($\mathfrak{R}$ $\mathcal{G}$) $\Longrightarrow$ *derives1* $\mathcal{G}$ [A] α
unfolding *derives1-def*
apply (*rule-tac exI*[**where** *x*=[]])
apply (*rule-tac exI*[**where** *x*=[]])
by *simp*

lemma *derives-if-valid-rule*:
(A, α) $\in$ *set* ($\mathfrak{R}$ $\mathcal{G}$) $\Longrightarrow$ *derives* $\mathcal{G}$ [A] α
using *derives1-if-valid-rule* **by** *fastforce*

lemma *Derivation-from-empty*:
Derivation $\mathcal{G}$ [] *D a* $\Longrightarrow$ *a* = []
by (*cases D*) (*auto simp*: *Derives1-def*)

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 (*induction D arbitrary*: *a b*)
case *Nil*
thus *?case*
by (*metis Derivation.simps*(*1*) *order-refl*)
next
case (*Cons d D*)
then obtain *ab* **where** $*$: *Derives1* $\mathcal{G}$ (*a*@*b*) (*fst d*) (*snd d*) *ab Derivation* $\mathcal{G}$ *ab D c*
by *auto*
then obtain *x y A* α **where** #:
a@*b* = *x* @ [*A*] @ *y ab* = *x* @ α @ *y* (*A*,α) $\in$ *set* ($\mathfrak{R}$ $\mathcal{G}$) *snd d* = (*A*,α) *fst d* = *length x*
using $*$ **unfolding** *Derives1-def* **by** *blast*
show *?case*
proof (*cases length a* $\leq$ *length x*)
case *True*
hence *ab-def*:
a = *take* (*length a*) *x*
b = *drop* (*length a*) *x* @ [*A*] @ *y*
ab = *take* (*length a*) *x* @ *drop* (*length a*) *x* @ α @ *y*
using #(*1*,*2*) *True* **by** (*metis append-eq-append-conv-if*)+
then obtain *E F a' b'* **where** *IH*:
Derivation $\mathcal{G}$ (*take* (*length a*) *x*) *E a'*
Derivation $\mathcal{G}$ (*drop* (*length a*) *x* @ α @ *y*) *F b'*
c = *a'* @ *b'*

length E ≤ length D
length F ≤ length D
using *Cons* ∗(*2*) **by** *blast*
have *Derives1 𝒢 b* (*fst d* − *length a*) (*snd d*) (*drop* (*length a*) *x* @ *α* @ *y*)
unfolding *Derives1-def* **using** ∗(*1*) #(*3*−*5*) *ab-def*(*2*) **by** (*metis length-drop*)
hence *Derivation 𝒢 b* ((*fst d* − *length a*, *snd d*) # *F*) *b′*
using *IH*(*2*) **by** *force*
moreover have *Derivation 𝒢 a E a′*
using *IH*(*1*) *ab-def*(*1*) **by** *fastforce*
ultimately show *?thesis*
using *IH*(*3*−*5*) **by** *fastforce*
next
case *False*
hence *a-def*: *a* = *x* @ [*A*] @ *take* (*length a* − *length x* − *1*) *y*
using #(*1*) *append-eq-conv-conj*[*of a b x* @ [*A*] @ *y*] *take-all-iff take-append*
by (*metis append-Cons append-Nil diff-is-0-eq le-cases take-Cons′*)
hence *b-def*: *b* = *drop* (*length a* − *length x* − *1*) *y*
using #(*1*) **by** (*metis List.append.assoc append-take-drop-id same-append-eq*)
have *ab* = *x* @ *α* @ *take* (*length a* − *length x* − *1*) *y* @ *drop* (*length a* − *length x* − *1*) *y*
using #(*2*) **by** *force*
then obtain *E F a′ b′* **where** *IH*:
Derivation 𝒢 (*x* @ *α* @ *take* (*length a* − *length x* − *1*) *y*) *E a′*
Derivation 𝒢 (*drop* (*length a* − *length x* − *1*) *y*) *F b′*
c = *a′* @ *b′*
length E ≤ length D
length F ≤ length D
using *Cons.IH*[*of x* @ *α* @ *take* (*length a* − *length x* − *1*) *y drop* (*length a* − *length x* − *1*) *y*] ∗(*2*) **by** *auto*
have *Derives1 𝒢 a* (*fst d*) (*snd d*) (*x* @ *α* @ *take* (*length a* − *length x* − *1*) *y*)
unfolding *Derives1-def* **using** #(*3*−*5*) *a-def* **by** *blast*
hence *Derivation 𝒢 a* ((*fst d*, *snd d*) # *E*) *a′*
using *IH*(*1*) **by** *fastforce*
moreover have *Derivation 𝒢 b F b′*
using *b-def IH*(*2*) **by** *blast*
ultimately show *?thesis*
using *IH*(*3*−*5*) **by** *fastforce*
qed
qed

lemma *Derivation-𝔖1*:
assumes *Derivation 𝒢* [𝔖 *𝒢*] *D ω is-word 𝒢 ω*
shows ∃*α E*. *Derivation 𝒢 α E ω* ∧ (𝔖 *𝒢*,*α*) ∈ *set* (𝔑 *𝒢*)
proof (*cases D*)
case *Nil*
thus *?thesis*
using *assms* **by** (*auto simp*: *is-word-def nonterminals-def*)
next
case (*Cons d D*)

then obtain α **where** *Derives1* $\mathcal{G}$ $[\mathfrak{S}\ \mathcal{G}]$ (*fst d*) (*snd d*) α *Derivation* $\mathcal{G}$ α *D* ω
  using *assms* **by** *auto*
hence $(\mathfrak{S}\ \mathcal{G},\ \alpha) \in$ *set* $(\mathfrak{R}\ \mathcal{G})$
  unfolding *Derives1-def*
  by (*simp add*: *Cons-eq-append-conv*)
thus *?thesis*
  using ‹*Derivation* $\mathcal{G}$ α *D* ω› **by** *auto*
qed

end
theory *Earley*
  imports
    Derivations
begin

5 Slices

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

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

notation (*latex* **output**)
  slice (‹$\text{-}_{\text{-}'/\text{-}}$› [*1000*,*0*,*0*] *1000*)

lemma *slice-drop-take*:
  slice xs a b = *drop a* (*take b xs*)
  by (*induction xs a b rule*: *slice.induct*) *auto*

lemma *slice-append-aux*:
  Suc b ≤ *c* ⟹ *slice* (*x#xs*) (*Suc b*) *c* = *slice xs b* (*c*−*1*)
  using *Suc-le-D* **by** *fastforce*

lemma *slice-concat*:
  a ≤ *b* ⟹ *b* ≤ *c* ⟹ *slice xs a b* @ *slice xs b c* = *slice xs a c*
proof (*induction xs a b arbitrary*: *c rule*: *slice.induct*)
  case (*3 b x xs*)
  then show *?case*
    using *Suc-le-D* **by**(*fastforce simp*: *slice-append-aux*)
qed (*auto simp*: *slice-append-aux*)

lemma *slice-concat-Ex*:
  a ≤ *c* ⟹ *slice xs a c* = *ys* @ *zs* ⟹ ∃ *b*. *ys* = *slice xs a b* ∧ *zs* = *slice xs b c* ∧
a ≤ *b* ∧ *b* ≤ *c*
proof (*induction xs a c arbitrary*: *ys zs rule*: *slice.induct*)

case (*3 x xs b*)
show *?case*
proof (*cases ys*)
case *Nil*
then obtain *zs'* **where** x # *slice xs 0 b* = x # *zs'* x # *zs'* = *zs*
using *3.prems*(*2*) **by** *auto*
thus *?thesis*
using *Nil* **by** *force*
next
case (*Cons y ys'*)
then obtain *ys'* **where** x # *slice xs 0 b* = x # *ys'* @ *zs* x # *ys'* = *ys*
using *3.prems*(*2*) **by** *auto*
thus *?thesis*
using *3.IH*[*of ys' zs*] **by** *force*
qed
next
case (*4 a b x xs*)
thus *?case*
by (*auto*, *metis slice.simps*(*4*) *Suc-le-mono*)
qed *auto*

lemma *slice-nth*:
$a < \mathit{length}\ xs \Longrightarrow \mathit{slice}\ xs\ a\ (a+1) = [xs!a]$
unfolding *slice-drop-take*
by (*metis Cons-nth-drop-Suc One-nat-def diff-add-inverse drop-take take-Suc-Cons take-eq-Nil*)

lemma *slice-append-nth*:
$a \le b \Longrightarrow b < \mathit{length}\ xs \Longrightarrow \mathit{slice}\ xs\ a\ b\ @\ [xs!b] = \mathit{slice}\ xs\ a\ (b+1)$
by (*metis le-add1 slice-concat slice-nth*)

lemma *slice-empty*:
$b \le a \Longrightarrow \mathit{slice}\ xs\ a\ b = []$
by (*simp add*: *slice-drop-take*)

lemma *slice-id*[*simp*]:
$\mathit{slice}\ xs\ 0\ (\mathit{length}\ xs) = xs$
by (*simp add*: *slice-drop-take*)

lemma *slice-singleton*:
$b \le \mathit{length}\ xs \Longrightarrow [x] = \mathit{slice}\ xs\ a\ b \Longrightarrow b = a + 1$
by (*induction xs a b rule*: *slice.induct*) (*auto simp*: *slice-drop-take*)

6 Earley recognizer

6.1 Earley items

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

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

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

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

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

definition *α-item* :: *'a item* ⇒ *'a list* **where**
α-item x ≡ *take* (*dot-item x*) (*rhs-item x*)

definition *β-item* :: *'a item* ⇒ *'a list* **where**
β-item x ≡ *drop* (*dot-item x*) (*rhs-item x*)

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

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

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

definition *is-finished* :: *'a cfg* ⇒ *'a list* ⇒ *'a item* ⇒ *bool* **where**
is-finished $\mathcal{G}$ *ω x* ≡
lhs-item x = $\mathfrak{S}$ $\mathcal{G}$ ∧
start-item x = *0* ∧
end-item x = *length ω* ∧
is-complete x

definition *recognizing* :: *'a item set* ⇒ *'a cfg* ⇒ *'a list* ⇒ *bool* **where**
recognizing I $\mathcal{G}$ *ω* ≡ ∃ *x* ∈ *I*. *is-finished* $\mathcal{G}$ *ω x*

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

$y \in Earley\ \mathcal{G}\ \omega \Longrightarrow$
$\quad is\text{-}complete\ y \Longrightarrow next\text{-}symbol\ x = Some\ (lhs\text{-}item\ y) \Longrightarrow$
$\quad\quad Item\ r_x\ (b_x + 1)\ i\ k \in Earley\ \mathcal{G}\ \omega$

6.2 Well-formedness

definition *wf-item* :: *'a cfg* ⇒ *'a list* => *'a item* ⇒ *bool* **where**
wf-item $\mathcal{G}$ ω *x* ≡
rule-item x ∈ *set* ($\mathfrak{R}$ $\mathcal{G}$) ∧
dot-item x ≤ *length* (*rhs-item x*) ∧
start-item x ≤ *end-item x* ∧
end-item x ≤ *length* ω

lemma *wf-Init*:
assumes *r* ∈ *set* ($\mathfrak{R}$ $\mathcal{G}$) *fst r* = $\mathfrak{S}$ $\mathcal{G}$
shows *wf-item* $\mathcal{G}$ ω (*Item r 0 0 0*)
using *assms* **unfolding** *wf-item-def* **by** *simp*

lemma *wf-Scan*:
assumes *x* = *Item r b i j* *wf-item* $\mathcal{G}$ ω *x* ω!*j* = *a j* < *length* ω *next-symbol x* = *Some a*
shows *wf-item* $\mathcal{G}$ ω (*Item r* (*b* + *1*) *i* (*j*+*1*))
using *assms* **unfolding** *wf-item-def* **by** (*auto simp*: *item-defs is-complete-def next-symbol-def split*: *if-splits*)

lemma *wf-Predict*:
assumes *x* = *Item r b i j* *wf-item* $\mathcal{G}$ ω *x r'* ∈ *set* ($\mathfrak{R}$ $\mathcal{G}$) *next-symbol x* = *Some* (*lhs-rule r'*)
shows *wf-item* $\mathcal{G}$ ω (*Item r' 0 j j*)
using *assms* **unfolding** *wf-item-def* **by** *simp*

lemma *wf-Complete*:
assumes *x* = *Item* r_x b_x *i j* *wf-item* $\mathcal{G}$ ω *x y* = *Item* r_y b_y *j k* *wf-item* $\mathcal{G}$ ω *y*
assumes *is-complete y next-symbol x* = *Some* (*lhs-item y*)
shows *wf-item* $\mathcal{G}$ ω (*Item* r_x (b_x + *1*) *i k*)
using *assms* **unfolding** *wf-item-def is-complete-def next-symbol-def rhs-item-def*
by (*auto split*: *if-splits*)

lemma *wf-Earley*:
assumes *x* ∈ *Earley* $\mathcal{G}$ ω
shows *wf-item* $\mathcal{G}$ ω *x*
using *assms wf-Init wf-Scan wf-Predict wf-Complete*
by (*induction rule*: *Earley.induct*) *fast*+

6.3 Soundness

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

lemma *sound-Init*:
  assumes $r \in set\ (\mathfrak{R}\ \mathcal{G})\ fst\ r = \mathfrak{S}\ \mathcal{G}$
  shows *sound-item* $\mathcal{G}\ \omega$ (*Item r 0 0 0*)
proof −
  let *?x* = *Item r 0 0 0*
  have (*lhs-item ?x*, *β-item ?x*) ∈ *set* ($\mathfrak{R}\ \mathcal{G}$)
    using *assms*(*1*) **by** (*simp add*: *item-defs*)
  hence *derives* $\mathcal{G}$ [*lhs-item ?x*] (*β-item ?x*)
    using *derives-if-valid-rule* **by** *metis*
  thus *sound-item* $\mathcal{G}\ \omega$ *?x*
    unfolding *sound-item-def* **by** (*simp add*: *slice-empty*)
qed

lemma *sound-Scan*:
  assumes *x* = *Item r b i j wf-item* $\mathcal{G}\ \omega$ *x sound-item* $\mathcal{G}\ \omega$ *x*
  assumes $\omega!j = a\ j <$ *length* ω *next-symbol x* = *Some a*
  shows *sound-item* $\mathcal{G}\ \omega$ (*Item r* (*b+1*) *i* (*j+1*))
proof −
  define *x′* **where** [*simp*]: *x′* = *Item r* (*b+1*) *i* (*j+1*)
  obtain *β-item′* **where** ∗: *β-item x* = *a* # *β-item′ β-item x′* = *β-item′*
    using *assms*(*1*,*6*) **apply** (*auto simp*: *item-defs next-symbol-def is-complete-def split*: *if-splits*)
    by (*metis Cons-nth-drop-Suc leI*)
  have *slice* ω *i j* @ *β-item x* = *slice* ω *i* (*j+1*) @ *β-item′*
    using ∗ *assms*(*1*,*2*,*4*,*5*) **by** (*auto simp*: *slice-append-nth wf-item-def*)
  moreover have *derives* $\mathcal{G}$ [*lhs-item x*] (*slice* ω *i j* @ *β-item x*)
    using *assms*(*1*,*3*) *sound-item-def* **by** *force*
  ultimately show *?thesis*
    using *assms*(*1*) ∗ **by** (*auto simp*: *item-defs sound-item-def*)
qed

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* (*lhs-rule r′*)
  shows *sound-item* $\mathcal{G}\ \omega$ (*Item r′ 0 j j*)
  using *assms* **by** (*auto simp*: *sound-item-def derives-if-valid-rule slice-empty item-defs*)

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* (*lhs-item y*)
  shows *sound-item* $\mathcal{G}\ \omega$ ($Item\ r_x\ (b_x + 1)\ i\ k$)
proof −
  have *derives* $\mathcal{G}$ [*lhs-item y*] (*slice* ω *j k*)
    using *assms*(*4*,*6*,*7*) **by** (*auto simp*: *sound-item-def is-complete-def item-defs*)
  then obtain *E* **where** *E*: *Derivation* $\mathcal{G}$ [*lhs-item y*] *E* (*slice* ω *j k*)
    using *derives-implies-Derivation* **by** *blast*
  have *derives* $\mathcal{G}$ [*lhs-item x*] (*slice* ω *i j* @ *β-item x*)
    using *assms*(*1*,*3*,*4*) **by** (*auto simp*: *sound-item-def*)

moreover have *0*: *β-item x* = (*lhs-item y*) # *tl* (*β-item x*)
using *assms*(*8*) **apply** (*auto simp*: *next-symbol-def is-complete-def item-defs split*: *if-splits*)
by (*metis drop-eq-Nil hd-drop-conv-nth leI list.collapse*)
ultimately obtain *D* **where** *D*:
Derivation $\mathcal{G}$ [*lhs-item x*] *D* (*slice* ω *i j* @ [*lhs-item y*] @ (*tl* (*β-item x*)))
using *derives-implies-Derivation* **by** (*metis append-Cons append-Nil*)
obtain *F* **where** *F*:
Derivation $\mathcal{G}$ [*lhs-item x*] *F* (*slice* ω *i j* @ *slice* ω *j k* @ *tl* (*β-item x*))
using *Derivation-append-rewrite D E*
by *metis*
moreover have $i \leq j$
using *assms*(*1*,*2*) *wf-item-def* **by** *force*
moreover have $j \leq k$
using *assms*(*4*,*5*) *wf-item-def* **by** *force*
ultimately have *derives* $\mathcal{G}$ [*lhs-item x*] (*slice* ω *i k* @ *tl* (*β-item x*))
by (*metis Derivation-implies-derives append.assoc slice-concat*)
thus *sound-item* $\mathcal{G}$ ω (*Item* r_x (b_x + *1*) *i k*)
using *assms*(*1*,*4*) **by** (*auto simp*: *sound-item-def item-defs drop-Suc tl-drop*)
qed

lemma *sound-Earley*:
assumes *x* ∈ *Earley* $\mathcal{G}$ ω *wf-item* $\mathcal{G}$ ω *x*
shows *sound-item* $\mathcal{G}$ ω *x*
using *assms*
proof (*induction rule*: *Earley.induct*)
case (*Init r*)
thus *?case*
using *sound-Init* **by** *blast*
next
case (*Scan x r b i j a*)
thus *?case*
using *wf-Earley sound-Scan* **by** *fast*
next
case (*Predict x r b i j r′*)
thus *?case*
using *wf-Earley sound-Predict* **by** *blast*
next
case (*Complete x* r_x b_x *i j y* r_y b_y *k*)
thus *?case*
using *wf-Earley sound-Complete* **by** *metis*
qed

theorem *soundness-Earley*:
assumes *recognizing* (*Earley* $\mathcal{G}$ ω) $\mathcal{G}$ ω
shows $\mathcal{G} \vdash [\mathfrak{S}\ \mathcal{G}] \Rightarrow^* \omega$
proof −
obtain *x* **where** *x*: *x* ∈ *Earley* $\mathcal{G}$ ω *is-finished* $\mathcal{G}$ ω *x*
using *assms recognizing-def* **by** *blast*

hence *sound-item* $\mathcal{G}$ ω x
using *wf-Earley sound-Earley* **by** *blast*
thus *?thesis*
unfolding *sound-item-def* **using** x **by** (*auto simp*: *is-finished-def is-complete-def item-defs*)
qed

6.4 Completeness

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

lemma *partially-completed-upto*:
assumes $j \le k\ k \le length\ \omega$
assumes $x = Item\ (N,\alpha)\ d\ i\ j\ x \in I\ \forall x \in I.\ wf\text{-}item\ \mathcal{G}\ \omega\ x$
assumes $Derivation\ \mathcal{G}\ (\beta\text{-}item\ x)\ D\ (slice\ \omega\ j\ k)$
assumes $partially\text{-}completed\ k\ \mathcal{G}\ \omega\ I\ (\lambda D'.\ length\ D' \le length\ D)$
shows $Item\ (N,\alpha)\ (length\ \alpha)\ i\ k \in I$
using *assms*
proof (*induction* β*-item* x *arbitrary*: $d\ i\ j\ k\ N\ \alpha\ x\ D$)
case *Nil*
have $\alpha\text{-}item\ x = \alpha$
using *Nil*(*1*,*4*) **unfolding** α*-item-def* β*-item-def rhs-item-def rhs-rule-def* **by** *simp*
hence $x = Item\ (N,\alpha)\ (length\ \alpha)\ i\ j$
using *Nil.hyps Nil.prems*($3-5$) **unfolding** *wf-item-def item-defs* **by** *auto*
have $Derivation\ \mathcal{G}\ []\ D\ (slice\ \omega\ j\ k)$
using *Nil.hyps Nil.prems*(*6*) **by** *auto*
hence $slice\ \omega\ j\ k = []$
using *Derivation-from-empty* **by** *blast*
hence $j = k$
unfolding *slice-drop-take* **using** *Nil.prems*(*1*,*2*) **by** *simp*
thus *?case*
using ‹$x = Item\ (N,\ \alpha)\ (length\ \alpha)\ i\ j$› *Nil.prems*(*4*) **by** *blast*
next
case (*Cons b bs*)
obtain $j'\ E\ F$ **where** $*$:
$Derivation\ \mathcal{G}\ [b]\ E\ (slice\ \omega\ j\ j')$
$Derivation\ \mathcal{G}\ bs\ F\ (slice\ \omega\ j'\ k)$
$j \le j'\ j' \le k\ length\ E \le length\ D\ length\ F \le length\ D$
using *Derivation-concat-split*[*of* $\mathcal{G}$ [b] *bs D slice* ω *j k*] *slice-concat-Ex*
using *Cons.hyps*(*2*) *Cons.prems*(*1*,*6*)
by (*smt* (*verit*, *ccfv-threshold*) *Cons-eq-appendI append-self-conv2*)
have $next\text{-}symbol\ x = Some\ b$

```
    using Cons.hyps(2) unfolding item-defs(4) next-symbol-def is-complete-def
by (auto, metis nth-via-drop)
  hence Item (N, α) (d+1) i j' ∈ I
    using Cons.prems(7) unfolding partially-completed-def
    using Cons.prems(2,3,4) *(1,3−5) by blast
  moreover have partially-completed k 𝒢 ω I (λD'. length D' ≤ length F)
    using Cons.prems(7) *(6) unfolding partially-completed-def by fastforce
  moreover have bs = β-item (Item (N,α) (d+1) i j')
    using Cons.hyps(2) Cons.prems(3) unfolding item-defs(4) rhs-item-def
    by (auto, metis List.list.sel(3) drop-Suc drop-tl)
  ultimately show ?case
    using Cons.hyps(1) *(2,4) Cons.prems(2,3,5) wf-item-def by blast
qed

lemma partially-completed-Earley:
  partially-completed k 𝒢 ω (Earley 𝒢 ω) (λ-. True)
unfolding partially-completed-def
proof (intro allI impI)
  fix r b i' i j x a D
  assume
    i ≤ j ∧ j ≤ k ∧ k ≤ length ω ∧
     x = Item r b i' i ∧ x ∈ Earley 𝒢 ω ∧
     next-symbol x = Some a ∧
     Derivation 𝒢 [a] D (slice ω i j) ∧ True
  thus Item r (b + 1) i' j ∈ Earley 𝒢 ω
  proof (induction length D arbitrary: r b i' i j x a D rule: nat-less-induct)
    case 1
    show ?case
    proof cases
      assume D = []
      hence [a] = slice ω i j
        using 1.prems by force
      moreover have j ≤ length ω
        using le-trans 1.prems by blast
      ultimately have j = i+1
        using slice-singleton by metis
      hence i < length ω
        using ‹j ≤ length ω› by simp
      hence ω!i = a
        using slice-nth ‹[a] = slice ω i j› ‹j = i + 1› by fastforce
      hence Item r (b + 1) i' j ∈ Earley 𝒢 ω
        using Earley.Scan 1.prems ‹i < length ω› ‹j = i + 1› by metis
      thus ?thesis
        by (simp add: ‹j = i + 1›)
    next
      assume ¬ D = []
      then obtain d D' where D = d # D'
        by (meson List.list.exhaust)
     then obtain α where *: Derives1 𝒢 [a] (fst d) (snd d) α Derivation 𝒢 α D'
```

(*slice* ω *i j*)
using *1.prems* **by** *auto*
hence *rule*: $(a, \alpha) \in$ *set* $(\mathfrak{R}\ \mathcal{G})$ *fst d* = *0 snd d* = $(a\ ,\alpha)$
using $*(1)$ **unfolding** *Derives1-def* **by** (*simp add*: *Cons-eq-append-conv*)+
define *y* **where** *y-def*: *y* = *Item* $(a\ ,\alpha)$ *0 i i*
have *length* $D' <$ *length D*
using ‹$D = d\ \#\ D'$› **by** *fastforce*
hence *partially-completed k* $\mathcal{G}\ \omega$ (*Earley* $\mathcal{G}\ \omega$) (λE. *length E* $\leq$ *length* D')
unfolding *partially-completed-def* **using** *1.hyps order-le-less-trans* **by** (*smt* (*verit*, *best*))
hence *partially-completed j* $\mathcal{G}\ \omega$ (*Earley* $\mathcal{G}\ \omega$) (λE. *length E* $\leq$ *length* D')
unfolding *partially-completed-def* **using** *1.prems* **by** *force*
moreover have *Derivation* $\mathcal{G}$ (β-*item y*) D' (*slice* ω *i j*)
using $*(2)$ **by** (*auto simp*: *item-defs y-def*)
moreover have $y \in$ *Earley* $\mathcal{G}\ \omega$
using *y-def 1.prems rule* **by** (*auto simp*: *item-defs Earley.Predict*)
moreover have $j \leq$ *length* ω
using *1.prems* **by** *simp*
ultimately have *Item* (a,α) (*length* α) *i j* $\in$ *Earley* $\mathcal{G}\ \omega$
using *partially-completed-upto 1.prems wf-Earley y-def* **by** *metis*
moreover have *x*: *x* = *Item r b* i' *i x* $\in$ *Earley* $\mathcal{G}\ \omega$
using *1.prems* **by** *blast*+
moreover have *next-symbol x* = *Some a*
using *1.prems* **by** *linarith*
ultimately show *?thesis*
using *Earley.Complete*[*OF x*] **by** (*auto simp*: *is-complete-def item-defs*)
qed
qed
qed

theorem *completeness-Earley*:
assumes $\mathcal{G} \vdash [\mathfrak{S}\ \mathcal{G}] \Rightarrow^* \omega$ *is-word* $\mathcal{G}\ \omega$
shows *recognizing* (*Earley* $\mathcal{G}\ \omega$) $\mathcal{G}\ \omega$
proof −
obtain α *D* **where** $*$: $(\mathfrak{S}\ \mathcal{G}\ ,\alpha) \in$ *set* $(\mathfrak{R}\ \mathcal{G})$ *Derivation* $\mathcal{G}\ \alpha$ *D* ω
using *Derivation-*$\mathfrak{S}$*1 assms derives-implies-Derivation* **by** *metis*
define *x* **where** *x-def*: *x* = *Item* $(\mathfrak{S}\ \mathcal{G}, \alpha)$ *0 0 0*
have *partially-completed* (*length* ω) $\mathcal{G}\ \omega$ (*Earley* $\mathcal{G}\ \omega$) (λ-. *True*)
using *assms*(*2*) *partially-completed-Earley* **by** *blast*
hence *0*: *partially-completed* (*length* ω) $\mathcal{G}\ \omega$ (*Earley* $\mathcal{G}\ \omega$) ($\lambda D'$. *length* $D' \leq$ *length D*)
unfolding *partially-completed-def* **by** *blast*
have *1*: $x \in$ *Earley* $\mathcal{G}\ \omega$
using *x-def Earley.Init* $*(1)$ **by** *fastforce*
have *2*: *Derivation* $\mathcal{G}$ (β-*item x*) *D* (*slice* ω *0* (*length* ω))
using $*(2)$ *x-def* **by** (*simp add*: *item-defs*)
have *Item* $(\mathfrak{S}\ \mathcal{G},\alpha)$ (*length* α) *0* (*length* ω) $\in$ *Earley* $\mathcal{G}\ \omega$
using *partially-completed-upto*[*OF - - - - - 2 0*] *wf-Earley 1 x-def* **by** *auto*
then show *?thesis*

unfolding *recognizing-def is-finished-def* **by** (*auto simp*: *is-complete-def item-defs*, *force*)
qed

6.5 Correctness

theorem *correctness-Earley*:
assumes *is-word* $\mathcal{G}$ ω
shows *recognizing* (*Earley* $\mathcal{G}$ ω) $\mathcal{G}$ $\omega \longleftrightarrow \mathcal{G} \vdash [\mathfrak{S}\ \mathcal{G}] \Rightarrow^* \omega$
using *assms soundness-Earley completeness-Earley* **by** *blast*

6.6 Finiteness

lemma *finiteness-empty*:
set $(\mathfrak{R}\ \mathcal{G}) = \{\} \Longrightarrow$ *finite* $\{ x \mid x.$ *wf-item* $\mathcal{G}\ \omega\ x \}$
unfolding *wf-item-def* **by** *simp*

fun *item-intro* :: *'a rule* × *nat* × *nat* × *nat* ⇒ *'a item* **where**
item-intro (*rule*, *dot*, *origin*, *ends*) = *Item rule dot origin ends*

lemma *finiteness-nonempty*:
assumes *set* $(\mathfrak{R}\ \mathcal{G}) \neq \{\}$
shows *finite* $\{ x.$ *wf-item* $\mathcal{G}\ \omega\ x \}$
proof −
define *M* **where** *M* = *Max* { *length* (*rhs-rule r*) | *r*. *r* ∈ *set* $(\mathfrak{R}\ \mathcal{G})$ }
define *Top* **where** *Top* = (*set* $(\mathfrak{R}\ \mathcal{G})$ × {*0*..*M*} × {*0*..*length* ω} × {*0*..*length* ω})
hence *finite Top*
using *finite-cartesian-product finite* **by** *blast*
have *inj-on item-intro Top*
unfolding *Top-def inj-on-def* **by** *simp*
hence *finite* (*item-intro* ‘ *Top*)
using *finite-image-iff* ‹*finite Top*› **by** *auto*
have $\{ x \mid x.$ *wf-item* $\mathcal{G}\ \omega\ x \} \subseteq$ *item-intro* ‘ *Top*
proof *standard*
fix *x*
assume $x \in \{ x \mid x.$ *wf-item* $\mathcal{G}\ \omega\ x \}$
then obtain *rule dot origin endp* **where** ∗: *x* = *Item rule dot origin endp*
rule ∈ *set* $(\mathfrak{R}\ \mathcal{G})$ *dot* ≤ *length* (*rhs-item x*) *origin* ≤ *length* ω *endp* ≤ *length* ω
unfolding *wf-item-def* **using** *item.exhaust-sel le-trans* **by** *blast*
hence *length* (*rhs-rule rule*) ∈ { *length* (*rhs-rule r*) | *r*. *r* ∈ *set* $(\mathfrak{R}\ \mathcal{G})$ }
using ∗(*1*,*2*) *rhs-item-def* **by** *blast*
moreover have *finite* { *length* (*rhs-rule r*) | *r*. *r* ∈ *set* $(\mathfrak{R}\ \mathcal{G})$ }
using *finite finite-image-set*[*of* λ*x*. *x* ∈ *set* $(\mathfrak{R}\ \mathcal{G})$] **by** *fastforce*
ultimately have *M* ≥ *length* (*rhs-rule rule*)
unfolding *M-def* **by** *simp*
hence *dot* ≤ *M*
using ∗(*1*,*3*) *rhs-item-def* **by** (*metis item.sel*(*1*) *le-trans*)
hence (*rule*, *dot*, *origin*, *endp*) ∈ *Top*
using ∗(*2*,*4*,*5*) **unfolding** *Top-def* **by** *simp*

thus *x* ∈ *item-intro* ‘ *Top*
using ∗(*1*) **by** *force*
qed
thus *?thesis*
using ‹*finite* (*item-intro* ‘ *Top*)› *rev-finite-subset* **by** *auto*
qed

lemma *finiteness-UNIV-wf-item*:
finite { *x. wf-item* $\mathcal{G}$ *ω x* }
using *finiteness-empty finiteness-nonempty* **by** *fastforce*

theorem *finiteness-Earley*:
finite (*Earley* $\mathcal{G}$ *ω*)
using *finiteness-UNIV-wf-item wf-Earley rev-finite-subset* **by** (*metis mem-Collect-eq subsetI*)

end
theory *Earley-Fixpoint*
imports
Earley
Limit
begin

7 Earley fixpoint

7.1 Definitions

definition *init-item* :: *'a rule* ⇒ *nat* ⇒ *'a item* **where**
init-item r k ≡ *Item r 0 k k*

definition *inc-item* :: *'a item* ⇒ *nat* ⇒ *'a item* **where**
inc-item x k ≡ *Item* (*rule-item x*) (*dot-item x* + *1*) (*start-item x*) *k*

definition *bin* :: *'a item set* ⇒ *nat* ⇒ *'a item set* **where**
bin I k ≡ { *x* . *x* ∈ *I* ∧ *end-item x* = *k* }

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

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

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

definition $Scan_F$ :: *nat* ⇒ *'a list* ⇒ *'a item set* ⇒ *'a item set* **where**
$Scan_F$ *k ω I* ≡ { *inc-item x* (*k*+*1*) | *x a*.

$x \in bin\ I\ k\ \wedge$
$\omega!k = a\ \wedge$
$k < length\ \omega\ \wedge$
$next\text{-}symbol\ x = Some\ a\ \}$

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

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

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

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

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

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

7.2 Monotonicity and Absorption

lemma $Earley_F\text{-}bin\text{-}step\text{-}empty$:
$Earley_F\text{-}bin\text{-}step\ k\ \mathcal{G}\ \omega\ \{\} = \{\}$
unfolding $Earley_F\text{-}bin\text{-}step\text{-}def\ Scan_F\text{-}def\ Complete_F\text{-}def\ Predict_F\text{-}def\ bin\text{-}def$
by *blast*

lemma $Earley_F\text{-}bin\text{-}step\text{-}setmonotone$:
$setmonotone\ (Earley_F\text{-}bin\text{-}step\ k\ \mathcal{G}\ \omega)$
by ($simp\ add$: $Un\text{-}assoc\ Earley_F\text{-}bin\text{-}step\text{-}def\ setmonotone\text{-}def$)

lemma $Earley_F\text{-}bin\text{-}step\text{-}continuous$:
$continuous\ (Earley_F\text{-}bin\text{-}step\ k\ \mathcal{G}\ \omega)$
unfolding *continuous-def*
proof (*standard*, *standard*, *standard*)
fix C :: $nat \Rightarrow {'a}\ item\ set$

assume *chain C*
thus *chain* ($Earley_F$*-bin-step* $k\ \mathcal{G}\ \omega \circ C$)
unfolding *chain-def* $Earley_F$*-bin-step-def* **by** (*auto simp*: $Scan_F$*-def* $Predict_F$*-def* $Complete_F$*-def bin-def subset-eq*)
next
fix C :: *nat* $\Rightarrow$ *'a item set*
assume $*$: *chain C*
show $Earley_F$*-bin-step* $k\ \mathcal{G}\ \omega$ (*natUnion C*) $=$ *natUnion* ($Earley_F$*-bin-step* $k\ \mathcal{G}\ \omega \circ C$)
unfolding *natUnion-def*
proof *standard*
show $Earley_F$*-bin-step* $k\ \mathcal{G}\ \omega$ ($\bigcup$ $\{C\ n\ |n.\ True\}$) $\subseteq \bigcup$ $\{(Earley_F\text{-}bin\text{-}step\ k\ \mathcal{G}\ \omega \circ C)\ n\ |n.\ True\}$
proof *standard*
fix x
assume #: $x \in Earley_F\text{-}bin\text{-}step\ k\ \mathcal{G}\ \omega\ (\bigcup\ \{C\ n\ |n.\ True\})$
show $x \in \bigcup\ \{(Earley_F\text{-}bin\text{-}step\ k\ \mathcal{G}\ \omega \circ C)\ n\ |n.\ True\}$
proof (*cases* $x \in Complete_F\ k\ (\bigcup\ \{C\ n\ |n.\ True\})$)
case *True*
then show *?thesis*
using $*$ **unfolding** *chain-def* $Earley_F$*-bin-step-def* $Complete_F$*-def bin-def*
apply *auto*
proof $-$
fix y :: *'a item* **and** z :: *'a item* **and** n :: *nat* **and** m :: *nat*
assume *a1*: *is-complete z*
assume *a2*: *end-item y* $=$ *start-item z*
assume *a3*: $y \in C\ n$
assume *a4*: $z \in C\ m$
assume *a5*: *next-symbol y* $=$ *Some* (*lhs-item z*)
assume $\forall i.\ C\ i \subseteq C\ (Suc\ i)$
hence *f6*: $\bigwedge n\ m.\ \neg\ n \leq m \vee C\ n \subseteq C\ m$
by (*meson lift-Suc-mono-le*)
hence *f7*: $\bigwedge n.\ \neg\ m \leq n \vee z \in C\ n$
using *a4* **by** *blast*
have $\exists n \geq m.\ y \in C\ n$
using *f6 a3* **by** (*meson le-sup-iff subset-eq sup-ge1*)
thus $\exists I.$
($\exists n.\ I = C\ n\ \cup$
$Scan_F$ (*end-item z*) ω ($C\ n$) $\cup$
$\{$*inc-item i* (*end-item z*) $|i.$
$i \in C\ n\ \wedge$
($\exists j.$
end-item i $=$ *start-item j* $\wedge$
$j \in C\ n\ \wedge$
end-item j $=$ *end-item z* $\wedge$
is-complete j $\wedge$
next-symbol i $=$ *Some* (*lhs-item j*))$\}$ $\cup$
$Predict_F$ (*end-item z*) $\mathcal{G}$ ($C\ n$))
$\wedge$ *inc-item y* (*end-item z*) $\in I$

using *f7 a5 a2 a1* **by** *blast*
qed
next
case *False*
thus *?thesis*
using # *Un-iff* **by** (*auto simp*: *Earley$_F$-bin-step-def Scan$_F$-def Predict$_F$-def bin-def*; *blast*)
qed
qed
next
show $\bigcup$ {(*Earley$_F$-bin-step k* $\mathcal{G}$ ω $\circ$ *C*) *n* |*n*. *True*} $\subseteq$ *Earley$_F$-bin-step k* $\mathcal{G}$ ω ($\bigcup$ {*C n* |*n*. *True*})
unfolding *Earley$_F$-bin-step-def*
using $*$ **by** (*auto simp*: *Scan$_F$-def Predict$_F$-def Complete$_F$-def chain-def bin-def*, *metis+*)
qed
qed

lemma *Earley$_F$-bin-step-regular*:
regular (*Earley$_F$-bin-step k* $\mathcal{G}$ ω)
by (*simp add*: *Earley$_F$-bin-step-continuous Earley$_F$-bin-step-setmonotone regular-def*)

lemma *Earley$_F$-bin-idem*:
Earley$_F$-bin k $\mathcal{G}$ ω (*Earley$_F$-bin k* $\mathcal{G}$ ω *I*) = *Earley$_F$-bin k* $\mathcal{G}$ ω *I*
by (*simp add*: *Earley$_F$-bin-def Earley$_F$-bin-step-regular limit-is-idempotent*)

lemma *Scan$_F$-bin-absorb*:
Scan$_F$ k ω (*bin I k*) = *Scan$_F$ k* ω *I*
unfolding *Scan$_F$-def bin-def* **by** *simp*

lemma *Predict$_F$-bin-absorb*:
Predict$_F$ k $\mathcal{G}$ (*bin I k*) = *Predict$_F$ k* $\mathcal{G}$ *I*
unfolding *Predict$_F$-def bin-def* **by** *simp*

lemma *Scan$_F$-Un*:
Scan$_F$ k ω (*I* $\cup$ *J*) = *Scan$_F$ k* ω *I* $\cup$ *Scan$_F$ k* ω *J*
unfolding *Scan$_F$-def bin-def* **by** *blast*

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*
unfolding *Predict$_F$-def bin-def* **by** *blast*

lemma *Scan$_F$-sub-mono*:
I $\subseteq$ *J* $\Longrightarrow$ *Scan$_F$ k* ω *I* $\subseteq$ *Scan$_F$ k* ω *J*
unfolding *Scan$_F$-def bin-def* **by** *blast*

lemma *Predict$_F$-sub-mono*:
I $\subseteq$ *J* $\Longrightarrow$ *Predict$_F$ k* $\mathcal{G}$ *I* $\subseteq$ *Predict$_F$ k* $\mathcal{G}$ *J*

unfolding $\mathit{Predict}_F$*-def bin-def* **by** *blast*

lemma $\mathit{Complete}_F$*-sub-mono*:
$I \subseteq J \Longrightarrow \mathit{Complete}_F\ k\ I \subseteq \mathit{Complete}_F\ k\ J$
unfolding $\mathit{Complete}_F$*-def bin-def* **by** *blast*

lemma Earley_F*-bin-step-sub-mono*:
$I \subseteq J \Longrightarrow \mathit{Earley}_F\text{-}\mathit{bin}\text{-}\mathit{step}\ k\ \mathcal{G}\ \omega\ I \subseteq \mathit{Earley}_F\text{-}\mathit{bin}\text{-}\mathit{step}\ k\ \mathcal{G}\ \omega\ J$
unfolding Earley_F*-bin-step-def* **using** Scan_F*-sub-mono* $\mathit{Predict}_F$*-sub-mono* $\mathit{Complete}_F$*-sub-mono* **by** (*metis sup.mono*)

lemma *funpower-sub-mono*:
$I \subseteq J \Longrightarrow \mathit{funpower}\ (\mathit{Earley}_F\text{-}\mathit{bin}\text{-}\mathit{step}\ k\ \mathcal{G}\ \omega)\ n\ I \subseteq \mathit{funpower}\ (\mathit{Earley}_F\text{-}\mathit{bin}\text{-}\mathit{step}\ k\ \mathcal{G}\ \omega)\ n\ J$
by (*induction n*) (*auto simp*: Earley_F*-bin-step-sub-mono*)

lemma Earley_F*-bin-sub-mono*:
$I \subseteq J \Longrightarrow \mathit{Earley}_F\text{-}\mathit{bin}\ k\ \mathcal{G}\ \omega\ I \subseteq \mathit{Earley}_F\text{-}\mathit{bin}\ k\ \mathcal{G}\ \omega\ J$
proof *standard*
fix x
assume $I \subseteq J$ $x \in \mathit{Earley}_F\text{-}\mathit{bin}\ k\ \mathcal{G}\ \omega\ I$
then obtain n **where** $x \in \mathit{funpower}\ (\mathit{Earley}_F\text{-}\mathit{bin}\text{-}\mathit{step}\ k\ \mathcal{G}\ \omega)\ n\ I$
unfolding Earley_F*-bin-def limit-def natUnion-def* **by** *blast*
hence $x \in \mathit{funpower}\ (\mathit{Earley}_F\text{-}\mathit{bin}\text{-}\mathit{step}\ k\ \mathcal{G}\ \omega)\ n\ J$
using $\langle I \subseteq J \rangle$ *funpower-sub-mono* **by** *blast*
thus $x \in \mathit{Earley}_F\text{-}\mathit{bin}\ k\ \mathcal{G}\ \omega\ J$
unfolding Earley_F*-bin-def limit-def natUnion-def* **by** *blast*
qed

lemma Scan_F*-*Earley_F*-bin-step-mono*:
$\mathit{Scan}_F\ k\ \omega\ I \subseteq \mathit{Earley}_F\text{-}\mathit{bin}\text{-}\mathit{step}\ k\ \mathcal{G}\ \omega\ I$
using Earley_F*-bin-step-def* **by** *blast*

lemma $\mathit{Predict}_F$*-*Earley_F*-bin-step-mono*:
$\mathit{Predict}_F\ k\ \mathcal{G}\ I \subseteq \mathit{Earley}_F\text{-}\mathit{bin}\text{-}\mathit{step}\ k\ \mathcal{G}\ \omega\ I$
using Earley_F*-bin-step-def* **by** *blast*

lemma $\mathit{Complete}_F$*-*Earley_F*-bin-step-mono*:
$\mathit{Complete}_F\ k\ I \subseteq \mathit{Earley}_F\text{-}\mathit{bin}\text{-}\mathit{step}\ k\ \mathcal{G}\ \omega\ I$
using Earley_F*-bin-step-def* **by** *blast*

lemma Earley_F*-bin-step-*Earley_F*-bin-mono*:
$\mathit{Earley}_F\text{-}\mathit{bin}\text{-}\mathit{step}\ k\ \mathcal{G}\ \omega\ I \subseteq \mathit{Earley}_F\text{-}\mathit{bin}\ k\ \mathcal{G}\ \omega\ I$
proof −
have $\mathit{Earley}_F\text{-}\mathit{bin}\text{-}\mathit{step}\ k\ \mathcal{G}\ \omega\ I \subseteq \mathit{funpower}\ (\mathit{Earley}_F\text{-}\mathit{bin}\text{-}\mathit{step}\ k\ \mathcal{G}\ \omega)\ 1\ I$
by *simp*
thus *?thesis*
by (*metis* Earley_F*-bin-def limit-elem subset-eq*)
qed

lemma $Scan_F$*-Earley*$_F$*-bin-mono*:
$Scan_F\ k\ \omega\ \ I \subseteq Earley_F\text{-}bin\ k\ \mathcal{G}\ \omega\ I$
using $Scan_F$*-Earley*$_F$*-bin-step-mono Earley*$_F$*-bin-step-Earley*$_F$*-bin-mono* **by** *force*

lemma $Predict_F$*-Earley*$_F$*-bin-mono*:
$Predict_F\ k\ \mathcal{G}\ I \subseteq Earley_F\text{-}bin\ k\ \mathcal{G}\ \omega\ I$
using $Predict_F$*-Earley*$_F$*-bin-step-mono Earley*$_F$*-bin-step-Earley*$_F$*-bin-mono* **by** *force*

lemma $Complete_F$*-Earley*$_F$*-bin-mono*:
$Complete_F\ k\ I \subseteq Earley_F\text{-}bin\ k\ \mathcal{G}\ \omega\ I$
using $Complete_F$*-Earley*$_F$*-bin-step-mono Earley*$_F$*-bin-step-Earley*$_F$*-bin-mono* **by** *force*

lemma *Earley*$_F$*-bin-mono*:
$I \subseteq Earley_F\text{-}bin\ k\ \mathcal{G}\ \omega\ I$
using *Earley*$_F$*-bin-step-Earley*$_F$*-bin-mono Earley*$_F$*-bin-step-def* **by** *blast*

lemma $Init_F$*-sub-Earley*$_F$*-bins*:
$Init_F\ \mathcal{G} \subseteq Earley_F\text{-}bins\ n\ \mathcal{G}\ \omega$
apply (*induction n*)
apply *auto*
using *Earley*$_F$*-bin-mono* **by** *blast+*

7.3 Soundness

lemma $Init_F$*-sub-Earley*:
$Init_F\ \mathcal{G} \subseteq Earley\ \mathcal{G}\ \omega$
unfolding $Init_F$*-def init-item-def* **using** *Init* **by** *blast*

lemma $Scan_F$*-sub-Earley*:
assumes $I \subseteq Earley\ \mathcal{G}\ \omega$
shows $Scan_F\ k\ \omega\ I \subseteq Earley\ \mathcal{G}\ \omega$
unfolding $Scan_F$*-def inc-item-def bin-def* **using** *assms Scan*
by (*smt* (*verit*, *ccfv-SIG*) *item.exhaust-sel mem-Collect-eq subsetD subsetI*)

lemma $Predict_F$*-sub-Earley*:
assumes $I \subseteq Earley\ \mathcal{G}\ \omega$
shows $Predict_F\ k\ \mathcal{G}\ I \subseteq Earley\ \mathcal{G}\ \omega$
unfolding $Predict_F$*-def init-item-def bin-def* **using** *assms Predict*
using *item.exhaust-sel* **by** *blast*

lemma $Complete_F$*-sub-Earley*:
assumes $I \subseteq Earley\ \mathcal{G}\ \omega$
shows $Complete_F\ k\ I \subseteq Earley\ \mathcal{G}\ \omega$
unfolding $Complete_F$*-def inc-item-def bin-def* **using** *assms Complete*
by (*smt* (*verit*, *del-insts*) *item.exhaust-sel mem-Collect-eq subset-eq*)

lemma *Earley$_F$-bin-step-sub-Earley*:
  assumes $I \subseteq$ *Earley* $\mathcal{G}$ ω
  shows *Earley$_F$-bin-step* k $\mathcal{G}$ ω $I \subseteq$ *Earley* $\mathcal{G}$ ω
  unfolding *Earley$_F$-bin-step-def* **using** *assms Complete$_F$-sub-Earley Predict$_F$-sub-Earley Scan$_F$-sub-Earley* **by** (*metis le-supI*)

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
  using *assms Earley$_F$-bin-step-sub-Earley* **by** (*metis Earley$_F$-bin-def limit-upperbound*)

lemma *Earley$_F$-bins-sub-Earley*:
  shows *Earley$_F$-bins* n $\mathcal{G}$ $\omega \subseteq$ *Earley* $\mathcal{G}$ ω
  by (*induction n*) (*auto simp*: *Earley$_F$-bin-sub-Earley Init$_F$-sub-Earley*)

lemma *Earley$_F$-sub-Earley*:
  shows *Earley$_F$* $\mathcal{G}$ $\omega \subseteq$ *Earley* $\mathcal{G}$ ω
  by (*simp add*: *Earley$_F$-bins-sub-Earley Earley$_F$-def*)

theorem *soundness-Earley$_F$*:
  assumes *recognizing* (*Earley$_F$* $\mathcal{G}$ ω) $\mathcal{G}$ ω
  shows $\mathcal{G} \vdash [\mathfrak{S}\ \mathcal{G}] \Rightarrow^* \omega$
  using *soundness-Earley Earley$_F$-sub-Earley assms recognizing-def* **by** (*metis subsetD*)

7.4 Completeness

lemma *Earley$_F$-bin-sub-Earley$_F$-bin*:
  assumes *Init$_F$* $\mathcal{G} \subseteq I$
  assumes $\forall k' < k.$ *bin* (*Earley* $\mathcal{G}$ 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 *standard*
  fix x
  assume $*$: $x \in$ *bin* (*Earley* $\mathcal{G}$ ω) k
  hence $x \in$ *Earley* $\mathcal{G}$ ω
    using *bin-def* **by** *blast*
  thus $x \in$ *bin* (*Earley$_F$-bin* k $\mathcal{G}$ ω I) k
    using *assms* $*$
  proof (*induction rule*: *Earley.induct*)
    case (*Init r*)
    thus *?case*
      unfolding *Init$_F$-def init-item-def bin-def* **using** *Earley$_F$-bin-mono* **by** *fast*
  next
    case (*Scan x r b i j a*)
    have $j{+}1 = k$
    using *Scan.prems(4) bin-def* **by** (*metis* (*mono-tags*, *lifting*) *CollectD item.sel(4)*)
    have *prev-symbol* (*Item* r $(b{+}1)$ i $(j{+}1)$) $=$ *Some* $(\omega!(k-1))$
    using *Scan.hyps(1,3,5)* ‹$j{+}1 = k$› **by** (*auto simp*: *next-symbol-def prev-symbol-def*

rhs-item-def split: *if-splits*)
    hence *Item r* (*b*+*1*) *i* (*j*+*1*) ∈ *base* ω (*Earley* $\mathcal{G}$ ω) *k*
      unfolding *base-def* **using** *Scan.prems*(*4*) *bin-def* **by** *fastforce*
    hence *Item r* (*b*+*1*) *i* (*j*+*1*) ∈ *I*
      using *Scan.prems*(*3*) **by** *blast*
    hence *Item r* (*b*+*1*) *i* (*j*+*1*) ∈ $Earley_F$*-bin k* $\mathcal{G}$ ω *I*
      using $Earley_F$*-bin-mono* **by** *blast*
    thus *?case*
      using ‹*j*+*1* = *k*› *bin-def* **by** *fastforce*
  next
    case (*Predict x r b i j r'*)
    have *j* = *k*
      using *Predict.prems*(*4*) *bin-def* **by** (*metis* (*mono-tags*, *lifting*) *CollectD item.sel*(*4*))
    hence *x* ∈ *bin* (*Earley* $\mathcal{G}$ ω) *k*
      using *Predict.hyps*(*1*,*2*) *bin-def* **by** *fastforce*
    hence *x* ∈ *bin* ($Earley_F$*-bin k* $\mathcal{G}$ ω *I*) *k*
      using *Predict.IH Predict.prems*(*1*−*3*) **by** *blast*
    hence *Item r' 0 j j* ∈ $Predict_F$ *k* $\mathcal{G}$ ($Earley_F$*-bin k* $\mathcal{G}$ ω *I*)
      unfolding $Predict_F$*-def init-item-def* **using** *Predict.hyps*(*1*,*3*,*4*) ‹*j* = *k*› **by** *blast*
    hence *Item r' 0 j j* ∈ $Earley_F$*-bin-step k* $\mathcal{G}$ ω ($Earley_F$*-bin k* $\mathcal{G}$ ω *I*)
      using $Predict_F$*-*$Earley_F$*-bin-step-mono* **by** *blast*
    hence *Item r' 0 j j* ∈ $Earley_F$*-bin k* $\mathcal{G}$ ω *I*
      using $Earley_F$*-bin-idem* $Earley_F$*-bin-step-*$Earley_F$*-bin-mono* **by** *blast*
    thus *?case*
      by (*simp add*: ‹*j* = *k*› *bin-def*)
  next
    case (*Complete x* r_x b_x *i j y* r_y b_y *l*)
    have *l* = *k*
      using *Complete.prems*(*4*) *bin-def* **by** (*metis* (*mono-tags*, *lifting*) *CollectD item.sel*(*4*))
    hence *y* ∈ *bin* (*Earley* $\mathcal{G}$ ω) *l*
      using *Complete.hyps*(*3*,*4*) *bin-def* **by** *fastforce*
    hence *0*: *y* ∈ *bin* ($Earley_F$*-bin k* $\mathcal{G}$ ω *I*) *k*
      using *Complete.IH*(*2*) *Complete.prems*(*1*−*3*) ‹*l* = *k*› **by** *blast*
    have *1*: *x* ∈ *bin* ($Earley_F$*-bin k* $\mathcal{G}$ ω *I*) (*start-item y*)
    proof (*cases j* = *k*)
      case *True*
      hence *x* ∈ *bin* (*Earley* $\mathcal{G}$ ω) *k*
        using *Complete.hyps*(*1*,*2*) *bin-def* **by** *fastforce*
      hence *x* ∈ *bin* ($Earley_F$*-bin k* $\mathcal{G}$ ω *I*) *k*
        using *Complete.IH*(*1*) *Complete.prems*(*1*−*3*) **by** *blast*
      thus *?thesis*
        using *Complete.hyps*(*3*) *True* **by** *simp*
    next
      case *False*
      hence *j* < *k*
        using ‹*l* = *k*› *wf-Earley wf-item-def Complete.hyps*(*3*,*4*) **by** *force*

moreover have $x \in bin\ (Earley\ \mathcal{G}\ \omega)\ j$
using *Complete.hyps(1,2) bin-def* **by** *force*
ultimately have $x \in I$
using *Complete.prems(2)* **by** *blast*
hence $x \in bin\ (Earley_F\text{-}bin\ k\ \mathcal{G}\ \omega\ I)\ j$
using *Complete.hyps(1)* $Earley_F$*-bin-mono bin-def* **by** *fastforce*
thus *?thesis*
using *Complete.hyps(3)* **by** *simp*
qed
have $Item\ r_x\ (b_x + 1)\ i\ k \in Complete_F\ k\ (Earley_F\text{-}bin\ k\ \mathcal{G}\ \omega\ I)$
unfolding $Complete_F$*-def inc-item-def* **using** *0 1 Complete.hyps(1,5,6)* **by** *force*
hence $Item\ r_x\ (b_x + 1)\ i\ k \in Earley_F\text{-}bin\text{-}step\ k\ \mathcal{G}\ \omega\ (Earley_F\text{-}bin\ k\ \mathcal{G}\ \omega\ I)$
unfolding $Earley_F$*-bin-step-def* **by** *blast*
hence $Item\ r_x\ (b_x + 1)\ i\ k \in Earley_F\text{-}bin\ k\ \mathcal{G}\ \omega\ I$
using $Earley_F$*-bin-idem* $Earley_F$*-bin-step-*$Earley_F$*-bin-mono* **by** *blast*
thus *?case*
using *bin-def* $\langle l = k \rangle$ **by** *fastforce*
qed
qed

lemma *Earley-base-sub-*$Earley_F$*-bin*:
assumes $Init_F\ \mathcal{G} \subseteq I$
assumes $\forall k' < k.\ bin\ (Earley\ \mathcal{G}\ \omega)\ k' \subseteq I$
assumes $base\ \omega\ (Earley\ \mathcal{G}\ \omega)\ k \subseteq I$
assumes $is\text{-}word\ \mathcal{G}\ \omega$
shows $base\ \omega\ (Earley\ \mathcal{G}\ \omega)\ (k+1) \subseteq bin\ (Earley_F\text{-}bin\ k\ \mathcal{G}\ \omega\ I)\ (k+1)$
proof *standard*
fix x
assume $*$: $x \in base\ \omega\ (Earley\ \mathcal{G}\ \omega)\ (k+1)$
hence $x \in Earley\ \mathcal{G}\ \omega$
using *base-def* **by** *blast*
thus $x \in bin\ (Earley_F\text{-}bin\ k\ \mathcal{G}\ \omega\ I)\ (k+1)$
using *assms* $*$
proof (*induction rule*: *Earley.induct*)
case (*Init r*)
have $k = 0$
using *Init.prems(5)* **unfolding** *base-def* **by** *simp*
hence *False*
using *Init.prems(5)* **unfolding** *base-def* **by** *simp*
thus *?case*
by *blast*
next
case (*Scan x r b i j a*)
have $j = k$
using *Scan.prems(5) base-def* **by** (*metis* (*mono-tags*, *lifting*) *CollectD add-right-cancel item.sel(4)*)
hence $x \in bin\ (Earley_F\text{-}bin\ k\ \mathcal{G}\ \omega\ I)\ k$
using $Earley_F$*-bin-sub-*$Earley_F$*-bin Scan.prems Scan.hyps(1,2) bin-def*

by (*metis* (*mono-tags*, *lifting*) *CollectI item.sel*(*4*) *subsetD*)
hence *Item r* (*b*+*1*) *i* (*j*+*1*) $\in$ $Scan_F$ *k* ω ($Earley_F$*-bin k* $\mathcal{G}$ ω *I*)
unfolding $Scan_F$*-def inc-item-def* **using** *Scan.hyps* ‹*j* = *k*› **by** *force*
hence *Item r* (*b*+*1*) *i* (*j*+*1*) $\in$ $Earley_F$*-bin-step k* $\mathcal{G}$ ω ($Earley_F$*-bin k* $\mathcal{G}$ ω *I*)
using $Scan_F$*-*$Earley_F$*-bin-step-mono* **by** *blast*
hence *Item r* (*b*+*1*) *i* (*j*+*1*) $\in$ $Earley_F$*-bin k* $\mathcal{G}$ ω *I*
using $Earley_F$*-bin-idem* $Earley_F$*-bin-step-*$Earley_F$*-bin-mono* **by** *blast*
thus *?case*
using ‹*j* = *k*› *bin-def* **by** *fastforce*
next
case (*Predict x r b i j r'*)
have *False*
using *Predict.prems*(*5*) **unfolding** *base-def* **by** (*auto simp*: *prev-symbol-def*)
thus *?case*
by *blast*
next
case (*Complete x* r_x b_x *i j y* r_y b_y *l*)
have *l*−*1* < *length* ω
using *Complete.prems*(*5*) *base-def wf-Earley wf-item-def*
by (*metis* (*mono-tags*, *lifting*) *CollectD add.right-neutral add-Suc-right add-diff-cancel-right'* *item.sel*(*4*) *less-eq-Suc-le plus-1-eq-Suc*)
hence ω!(*l*−*1*) $\notin$ *nonterminals* $\mathcal{G}$
using *Complete.prems*(*4*) *is-word-def* **by** *force*
moreover have *lhs-item y* $\in$ *nonterminals* $\mathcal{G}$
using *Complete.hyps*(*3*,*4*) *wf-Earley wf-item-def lhs-item-def lhs-rule-def non-terminals-def*
by (*metis UnCI image-eqI list.set-map*)
moreover have *prev-symbol* (*Item* r_x (b_x+*1*) *i l*) = *next-symbol x*
using *Complete.hyps*(*1*,*6*)
by (*auto simp*: *next-symbol-def prev-symbol-def is-complete-def rhs-item-def split*: *if-splits*)
moreover have *prev-symbol* (*Item* r_x (b_x+*1*) *i l*) = *Some* (ω!(*l*−*1*))
using *Complete.prems*(*5*) *base-def* **by** (*metis* (*mono-tags*, *lifting*) *CollectD item.sel*(*4*))
ultimately have *False*
using *Complete.hyps*(*6*) *Complete.prems*(*4*) **by** *simp*
thus *?case*
by *blast*
qed
qed

lemma $Earley_F$*-bin-k-sub-*$Earley_F$*-bins*:
assumes *is-word* $\mathcal{G}$ ω $k \leq n$
shows *bin* (*Earley* $\mathcal{G}$ ω) *k* $\subseteq$ $Earley_F$*-bins n* $\mathcal{G}$ ω
using *assms*
proof (*induction n arbitrary*: *k*)
case *0*
have *bin* (*Earley* $\mathcal{G}$ ω) *0* $\subseteq$ *bin* ($Earley_F$*-bin 0* $\mathcal{G}$ ω ($Init_F$ $\mathcal{G}$)) *0*
using $Earley_F$*-bin-sub-*$Earley_F$*-bin base-def* **by** *fastforce*

```
  thus ?case
    unfolding bin-def using 0.prems(2) by auto
next
  case (Suc n)
  show ?case
  proof (cases k ≤ n)
    case True
    thus ?thesis
      using Suc Earley_F-bin-mono by force
  next
    case False
    hence k = n+1
      using Suc.prems(2) by force
    have 0: ∀ k' < k. bin (Earley 𝒢 ω) k' ⊆ Earley_F-bins n 𝒢 ω
      using Suc by simp
    moreover have base ω (Earley 𝒢 ω) k ⊆ Earley_F-bins n 𝒢 ω
    proof −
      have ∀ k' < k−1. bin (Earley 𝒢 ω) k' ⊆ Earley_F-bins n 𝒢 ω
        using Suc ‹k = n + 1› by auto
      moreover have base ω (Earley 𝒢 ω) (k−1) ⊆ Earley_F-bins n 𝒢 ω
        using 0 bin-def base-def False ‹k = n+1›
          by (smt (verit) Suc-eq-plus1 diff-Suc-1 linorder-not-less mem-Collect-eq
subsetD subsetI)
    ultimately have base ω (Earley 𝒢 ω) k ⊆ bin (Earley_F-bin n 𝒢 ω (Earley_F-bins
n 𝒢 ω)) k
      using Suc.prems(1,2) Earley-base-sub-Earley_F-bin ‹k = n + 1› Init_F-sub-Earley_F-bins
by (metis add-diff-cancel-right')
      hence base ω (Earley 𝒢 ω) k ⊆ bin (Earley_F-bins n 𝒢 ω) k
        by (metis Earley_F-bins.elims Earley_F-bin-idem)
      thus ?thesis
        using bin-def by blast
    qed
    ultimately have bin (Earley 𝒢 ω) k ⊆ bin (Earley_F-bin k 𝒢 ω (Earley_F-bins
n 𝒢 ω)) k
      using Earley_F-bin-sub-Earley_F-bin Init_F-sub-Earley_F-bins by metis
    thus ?thesis
      using Earley_F-bins.simps(2) ‹k = n + 1› bin-def by auto
  qed
qed

lemma Earley-sub-Earley_F:
  assumes is-word 𝒢 ω
  shows Earley 𝒢 ω ⊆ Earley_F 𝒢 ω
proof −
  have ∀ k ≤ length ω. bin (Earley 𝒢 ω) k ⊆ Earley_F 𝒢 ω
    by (simp add: Earley_F-bin-k-sub-Earley_F-bins Earley_F-def assms)
  thus ?thesis
    using wf-Earley wf-item-def bin-def by blast
qed
```

theorem *completeness-Earley$_F$*:
assumes $\mathcal{G} \vdash [\mathfrak{S}\ \mathcal{G}] \Rightarrow^* \omega$ *is-word* $\mathcal{G}$ ω
shows *recognizing* (*Earley$_F$* $\mathcal{G}$ ω) $\mathcal{G}$ ω
using *assms Earley-sub-Earley$_F$ Earley$_F$-sub-Earley completeness-Earley* **by** (*metis subset-antisym*)

7.5 Correctness

theorem *Earley-eq-Earley$_F$*:
assumes *is-word* $\mathcal{G}$ ω
shows *Earley* $\mathcal{G}$ ω = *Earley$_F$* $\mathcal{G}$ ω
using *Earley-sub-Earley$_F$ Earley$_F$-sub-Earley assms* **by** *blast*

theorem *correctness-Earley$_F$*:
assumes *is-word* $\mathcal{G}$ ω
shows *recognizing* (*Earley$_F$* $\mathcal{G}$ ω) $\mathcal{G}$ $\omega \longleftrightarrow \mathcal{G} \vdash [\mathfrak{S}\ \mathcal{G}] \Rightarrow^* \omega$
using *assms Earley-eq-Earley$_F$ correctness-Earley* **by** *fastforce*

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*
by (*induction xs arbitrary*: *i*) (*auto split*: *if-splits*)

lemma *filter-with-index-P*:
(*x*, *n*) ∈ *set* (*filter-with-index P xs*) ⟹ *P x*
by (*metis filter-with-index'-P filter-with-index-def*)

lemma *filter-with-index'-cong-filter*:
map fst (*filter-with-index' i P xs*) = *filter P xs*
by (*induction xs arbitrary*: *i*) *auto*

lemma *filter-with-index-cong-filter*:
map fst (*filter-with-index P xs*) = *filter P xs*
by (*simp add*: *filter-with-index′-cong-filter filter-with-index-def*)

lemma *size-index-filter-with-index′*:
$(x, n) \in set$ (*filter-with-index′ i P xs*) $\Longrightarrow n \geq i$
by (*induction xs arbitrary*: *i*) (*auto simp*: *Suc-leD split*: *if-splits*)

lemma *index-filter-with-index′-lt-length*:
$(x, n) \in set$ (*filter-with-index′ i P xs*) $\Longrightarrow n{-}i <$ *length xs*
by (*induction xs arbitrary*: *i*)(*auto simp*: *less-Suc-eq-0-disj split*: *if-splits*; *metis Suc-diff-Suc leI*)+

lemma *index-filter-with-index-lt-length*:
$(x, n) \in set$ (*filter-with-index P xs*) $\Longrightarrow n <$ *length xs*
by (*metis filter-with-index-def index-filter-with-index′-lt-length minus-nat.diff-0*)

lemma *filter-with-index′-nth*:
$(x, n) \in set$ (*filter-with-index′ i P xs*) $\Longrightarrow$ *xs* ! $(n{-}i) = x$
proof (*induction xs arbitrary*: *i*)
case (*Cons y xs*)
show *?case*
proof (*cases* $x = y$)
case *True*
thus *?thesis*
using *Cons* **by** (*auto simp*: *nth-Cons′ split*: *if-splits*)
next
case *False*
hence $(x, n) \in set$ (*filter-with-index′* $(i{+}1)$ *P xs*)
using *Cons.prems* **by** (*cases xs*) (*auto split*: *if-splits*)
hence $n \geq i + 1$ *xs* ! $(n - i - 1) = x$
by (*auto simp*: *size-index-filter-with-index′ Cons.IH*)
thus *?thesis*
by *simp*
qed
qed *simp*

lemma *filter-with-index-nth*:
$(x, n) \in set$ (*filter-with-index P xs*) $\Longrightarrow$ *xs* ! $n = x$
by (*metis diff-zero filter-with-index′-nth filter-with-index-def*)

lemma *filter-with-index-nonempty*:
$x \in set\ xs \Longrightarrow P\ x \Longrightarrow$ *filter-with-index P xs* $\neq$ []
by (*metis filter-empty-conv filter-with-index-cong-filter list.map*(*1*))

lemma *filter-with-index′-Ex-first*:
($\exists x\ i\ xs'$. *filter-with-index′ n P xs* $= (x, i)\#xs'$) $\longleftrightarrow$ ($\exists x \in set\ xs.\ P\ x$)
by (*induction xs arbitrary*: *n*) *auto*

lemma *filter-with-index-Ex-first*:
$(\exists\, x\ i\ xs'.\ \textit{filter-with-index}\ P\ xs = (x,\ i)\#xs') \longleftrightarrow (\exists\, x \in \textit{set}\ xs.\ P\ x)$
using *filter-with-index'-Ex-first filter-with-index-def* **by** *metis*

8.2 Definitions

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

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

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

definition *pointers* :: *'a bin* ⇒ *pointer list* **where**
pointers b ≡ *map snd b*

definition *bins-eq-items* :: *'a bins* ⇒ *'a bins* ⇒ *bool* **where**
bins-eq-items bs0 bs1 ≡ *map items bs0* = *map items bs1*

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

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

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

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

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

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

definition *ε-free* :: *'a cfg* ⇒ *bool* **where**
$\varepsilon\textit{-free}\ \mathcal{G} = (\forall\, r \in \textit{set}\ (\mathfrak{R}\ \mathcal{G}).\ \textit{rhs-rule}\ r \neq [])$

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

definition $\textit{Init}_L$ :: *'a cfg* ⇒ *'a list* ⇒ *'a bins* **where**

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

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

definition $Predict_L :: nat \Rightarrow 'a\ cfg \Rightarrow 'a \Rightarrow ('a\ item \times pointer)\ list$ **where**
  $Predict_L\ k\ \mathcal{G}\ X \equiv$
  $let\ rs = filter\ (\lambda r.\ lhs\text{-}rule\ r = X)\ (\mathfrak{R}\ \mathcal{G})\ in$
  $map\ (\lambda r.\ (init\text{-}item\ r\ k,\ Null))\ rs$

definition $Complete_L :: nat \Rightarrow 'a\ item \Rightarrow 'a\ bins \Rightarrow nat \Rightarrow ('a\ item \times pointer)$ $list$ **where**
  $Complete_L\ k\ y\ bs\ red \equiv$
  $let\ orig = bs\ !\ (start\text{-}item\ y)\ in$
  $let\ is = filter\text{-}with\text{-}index\ (\lambda x.\ next\text{-}symbol\ x = Some\ (lhs\text{-}item\ y))\ (items\ orig)$
in
  $map\ (\lambda(x,\ pre).\ (inc\text{-}item\ x\ k,\ PreRed\ (start\text{-}item\ y,\ pre,\ red)\ []))\ is$

fun $upd\text{-}bin :: 'a\ item \times pointer \Rightarrow 'a\ bin \Rightarrow 'a\ bin$ **where**
  $upd\text{-}bin\ e'\ [] = [e']$
$|\ upd\text{-}bin\ e'\ (e\#es) = ($
  $case\ (e',\ e)\ of$
    $((x,\ PreRed\ px\ xs),\ (y,\ PreRed\ py\ ys)) \Rightarrow$
      $if\ x = y\ then\ (x,\ PreRed\ py\ (px\#xs@ys))\ \#\ es$
      $else\ e\ \#\ upd\text{-}bin\ e'\ es$
    $|\ \text{-} \Rightarrow$
      $if\ fst\ e' = fst\ e\ then\ e\ \#\ es$
      $else\ e\ \#\ upd\text{-}bin\ e'\ es)$

fun $upds\text{-}bin :: ('a\ item \times pointer)\ list \Rightarrow 'a\ bin \Rightarrow 'a\ bin$ **where**
  $upds\text{-}bin\ []\ b = b$
$|\ upds\text{-}bin\ (e\#es)\ b = upds\text{-}bin\ es\ (upd\text{-}bin\ e\ b)$

definition $upd\text{-}bins :: 'a\ bins \Rightarrow nat \Rightarrow ('a\ item \times pointer)\ list \Rightarrow 'a\ bins$ **where**
  $upd\text{-}bins\ bs\ k\ es \equiv bs[k := upds\text{-}bin\ es\ (bs!k)]$

partial-function $(tailrec)\ Earley_L\text{-}bin' :: nat \Rightarrow 'a\ cfg \Rightarrow 'a\ list \Rightarrow 'a\ bins \Rightarrow nat$ $\Rightarrow 'a\ bins$ **where**
  $Earley_L\text{-}bin'\ k\ \mathcal{G}\ \omega\ bs\ i = ($
  $if\ i \geq length\ (items\ (bs\ !\ k))\ then\ bs$

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

declare $Earley_L$*-bin'.simps*[*code*]

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

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

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

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

8.3 Epsilon productions

lemma ε*-free-impl-non-empty-word-deriv*:
ε*-free* $\mathcal{G} \Longrightarrow a \neq [] \Longrightarrow \neg$ *Derivation* $\mathcal{G}\ a\ D$ []
proof (*induction length D arbitrary*: *a D rule*: *nat-less-induct*)
case *1*
show *?case*
proof (*rule ccontr*)
assume *assm*: $\neg\ \neg$ *Derivation* $\mathcal{G}\ a\ D$ []
show *False*
proof (*cases* D = [])
case *True*
then show *?thesis*
using *1.prems*(*2*) *assm* **by** *auto*
next
case *False*
then obtain $d\ D'\ \alpha$ **where** $*$:
$D = d\ \#\ D'$ *Derives1* $\mathcal{G}\ a$ (*fst d*) (*snd d*) α *Derivation* $\mathcal{G}\ \alpha\ D'$ [] *snd d* $\in$ *set* $(\mathfrak{R}\ \mathcal{G})$
using *list.exhaust assm Derives1-def* **by** (*metis Derivation.simps*(*2*))
show *?thesis*
proof *cases*

assume $\alpha = []$
thus *?thesis*
using $*(2,4)$ *Derives1-split ε-free-def rhs-rule-def 1.prems(1)* **by** (*metis append-is-Nil-conv*)
next
assume $\neg\ \alpha = []$
thus *?thesis*
using $*(1,3)$ *1.hyps 1.prems(1)* **by** *auto*
qed
qed
qed
qed

lemma *ε-free-impl-nonempty-derives*:
ε-free $\mathcal{G} \Longrightarrow$ *nonempty-derives* $\mathcal{G}$
using *ε-free-impl-non-empty-word-deriv derives-implies-Derivation nonempty-derives-def*
by (*metis not-Cons-self2*)

lemma *nonempty-derives-impl-ε-free*:
assumes *nonempty-derives* $\mathcal{G}$
shows *ε-free* $\mathcal{G}$
proof (*rule ccontr*)
assume $\neg$ *ε-free* $\mathcal{G}$
then obtain $N\ \alpha$ **where** $*$: $(N, \alpha) \in$ *set* $(\mathfrak{R}\ \mathcal{G})$ *rhs-rule* $(N, \alpha) = []$
unfolding *ε-free-def* **by** *auto*
hence $\mathcal{G} \vdash [N] \Rightarrow []$
unfolding *derives1-def rhs-rule-def* **by** *auto*
hence $\mathcal{G} \vdash [N] \Rightarrow^* []$
by *auto*
thus *False*
using *assms(1) nonempty-derives-def* **by** *fast*
qed

lemma *nonempty-derives-iff-ε-free*:
shows *nonempty-derives* $\mathcal{G} \longleftrightarrow$ *ε-free* $\mathcal{G}$
using *ε-free-impl-nonempty-derives nonempty-derives-impl-ε-free* **by** *blast*

8.4 Bin lemmas

lemma *length-upd-bins[simp]*:
length (*upd-bins bs k es*) = *length bs*
unfolding *upd-bins-def* **by** *simp*

lemma *length-upd-bin*:
length (*upd-bin e b*) $\geq$ *length b*
by (*induction e b rule*: *upd-bin.induct*) (*auto split*: *pointer.splits*)

lemma *length-upds-bin*:
length (*upds-bin es b*) $\geq$ *length b*

by (*induction es arbitrary*: *b*) (*auto*, *meson le-trans length-upd-bin*)

lemma *length-nth-upd-bin-bins*:
length (*upd-bins bs k es* ! *n*) $\geq$ *length* (*bs* ! *n*)
unfolding *upd-bins-def* **using** *length-upds-bin*
by (*metis linorder-not-le list-update-beyond nth-list-update-eq nth-list-update-neq order-refl*)

lemma *nth-idem-upd-bins*:
$k \neq n \Longrightarrow$ *upd-bins bs k es* ! *n* = *bs* ! *n*
unfolding *upd-bins-def* **by** *simp*

lemma *items-nth-idem-upd-bin*:
$n <$ *length b* $\Longrightarrow$ *items* (*upd-bin e b*) ! *n* = *items b* ! *n*
by (*induction b arbitrary*: *e n*) (*auto simp*: *items-def less-Suc-eq-0-disj split*!: *pointer.split*)

lemma *items-nth-idem-upds-bin*:
$n <$ *length b* $\Longrightarrow$ *items* (*upds-bin es b*) ! *n* = *items b* ! *n*
by (*induction es arbitrary*: *b*)
(*auto*, *metis items-nth-idem-upd-bin length-upd-bin order.strict-trans2*)

lemma *items-nth-idem-upd-bins*:
$n <$ *length* (*bs* ! *k*) $\Longrightarrow$ *items* (*upd-bins bs k es* ! *k*) ! *n* = *items* (*bs* ! *k*) ! *n*
unfolding *upd-bins-def* **using** *items-nth-idem-upds-bin*
by (*metis linorder-not-less list-update-beyond nth-list-update-eq*)

lemma *bin-upto-eq-set-items*:
$i \geq$ *length b* $\Longrightarrow$ *bin-upto b i* = *set* (*items b*)
by (*auto simp*: *bin-upto-def items-def*, *metis fst-conv in-set-conv-nth nth-map order.strict-trans2*)

lemma *bins-upto-empty*:
bins-upto bs 0 0 = {}
unfolding *bins-upto-def bin-upto-def* **by** *simp*

lemma *set-items-upd-bin*:
set (*items* (*upd-bin e b*)) = *set* (*items b*) $\cup$ {*fst e*}
proof (*induction b arbitrary*: *e*)
case (*Cons b bs*)
show *?case*
proof (*cases* $\exists$ *x xp xs y yp ys*. *e* = (*x*, *PreRed xp xs*) $\wedge$ *b* = (*y*, *PreRed yp ys*))
case *True*
then obtain *x xp xs y yp ys* **where** *e* = (*x*, *PreRed xp xs*) *b* = (*y*, *PreRed yp ys*)
by *blast*
thus *?thesis*
using *Cons.IH* **by** (*auto simp*: *items-def*)
next

case *False*
then show *?thesis*
proof *cases*
assume $*$: *fst e* = *fst b*
hence *upd-bin e* (*b* # *bs*) = *b* # *bs*
using *False* **by** (*auto split*: *pointer.splits prod.split*)
thus *?thesis*
using $*$ **by** (*auto simp*: *items-def*)
next
assume $*$: ¬ *fst e* = *fst b*
hence *upd-bin e* (*b* # *bs*) = *b* # *upd-bin e bs*
using *False* **by** (*auto split*: *pointer.splits prod.split*)
thus *?thesis*
using $*$ *Cons.IH* **by** (*auto simp*: *items-def*)
qed
qed
qed (*auto simp*: *items-def*)

lemma *set-items-upds-bin*:
set (*items* (*upds-bin es b*)) = *set* (*items b*) ∪ *set* (*items es*)
apply (*induction es arbitrary*: *b*)
apply (*auto simp*: *items-def*)
by (*metis Domain.DomainI Domain-fst Un-insert-right fst-conv insert-iff items-def list.set-map set-items-upd-bin sup-bot.right-neutral*)+

lemma *bins-upd-bins*:
assumes $k < length\ bs$
shows *bins* (*upd-bins bs k es*) = *bins bs* ∪ *set* (*items es*)
proof −
let *?bs* = *upd-bins bs k es*
have *bins* (*upd-bins bs k es*) = $\bigcup$ {*set* (*items* (*?bs* ! *k*)) |*k*. $k < length$ *?bs*}
unfolding *bins-def* **by** *blast*
also have ... = $\bigcup$ {*set* (*items* (*bs* ! *l*)) |*l*. $l < length\ bs \wedge l \neq k$} ∪ *set* (*items* (*?bs* ! *k*))
unfolding *upd-bins-def* **using** *assms* **by** (*auto*, *metis nth-list-update*)
also have ... = $\bigcup$ {*set* (*items* (*bs* ! *l*)) |*l*. $l < length\ bs \wedge l \neq k$} ∪ *set* (*items* (*bs* ! *k*)) ∪ *set* (*items es*)
using *set-items-upds-bin*[*of es bs*!*k*] **by** (*simp add*: *assms upd-bins-def sup-assoc*)
also have ... = $\bigcup$ {*set* (*items* (*bs* ! *k*)) |*k*. $k < length\ bs$} ∪ *set* (*items es*)
using *assms* **by** *blast*
also have ... = *bins bs* ∪ *set* (*items es*)
unfolding *bins-def* **by** *blast*
finally show *?thesis* **.**
qed

lemma *kth-bin-sub-bins*:
$k < length\ bs \Longrightarrow$ *set* (*items* (*bs* ! *k*)) ⊆ *bins bs*
unfolding *bins-def bins-upto-def bin-upto-def* **by** *blast*+

lemma *bin-upto-Cons-0*:
  bin-upto (*e*#*es*) *0* = {}
  by (*auto simp*: *bin-upto-def*)

lemma *bin-upto-Cons*:
  assumes *0* < *n*
  shows *bin-upto* (*e*#*es*) *n* = { *fst e* } ∪ *bin-upto es* (*n*−*1*)
proof −
  have *bin-upto* (*e*#*es*) *n* = { *items* (*e*#*es*) ! *j* | *j*. *j* < *n* ∧ *j* < *length* (*items* (*e*#*es*)) }
    unfolding *bin-upto-def* **by** *blast*
  also have ... = { *fst e* } ∪ { *items es* ! *j* | *j*. *j* < (*n*−*1*) ∧ *j* < *length* (*items es*) }
    using *assms* **by** (*cases n*) (*auto simp*: *items-def nth-Cons′*, *metis One-nat-def Zero-not-Suc diff-Suc-1 not-less-eq nth-map*)
  also have ... = { *fst e* } ∪ *bin-upto es* (*n*−*1*)
    unfolding *bin-upto-def* **by** *blast*
  finally show *?thesis* **.**
qed

lemma *bin-upto-nth-idem-upd-bin*:
  n < *length b* ⟹ *bin-upto* (*upd-bin e b*) *n* = *bin-upto b n*
proof (*induction b arbitrary*: *e n*)
  case (*Cons b bs*)
  show *?case*
  proof (*cases* ∃ *x xp xs y yp ys*. *e* = (*x*, *PreRed xp xs*) ∧ *b* = (*y*, *PreRed yp ys*))
    case *True*
    then obtain *x xp xs y yp ys* **where** *e* = (*x*, *PreRed xp xs*) *b* = (*y*, *PreRed yp ys*)
      by *blast*
    thus *?thesis*
      using *Cons bin-upto-Cons-0*
      by (*cases n*) (*auto simp*: *items-def bin-upto-Cons*, *blast*+)
  next
    case *False*
    then show *?thesis*
    proof *cases*
      assume ∗: *fst e* = *fst b*
      hence *upd-bin e* (*b* # *bs*) = *b* # *bs*
        using *False* **by** (*auto split*: *pointer.splits prod.split*)
      thus *?thesis*
        using ∗ **by** (*auto simp*: *items-def*)
    next
      assume ∗: ¬ *fst e* = *fst b*
      hence *upd-bin e* (*b* # *bs*) = *b* # *upd-bin e bs*
        using *False* **by** (*auto split*: *pointer.splits prod.split*)
      thus *?thesis*
        using ∗ *Cons*
        by (*cases n*) (*auto simp*: *items-def bin-upto-Cons-0 bin-upto-Cons*)
    qed

qed
qed (*auto simp*: *items-def*)

lemma *bin-upto-nth-idem-upds-bin*:
$n < length\ b \Longrightarrow bin\text{-}upto\ (upds\text{-}bin\ es\ b)\ n = bin\text{-}upto\ b\ n$
using *bin-upto-nth-idem-upd-bin length-upd-bin*
apply (*induction es arbitrary*: *b*)
apply *auto*
using *order.strict-trans2 order.strict-trans1* **by** *blast+*

lemma *bins-upto-kth-nth-idem*:
assumes $l < length\ bs$ $k \leq l$ $n < length\ (bs\ !\ k)$
shows $bins\text{-}upto\ (upd\text{-}bins\ bs\ l\ es)\ k\ n = bins\text{-}upto\ bs\ k\ n$
proof −
let $?bs = upd\text{-}bins\ bs\ l\ es$
have $bins\text{-}upto\ ?bs\ k\ n = \bigcup\ \{set\ (items\ (?bs\ !\ l))\ |l.\ l < k\} \cup bin\text{-}upto\ (?bs\ !\ k)$
n
unfolding *bins-upto-def* **by** *blast*
also have ... $= \bigcup\ \{set\ (items\ (bs\ !\ l))\ |l.\ l < k\} \cup bin\text{-}upto\ (?bs\ !\ k)\ n$
unfolding *upd-bins-def* **using** *assms*(*1*,*2*) **by** *auto*
also have ... $= \bigcup\ \{set\ (items\ (bs\ !\ l))\ |l.\ l < k\} \cup bin\text{-}upto\ (bs\ !\ k)\ n$
unfolding *upd-bins-def* **using** *assms*(*1*,*3*) *bin-upto-nth-idem-upds-bin*
by (*metis* (*no-types*, *lifting*) *nth-list-update*)
also have ... $= bins\text{-}upto\ bs\ k\ n$
unfolding *bins-upto-def* **by** *blast*
finally show *?thesis* **.**
qed

lemma *bins-upto-sub-bins*:
$k < length\ bs \Longrightarrow bins\text{-}upto\ bs\ k\ n \subseteq bins\ bs$
unfolding *bins-def bins-upto-def bin-upto-def* **using** *less-trans* **by** (*auto*, *blast*)

lemma *bins-upto-Suc-Un*:
$n < length\ (bs\ !\ k) \Longrightarrow bins\text{-}upto\ bs\ k\ (n+1) = bins\text{-}upto\ bs\ k\ n \cup \{\ items\ (bs\ !$
$k)\ !\ n\ \}$
unfolding *bins-upto-def bin-upto-def* **using** *less-Suc-eq* **by** (*auto simp*: *items-def*,
metis nth-map)

lemma *bins-bin-exists*:
$x \in bins\ bs \Longrightarrow \exists\, k < length\ bs.\ x \in set\ (items\ (bs\ !\ k))$
unfolding *bins-def* **by** *blast*

lemma *distinct-upd-bin*:
$distinct\ (items\ b) \Longrightarrow distinct\ (items\ (upd\text{-}bin\ e\ b))$
proof (*induction b arbitrary*: *e*)
case (*Cons b bs*)
show *?case*
proof (*cases* $\exists\, x\ xp\ xs\ y\ yp\ ys.\ e = (x,\ PreRed\ xp\ xs) \wedge b = (y,\ PreRed\ yp\ ys)$)
case *True*

then obtain *x xp xs y yp ys* **where** $e = (x,\ PreRed\ xp\ xs)$ $b = (y,\ PreRed\ yp\ ys)$
by *blast*
thus *?thesis*
using *Cons*
apply (*auto simp*: *items-def split*: *prod.split*)
by (*metis Domain.DomainI Domain-fst UnE empty-iff fst-conv insert-iff items-def list.set-map set-items-upd-bin*)
next
case *False*
then show *?thesis*
proof *cases*
assume $*$: $fst\ e = fst\ b$
hence $upd\text{-}bin\ e\ (b\ \#\ bs) = b\ \#\ bs$
using *False* **by** (*auto split*: *pointer.splits prod.split*)
thus *?thesis*
using $*$ *Cons.prems* **by** (*auto simp*: *items-def*)
next
assume $*$: $\neg\ fst\ e = fst\ b$
hence $upd\text{-}bin\ e\ (b\ \#\ bs) = b\ \#\ upd\text{-}bin\ e\ bs$
using *False* **by** (*auto split*: *pointer.splits prod.split*)
moreover have *distinct* (*items* (*upd-bin e bs*))
using *Cons* **by** (*auto simp*: *items-def*)
ultimately show *?thesis*
using $*$ *Cons.prems set-items-upd-bin*
by (*metis Un-insert-right distinct.simps*(*2*) *insertE items-def list.simps*(*9*) *sup-bot-right*)
qed
qed
qed (*auto simp*: *items-def*)

lemma *distinct-upds-bin*:
distinct (*items b*) $\Longrightarrow$ *distinct* (*items* (*upds-bin es b*))
by (*induction es arbitrary*: *b*) (*auto simp add*: *distinct-upd-bin*)

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

lemma *wf-bin-upd-bin*:
assumes $wf\text{-}bin\ \mathcal{G}\ \omega\ k\ b$ $wf\text{-}item\ \mathcal{G}\ \omega\ (fst\ e) \wedge end\text{-}item\ (fst\ e) = k$
shows $wf\text{-}bin\ \mathcal{G}\ \omega\ k\ (upd\text{-}bin\ e\ b)$
using *assms*
proof (*induction b arbitrary*: *e*)
case (*Cons b bs*)
show *?case*
proof (*cases* $\exists\, x\ xp\ xs\ y\ yp\ ys.\ e = (x,\ PreRed\ xp\ xs) \wedge b = (y,\ PreRed\ yp\ ys)$)
case *True*

then obtain *x xp xs y yp ys* **where** *e* = (*x*, *PreRed xp xs*) *b* = (*y*, *PreRed yp ys*)
    by *blast*
  thus *?thesis*
    using *Cons distinct-upd-bin wf-bin-def wf-bin-items-def set-items-upd-bin*
    by (*smt* (*verit*, *best*) *Un-insert-right insertE sup-bot.right-neutral*)
next
  case *False*
  then show *?thesis*
  proof *cases*
    assume ∗: *fst e* = *fst b*
    hence *upd-bin e* (*b* # *bs*) = *b* # *bs*
      using *False* **by** (*auto split*: *pointer.splits prod.split*)
    thus *?thesis*
      using ∗ *Cons.prems* **by** (*auto simp*: *items-def*)
  next
    assume ∗: ¬ *fst e* = *fst b*
    hence *upd-bin e* (*b* # *bs*) = *b* # *upd-bin e bs*
      using *False* **by** (*auto split*: *pointer.splits prod.split*)
    thus *?thesis*
    using ∗ *Cons.prems set-items-upd-bin distinct-upd-bin wf-bin-def wf-bin-items-def*
      by (*smt* (*verit*, *best*) *Un-insert-right insertE sup-bot-right*)
  qed
qed
qed (*auto simp*: *items-def wf-bin-def wf-bin-items-def*)

lemma *wf-upd-bins-bin*:
  assumes *wf-bin* $\mathcal{G}$ *ω k b*
  assumes ∀ *x* ∈ *set* (*items es*). *wf-item* $\mathcal{G}$ *ω x* ∧ *end-item x* = *k*
  shows *wf-bin* $\mathcal{G}$ *ω k* (*upds-bin es b*)
  using *assms* **by** (*induction es arbitrary*: *b*) (*auto simp*: *wf-bin-upd-bin items-def*)

lemma *wf-bins-upd-bins*:
  assumes *wf-bins* $\mathcal{G}$ *ω bs*
  assumes ∀ *x* ∈ *set* (*items es*). *wf-item* $\mathcal{G}$ *ω x* ∧ *end-item x* = *k*
  shows *wf-bins* $\mathcal{G}$ *ω* (*upd-bins bs k es*)
  unfolding *upd-bins-def* **using** *assms wf-upd-bins-bin wf-bins-def*
  by (*metis length-list-update nth-list-update-eq nth-list-update-neq*)

lemma *wf-bins-impl-wf-items*:
  wf-bins $\mathcal{G}$ *ω bs* ⟹ ∀ *x* ∈ (*bins bs*). *wf-item* $\mathcal{G}$ *ω x*
  unfolding *wf-bins-def wf-bin-def wf-bin-items-def bins-def* **by** *auto*

lemma *upds-bin-eq-items*:
  set (*items es*) ⊆ *set* (*items b*) ⟹ *set* (*items* (*upds-bin es b*)) = *set* (*items b*)
  apply (*induction es arbitrary*: *b*)
   apply (*auto simp*: *set-items-upd-bin set-items-upds-bin*)
   apply (*simp add*: *items-def*)
  by (*metis Un-upper2 upds-bin.simps*(*2*) *in-mono set-items-upds-bin sup.orderE*)

lemma *bin-eq-items-upd-bin*:
  fst e $\in$ *set* (*items b*) $\Longrightarrow$ *items* (*upd-bin e b*) = *items b*
proof (*induction b arbitrary*: *e*)
  case (*Cons b bs*)
  show *?case*
  proof (*cases* $\exists$ *x xp xs y yp ys*. *e* = (*x*, *PreRed xp xs*) $\wedge$ *b* = (*y*, *PreRed yp ys*))
    case *True*
    then obtain *x xp xs y yp ys* **where** *e* = (*x*, *PreRed xp xs*) *b* = (*y*, *PreRed yp ys*)
      by *blast*
    thus *?thesis*
      using *Cons* **by** (*auto simp*: *items-def*, *metis fst-conv image-eqI*)
  next
    case *False*
    then show *?thesis*
    proof *cases*
      assume *: *fst e* = *fst b*
      hence *upd-bin e* (*b* # *bs*) = *b* # *bs*
        using *False* **by** (*auto split*: *pointer.splits prod.split*)
      thus *?thesis*
        using * *Cons.prems* **by** (*auto simp*: *items-def*)
    next
      assume *: $\neg$ *fst e* = *fst b*
      hence *upd-bin e* (*b* # *bs*) = *b* # *upd-bin e bs*
        using *False* **by** (*auto split*: *pointer.splits prod.split*)
      thus *?thesis*
        using * *Cons* **by** (*auto simp*: *items-def*)
    qed
  qed
qed (*auto simp*: *items-def*)

lemma *bin-eq-items-upds-bin*:
  assumes *set* (*items es*) $\subseteq$ *set* (*items b*)
  shows *items* (*upds-bin es b*) = *items b*
  using *assms*
proof (*induction es arbitrary*: *b*)
  case (*Cons e es*)
  have *items* (*upds-bin es* (*upd-bin e b*)) = *items* (*upd-bin e b*)
    using *Cons upds-bin-eq-items set-items-upd-bin set-items-upds-bin*
    by (*metis Un-upper2 upds-bin.simps*(*2*) *sup.coboundedI1*)
  moreover have *items* (*upd-bin e b*) = *items b*
   by (*metis Cons.prems bin-eq-items-upd-bin items-def list.set-intros*(*1*) *list.simps*(*9*) *subset-code*(*1*))
  ultimately show *?case*
    by *simp*
qed (*auto simp*: *items-def*)

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

assumes *set* (*items es*) $\subseteq$ *set* (*items* (*bs*!*k*))
shows *bins-eq-items* (*upd-bins bs k es*) *bs*
unfolding *upd-bins-def* **using** *assms bin-eq-items-upds-bin bins-eq-items-def*
by (*metis list-update-id map-update*)

lemma *bins-eq-items-imp-eq-bins*:
bins-eq-items bs bs' $\Longrightarrow$ *bins bs* = *bins bs'*
unfolding *bins-eq-items-def bins-def items-def*
by (*metis* (*no-types*, *lifting*) *length-map nth-map*)

lemma *bin-eq-items-dist-upd-bin-bin*:
assumes *items a* = *items b*
shows *items* (*upd-bin e a*) = *items* (*upd-bin e b*)
using *assms*
proof (*induction a arbitrary*: *e b*)
case (*Cons a as*)
obtain *b' bs* **where** *bs*: *b* = *b'* # *bs fst a* = *fst b' items as* = *items bs*
using *Cons.prems* **by** (*auto simp*: *items-def*)
show *?case*
proof (*cases* $\exists$ *x xp xs y yp ys*. *e* = (*x*, *PreRed xp xs*) $\wedge$ *a* = (*y*, *PreRed yp ys*))
case *True*
then obtain *x xp xs y yp ys* **where** #: *e* = (*x*, *PreRed xp xs*) *a* = (*y*, *PreRed*
yp ys)
by *blast*
show *?thesis*
proof *cases*
assume $*$: *x* = *y*
hence *items* (*upd-bin e* (*a* # *as*)) = *x* # *items as*
using # **by** (*auto simp*: *items-def*)
moreover have *items* (*upd-bin e* (*b'* # *bs*)) = *x* # *items bs*
using *bs* # $*$ **by** (*auto simp*: *items-def split*: *pointer.splits prod.splits*)
ultimately show *?thesis*
using *bs* **by** *simp*
next
assume $*$: $\neg$ *x* = *y*
hence *items* (*upd-bin e* (*a* # *as*)) = *y* # *items* (*upd-bin e as*)
using # **by** (*auto simp*: *items-def*)
moreover have *items* (*upd-bin e* (*b'* # *bs*)) = *y* # *items* (*upd-bin e bs*)
using *bs* # $*$ **by** (*auto simp*: *items-def split*: *pointer.splits prod.splits*)
ultimately show *?thesis*
using *bs Cons.IH* **by** *simp*
qed
next
case *False*
then show *?thesis*
proof *cases*
assume $*$: *fst e* = *fst a*
hence *items* (*upd-bin e* (*a* # *as*)) = *fst a* # *items as*
using *False* **by** (*auto simp*: *items-def split*: *pointer.splits prod.splits*)

moreover have *items* (*upd-bin e* (b' # *bs*)) = *fst* b' # *items bs*
using *bs False* ∗ **by** (*auto simp*: *items-def split*: *pointer.splits prod.splits*)
ultimately show *?thesis*
using *bs* **by** *simp*
next
assume ∗: ¬ *fst e* = *fst a*
hence *items* (*upd-bin e* (*a* # *as*)) = *fst a* # *items* (*upd-bin e as*)
using *False* **by** (*auto simp*: *items-def split*: *pointer.splits prod.splits*)
moreover have *items* (*upd-bin e* (b' # *bs*)) = *fst* b' # *items* (*upd-bin e bs*)
using *bs False* ∗ **by** (*auto simp*: *items-def split*: *pointer.splits prod.splits*)
ultimately show *?thesis*
using *bs Cons* **by** *simp*
qed
qed
qed (*auto simp*: *items-def*)

lemma *bin-eq-items-dist-upds-bin-bin*:
assumes *items a* = *items b*
shows *items* (*upds-bin es a*) = *items* (*upds-bin es b*)
using *assms*
proof (*induction es arbitrary*: *a b*)
case (*Cons e es*)
hence *items* (*upds-bin es* (*upd-bin e a*)) = *items* (*upds-bin es* (*upd-bin e b*))
using *bin-eq-items-dist-upd-bin-bin* **by** *blast*
thus *?case*
by *simp*
qed *simp*

lemma *bin-eq-items-dist-upd-bin-entry*:
assumes *fst e* = *fst* e'
shows *items* (*upd-bin e b*) = *items* (*upd-bin* e' *b*)
using *assms*
proof (*induction b arbitrary*: *e* e')
case (*Cons a as*)
show *?case*
proof (*cases* ∃ *x xp xs y yp ys*. *e* = (*x*, *PreRed xp xs*) ∧ *a* = (*y*, *PreRed yp ys*))
case *True*
then obtain *x xp xs y yp ys* **where** #: *e* = (*x*, *PreRed xp xs*) *a* = (*y*, *PreRed yp ys*)
by *blast*
show *?thesis*
proof *cases*
assume ∗: *x* = *y*
thus *?thesis*
using # *Cons.prems* **by** (*auto simp*: *items-def split*: *pointer.splits prod.splits*)
next
assume ∗: ¬ *x* = *y*
thus *?thesis*
using # *Cons.prems*

by (*auto simp*: *items-def split*!: *pointer.splits prod.splits*, *metis Cons.IH Cons.prems items-def*)+
  qed
 next
  case *False*
  then show *?thesis*
  proof *cases*
   assume ∗: *fst e* = *fst a*
   thus *?thesis*
    using *Cons.prems* **by** (*auto simp*: *items-def split*: *pointer.splits prod.splits*)
  next
   assume ∗: ¬ *fst e* = *fst a*
   thus *?thesis*
    using *Cons.prems*
     by (*auto simp*: *items-def split*!: *pointer.splits prod.splits*, *metis Cons.IH Cons.prems items-def*)+
  qed
 qed
qed (*auto simp*: *items-def*)

lemma *bin-eq-items-dist-upds-bin-entries*:
 assumes *items es* = *items es′*
 shows *items* (*upds-bin es b*) = *items* (*upds-bin es′ b*)
 using *assms*
proof (*induction es arbitrary*: *es′ b*)
 case (*Cons e es*)
 then obtain *e′ es″* **where** *fst e* = *fst e′ items es* = *items es″ es′* = *e′* # *es″*
  by (*auto simp*: *items-def*)
 hence *items* (*upds-bin es* (*upd-bin e b*)) = *items* (*upds-bin es″* (*upd-bin e′ b*))
  using *Cons.IH*
  by (*metis bin-eq-items-dist-upd-bin-entry bin-eq-items-dist-upds-bin-bin*)
 thus *?case*
  by (*simp add*: ‹*es′* = *e′* # *es″*›)
qed (*auto simp*: *items-def*)

lemma *bins-eq-items-dist-upd-bins*:
 assumes *bins-eq-items as bs items aes* = *items bes k* < *length as*
 shows *bins-eq-items* (*upd-bins as k aes*) (*upd-bins bs k bes*)
proof −
 have *k* < *length bs*
  using *assms*(*1*,*3*) *bins-eq-items-def map-eq-imp-length-eq* **by** *metis*
 hence *items* (*upds-bin* (*as*!*k*) *aes*) = *items* (*upds-bin* (*bs*!*k*) *bes*)
  using *bin-eq-items-dist-upds-bin-entries bin-eq-items-dist-upds-bin-bin bins-eq-items-def assms*
  by (*metis* (*no-types*, *lifting*) *nth-map*)
 thus *?thesis*
  using ‹*k* < *length bs*› *assms bin-eq-items-dist-upds-bin-bin bin-eq-items-dist-upds-bin-entries bins-eq-items-def upd-bins-def* **by** (*smt* (*verit*) *map-update nth-map*)
qed

8.5 Well-formed bins

lemma *wf-bins-Scan$_L$'*:
assumes *wf-bins* $\mathcal{G}$ ω *bs* $k <$ *length bs* $x \in$ *set* (*items* (*bs* ! *k*))
assumes $k <$ *length* ω *next-symbol* $x \neq$ *None* $y =$ *inc-item* x $(k+1)$
shows *wf-item* $\mathcal{G}$ ω $y \wedge$ *end-item* $y = k+1$
using *assms wf-bins-kth-bin*[*OF assms*(*1*−*3*)]
unfolding *wf-item-def inc-item-def next-symbol-def is-complete-def rhs-item-def*
by (*auto split*: *if-splits*)

lemma *wf-bins-Scan$_L$*:
assumes *wf-bins* $\mathcal{G}$ ω *bs* $k <$ *length bs* $x \in$ *set* (*items* (*bs* ! *k*)) $k <$ *length* ω *next-symbol* $x \neq$ *None*
shows $\forall y \in$ *set* (*items* (*Scan$_L$* *k* ω *a x pre*)). *wf-item* $\mathcal{G}$ ω $y \wedge$ *end-item* $y = (k+1)$
using *wf-bins-Scan$_L$'*[*OF assms*] **by** (*simp add*: *Scan$_L$-def items-def*)

lemma *wf-bins-Predict$_L$*:
assumes *wf-bins* $\mathcal{G}$ ω *bs* $k <$ *length bs* $k \leq$ *length* ω
shows $\forall y \in$ *set* (*items* (*Predict$_L$* *k* $\mathcal{G}$ *X*)). *wf-item* $\mathcal{G}$ ω $y \wedge$ *end-item* $y = k$
using *assms* **by** (*auto simp*: *Predict$_L$-def wf-item-def wf-bins-def wf-bin-def init-item-def items-def*)

lemma *wf-item-inc-item*:
assumes *wf-item* $\mathcal{G}$ ω *x next-symbol* $x =$ *Some a start-item* $x \leq k$ $k \leq$ *length* ω
shows *wf-item* $\mathcal{G}$ ω (*inc-item x k*) $\wedge$ *end-item* (*inc-item x k*) $= k$
using *assms* **by** (*auto simp*: *wf-item-def inc-item-def rhs-item-def next-symbol-def is-complete-def split*: *if-splits*)

lemma *wf-bins-Complete$_L$*:
assumes *wf-bins* $\mathcal{G}$ ω *bs* $k <$ *length bs* $y \in$ *set* (*items* (*bs* ! *k*))
shows $\forall x \in$ *set* (*items* (*Complete$_L$* *k y bs red*)). *wf-item* $\mathcal{G}$ ω $x \wedge$ *end-item* $x = k$
proof −
let *?orig* = *bs* ! (*start-item y*)
let *?is* = *filter-with-index* (λx. *next-symbol* $x =$ *Some* (*lhs-item y*)) (*items ?orig*)
let *?is'* = *map* (λ(*x*, *pre*). (*inc-item x k*, *PreRed* (*start-item y*, *pre*, *red*) [])) *?is*
{
fix *x*
assume ∗: $x \in$ *set* (*map fst ?is*)
have *end-item* $x =$ *start-item* y
using ∗ *assms wf-bins-kth-bin wf-item-def filter-with-index-cong-filter*
by (*metis dual-order.strict-trans2 filter-is-subset subsetD*)
have *wf-item* $\mathcal{G}$ ω x
using ∗ *assms wf-bins-kth-bin wf-item-def filter-with-index-cong-filter*
by (*metis dual-order.strict-trans2 filter-is-subset subsetD*)
moreover have *next-symbol* $x =$ *Some* (*lhs-item y*)
using ∗ *filter-set filter-with-index-cong-filter member-filter* **by** *metis*
moreover have *start-item* $x \leq k$
using ‹*end-item* $x =$ *start-item* y› ‹*wf-item* $\mathcal{G}$ ω x› *assms wf-bins-kth-bin*

wf-item-def
  by (*metis dual-order.order-iff-strict dual-order.strict-trans1*)
  moreover have $k \le$ *length* ω
  using *assms wf-bins-kth-bin wf-item-def* **by** *blast*
  ultimately have *wf-item* $\mathcal{G}$ ω (*inc-item x k*) *end-item* (*inc-item x k*) = k
  by (*simp-all add*: *wf-item-inc-item*)
}
hence $\forall x \in$ *set* (*items ?is'*). *wf-item* $\mathcal{G}$ ω $x \wedge$ *end-item* $x = k$
  by (*auto simp*: *items-def rev-image-eqI*)
thus *?thesis*
  unfolding *Complete*$_L$*-def* **by** *presburger*
qed

lemma *Ex-wf-bins*:
$\exists n$ *bs* ω $\mathcal{G}$. $n \le$ *length* $\omega \wedge$ *length bs* = *Suc* (*length* ω) $\wedge$ *wf-bins* $\mathcal{G}$ ω *bs*
apply (*rule exI*[**where** *x*=*0*])
apply (*rule exI*[**where** *x*=[[]]])
apply (*rule exI*[**where** *x*=[]])
by (*auto simp*: *wf-bins-def wf-bin-def wf-bin-items-def items-def split*: *prod.splits*)

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

typedef *'a wf-bins* = *wf-earley-input*::(*nat* × *'a cfg* × *'a list* × *'a bins*) *set*
morphisms *from-wf-bins to-wf-bins*
using *Ex-wf-bins* **by** (*auto simp*: *wf-earley-input-def*)

lemma *wf-earley-input-elim*:
assumes (k, $\mathcal{G}$, ω, *bs*) $\in$ *wf-earley-input*
shows $k \le$ *length* $\omega \wedge k <$ *length bs* $\wedge$ *length bs* = *length* ω + *1* $\wedge$ *wf-bins* $\mathcal{G}$ ω *bs*
using *assms*(*1*) *from-wf-bins wf-earley-input-def* **by** (*smt* (*verit*) *Suc-eq-plus1 less-Suc-eq-le mem-Collect-eq prod.sel*(*1*) *snd-conv*)

lemma *wf-earley-input-intro*:
assumes $k \le$ *length* ω *length bs* = *length* ω + *1* *wf-bins* $\mathcal{G}$ ω *bs*
shows (k, $\mathcal{G}$, ω, *bs*) $\in$ *wf-earley-input*
by (*simp add*: *assms wf-earley-input-def*)

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

have ∗: $k \leq$ *length* ω *length bs* = *length* ω + *1 wf-bins* $\mathcal{G}$ ω *bs*
using *wf-earley-input-elim assms*(*1*) **by** *metis*+
have *x*: $x \in$ *set* (*items* (*bs* ! *k*))
using *assms*(*2*,*3*) **by** *simp*
have *start-item x* < *length bs*
using *x wf-bins-kth-bin* ∗ *wf-item-def*
by (*metis One-nat-def add.right-neutral add-Suc-right dual-order.trans le-imp-less-Suc*)
hence *wf-bins* $\mathcal{G}$ ω (*upd-bins bs k* (*Complete*$_L$ *k x bs red*))
using ∗ *Suc-eq-plus1 le-imp-less-Suc wf-bins-Complete*$_L$ *wf-bins-upd-bins x* **by** *metis*
thus *?thesis*
by (*simp add*: ∗(*1*−*3*) *wf-earley-input-def*)
qed

lemma *wf-earley-input-Scan*$_L$:
assumes (*k*, $\mathcal{G}$, ω, *bs*) ∈ *wf-earley-input* ¬ *length* (*items* (*bs* ! *k*)) ≤ *i*
assumes *x* = *items* (*bs* ! *k*) ! *i next-symbol x* = *Some a*
assumes *k* < *length* ω
shows (*k*, $\mathcal{G}$, ω, *upd-bins bs* (*k*+*1*) (*Scan*$_L$ *k* ω *a x pre*)) ∈ *wf-earley-input*
proof −
have ∗: $k \leq$ *length* ω *length bs* = *length* ω + *1 wf-bins* $\mathcal{G}$ ω *bs*
using *wf-earley-input-elim assms*(*1*) **by** *metis*+
have *x*: $x \in$ *set* (*items*(*bs* ! *k*))
using *assms*(*2*,*3*) **by** *simp*
have *wf-bins* $\mathcal{G}$ ω (*upd-bins bs* (*k*+*1*) (*Scan*$_L$ *k* ω *a x pre*))
using ∗ *x assms*(*1*,*4*,*5*) *wf-bins-Scan*$_L$ *wf-bins-upd-bins wf-earley-input-elim*
by (*metis option.discI*)
thus *?thesis*
by (*simp add*: ∗(*1*−*3*) *wf-earley-input-def*)
qed

lemma *wf-earley-input-Predict*$_L$:
assumes (*k*, $\mathcal{G}$, ω, *bs*) ∈ *wf-earley-input* ¬ *length* (*items* (*bs* ! *k*)) ≤ *i*
assumes *x* = *items* (*bs* ! *k*) ! *i next-symbol x* = *Some a*
shows (*k*, $\mathcal{G}$, ω, *upd-bins bs k* (*Predict*$_L$ *k* $\mathcal{G}$ *a*)) ∈ *wf-earley-input*
proof −
have ∗: $k \leq$ *length* ω *length bs* = *length* ω + *1 wf-bins* $\mathcal{G}$ ω *bs*
using *wf-earley-input-elim assms*(*1*) **by** *metis*+
have *x*: $x \in$ *set* (*items* (*bs* ! *k*))
using *assms*(*2*,*3*) **by** *simp*
hence *wf-bins* $\mathcal{G}$ ω (*upd-bins bs k* (*Predict*$_L$ *k* $\mathcal{G}$ *a*))
using ∗ *x assms*(*1*,*4*) *wf-bins-Predict*$_L$ *wf-bins-upd-bins wf-earley-input-elim* **by** *metis*
thus *?thesis*
by (*simp add*: ∗(*1*−*3*) *wf-earley-input-def*)
qed

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

$- i$

lemma $Earley_L$-bin'-$simps$[$simp$]:
$i \geq length\ (items\ (bs\ !\ k)) \Longrightarrow Earley_L\text{-}bin'\ k\ \mathcal{G}\ \omega\ bs\ i = bs$
$\neg\ i \geq length\ (items\ (bs\ !\ k)) \Longrightarrow x = items\ (bs!k)\ !\ i \Longrightarrow next\text{-}symbol\ x = None \Longrightarrow$
$Earley_L\text{-}bin'\ k\ \mathcal{G}\ \omega\ bs\ i = Earley_L\text{-}bin'\ k\ \mathcal{G}\ \omega\ (upd\text{-}bins\ bs\ k\ (Complete_L\ k\ x\ bs\ i))\ (i{+}1)$
$\neg\ i \geq length\ (items\ (bs\ !\ k)) \Longrightarrow x = items\ (bs!k)\ !\ i \Longrightarrow next\text{-}symbol\ x = Some\ a \Longrightarrow$
$a \notin nonterminals\ \mathcal{G} \Longrightarrow k < length\ \omega \Longrightarrow Earley_L\text{-}bin'\ k\ \mathcal{G}\ \omega\ bs\ i = Earley_L\text{-}bin'\ k\ \mathcal{G}\ \omega\ (upd\text{-}bins\ bs\ (k{+}1)\ (Scan_L\ k\ \omega\ a\ x\ i))\ (i{+}1)$
$\neg\ i \geq length\ (items\ (bs\ !\ k)) \Longrightarrow x = items\ (bs!k)\ !\ i \Longrightarrow next\text{-}symbol\ x = Some\ a \Longrightarrow$
$a \notin nonterminals\ \mathcal{G} \Longrightarrow \neg\ k < length\ \omega \Longrightarrow Earley_L\text{-}bin'\ k\ \mathcal{G}\ \omega\ bs\ i = Earley_L\text{-}bin'\ k\ \mathcal{G}\ \omega\ bs\ (i{+}1)$
$\neg\ i \geq length\ (items\ (bs\ !\ k)) \Longrightarrow x = items\ (bs!k)\ !\ i \Longrightarrow next\text{-}symbol\ x = Some\ a \Longrightarrow$
$a \in nonterminals\ \mathcal{G} \Longrightarrow Earley_L\text{-}bin'\ k\ \mathcal{G}\ \omega\ bs\ i = Earley_L\text{-}bin'\ k\ \mathcal{G}\ \omega\ (upd\text{-}bins\ bs\ k\ (Predict_L\ k\ \mathcal{G}\ a))\ (i{+}1)$
by ($subst$ $Earley_L$-bin'.$simps$, $auto$)+

lemma $Earley_L$-bin'-$induct$[$case$-$names$ $Base$ $Complete_F$ $Scan_F$ $Pass$ $Predict_F$]:
assumes $(k, \mathcal{G}, \omega, bs) \in wf\text{-}earley\text{-}input$
assumes *base*: $\bigwedge k\ \mathcal{G}\ \omega\ bs\ i.\ i \geq length\ (items\ (bs\ !\ k)) \Longrightarrow P\ k\ \mathcal{G}\ \omega\ bs\ i$
assumes *complete*: $\bigwedge k\ \mathcal{G}\ \omega\ bs\ i\ x.\ \neg\ i \geq length\ (items\ (bs\ !\ k)) \Longrightarrow x = items\ (bs\ !\ k)\ !\ i \Longrightarrow$
$next\text{-}symbol\ x = None \Longrightarrow P\ k\ \mathcal{G}\ \omega\ (upd\text{-}bins\ bs\ k\ (Complete_L\ k\ x\ bs\ i))\ (i{+}1) \Longrightarrow P\ k\ \mathcal{G}\ \omega\ bs\ i$
assumes *scan*: $\bigwedge k\ \mathcal{G}\ \omega\ bs\ i\ x\ a.\ \neg\ i \geq length\ (items\ (bs\ !\ k)) \Longrightarrow x = items\ (bs\ !\ k)\ !\ i \Longrightarrow$
$next\text{-}symbol\ x = Some\ a \Longrightarrow a \notin nonterminals\ \mathcal{G} \Longrightarrow k < length\ \omega \Longrightarrow$
$P\ k\ \mathcal{G}\ \omega\ (upd\text{-}bins\ bs\ (k{+}1)\ (Scan_L\ k\ \omega\ a\ x\ i))\ (i{+}1) \Longrightarrow P\ k\ \mathcal{G}\ \omega\ bs\ i$
assumes *pass*: $\bigwedge k\ \mathcal{G}\ \omega\ bs\ i\ x\ a.\ \neg\ i \geq length\ (items\ (bs\ !\ k)) \Longrightarrow x = items\ (bs\ !\ k)\ !\ i \Longrightarrow$
$next\text{-}symbol\ x = Some\ a \Longrightarrow a \notin nonterminals\ \mathcal{G} \Longrightarrow \neg\ k < length\ \omega \Longrightarrow$
$P\ k\ \mathcal{G}\ \omega\ bs\ (i{+}1) \Longrightarrow P\ k\ \mathcal{G}\ \omega\ bs\ i$
assumes *predict*: $\bigwedge k\ \mathcal{G}\ \omega\ bs\ i\ x\ a.\ \neg\ i \geq length\ (items\ (bs\ !\ k)) \Longrightarrow x = items\ (bs\ !\ k)\ !\ i \Longrightarrow$
$next\text{-}symbol\ x = Some\ a \Longrightarrow a \in nonterminals\ \mathcal{G} \Longrightarrow$
$P\ k\ \mathcal{G}\ \omega\ (upd\text{-}bins\ bs\ k\ (Predict_L\ k\ \mathcal{G}\ a))\ (i{+}1) \Longrightarrow P\ k\ \mathcal{G}\ \omega\ bs\ i$
shows $P\ k\ \mathcal{G}\ \omega\ bs\ i$
using $assms(1)$
proof ($induction\ n{\equiv}earley\text{-}measure\ (k, \mathcal{G}, \omega, bs)\ i\ arbitrary{:}\ bs\ i\ rule{:}\ nat\text{-}less\text{-}induct$)
case *1*
have wf: $k \leq length\ \omega\ \ length\ bs = length\ \omega + 1\ \ wf\text{-}bins\ \mathcal{G}\ \omega\ bs$
using *1.prems wf-earley-input-elim* **by** *metis+*
hence k: $k < length\ bs$

by *simp*
have *fin*: *finite* { x | x. *wf-item* $\mathcal{G}$ ω x $\wedge$ *end-item* x = k }
using *finiteness-UNIV-wf-item* **by** *fastforce*
show *?case*
proof *cases*
assume $i \geq$ *length* (*items* (*bs* ! k))
then show *?thesis*
by (*simp add*: *base*)
next
assume *a1*: $\neg$ $i \geq$ *length* (*items* (*bs* ! k))
let *?x* = *items* (*bs* ! k) ! i
have x: *?x* $\in$ *set* (*items* (*bs* ! k))
using *a1* **by** *fastforce*
show *?thesis*
proof *cases*
assume *a2*: *next-symbol ?x* = *None*
let *?bs'* = *upd-bins bs k* ($Complete_L$ *k ?x bs i*)
have *start-item ?x* < *length bs*
using *wf*(*3*) *k wf-bins-kth-bin wf-item-def x* **by** (*metis order-le-less-trans*)
hence *wf-bins'*: *wf-bins* $\mathcal{G}$ ω *?bs'*
using *wf-bins-*$Complete_L$ *wf*(*3*) *wf-bins-upd-bins k x* **by** *metis*
hence *wf'*: (k, $\mathcal{G}$, ω, *?bs'*) $\in$ *wf-earley-input*
using *wf*(*1*,*2*,*3*) *wf-earley-input-intro* **by** *fastforce*
have *sub*: *set* (*items* (*?bs'* ! k)) $\subseteq$ { x | x. *wf-item* $\mathcal{G}$ ω x $\wedge$ *end-item* x = k }
using *wf*(*1*,*2*) *wf-bins'* **unfolding** *wf-bin-def wf-bins-def wf-bin-items-def*
using *order-le-less-trans* **by** *auto*
have i < *length* (*items* (*?bs'* ! k))
using *a1* **by** (*metis dual-order.strict-trans1 items-def leI length-map length-nth-upd-bin-bins*)
also have ... = *card* (*set* (*items* (*?bs'* ! k)))
using *wf*(*1*,*2*) *wf-bins' distinct-card wf-bins-def wf-bin-def* **by** (*metis k length-upd-bins*)
also have ... $\leq$ *card* {x |x. *wf-item* $\mathcal{G}$ ω x $\wedge$ *end-item* x = k}
using *card-mono fin sub* **by** *blast*
finally have *card* {x |x. *wf-item* $\mathcal{G}$ ω x $\wedge$ *end-item* x = k} > i
by *blast*
hence *earley-measure* (k, $\mathcal{G}$, ω, *?bs'*) (*Suc* i) < *earley-measure* (k, $\mathcal{G}$, ω, *bs*) i
by *simp*
thus *?thesis*
using *1 a1 a2 complete wf'* **by** *simp*
next
assume *a2*: $\neg$ *next-symbol ?x* = *None*
then obtain a **where** *a-def*: *next-symbol ?x* = *Some a*
by *blast*
show *?thesis*
proof *cases*
assume *a3*: $a \notin$ *nonterminals* $\mathcal{G}$
show *?thesis*
proof *cases*
assume *a4*: k < *length* ω

let *?bs′* = *upd-bins bs* (*k*+*1*) (*Scan*$_L$ *k ω a ?x i*)
have *wf-bins′*: *wf-bins* $\mathcal{G}$ *ω ?bs′*
using *wf-bins-Scan*$_L$ *wf*(*1*,*3*) *wf-bins-upd-bins a2 a4 k x* **by** *metis*
hence *wf′*: (*k*, $\mathcal{G}$, *ω*, *?bs′*) ∈ *wf-earley-input*
using *wf*(*1*,*2*,*3*) *wf-earley-input-intro* **by** *fastforce*
have *sub*: *set* (*items* (*?bs′* ! *k*)) ⊆ { *x* | *x*. *wf-item* $\mathcal{G}$ *ω x* ∧ *end-item x* = *k* }
using *wf*(*1*,*2*) *wf-bins′* **unfolding** *wf-bin-def wf-bins-def wf-bin-items-def* **using** *order-le-less-trans* **by** *auto*
have *i* < *length* (*items* (*?bs′* ! *k*))
using *a1* **by** (*metis dual-order.strict-trans1 items-def leI length-map length-nth-upd-bin-bins*)
also have ... = *card* (*set* (*items* (*?bs′* ! *k*)))
using *wf*(*1*,*2*) *wf-bins′ distinct-card wf-bins-def wf-bin-def*
by (*metis Suc-eq-plus1 le-imp-less-Suc length-upd-bins*)
also have ... ≤ *card* {*x* |*x*. *wf-item* $\mathcal{G}$ *ω x* ∧ *end-item x* = *k*}
using *card-mono fin sub* **by** *blast*
finally have *card* {*x* |*x*. *wf-item* $\mathcal{G}$ *ω x* ∧ *end-item x* = *k*} > *i*
by *blast*
hence *earley-measure* (*k*, $\mathcal{G}$, *ω*, *?bs′*) (*Suc i*) < *earley-measure* (*k*, $\mathcal{G}$, *ω*, *bs*) *i*
by *simp*
thus *?thesis*
using *1 a1 a-def a3 a4 scan wf′* **by** *simp*
next
assume *a4*: ¬ *k* < *length ω*
have *sub*: *set* (*items* (*bs* ! *k*)) ⊆ { *x* | *x*. *wf-item* $\mathcal{G}$ *ω x* ∧ *end-item x* = *k* }
using *wf* **unfolding** *wf-bin-def wf-bins-def wf-bin-items-def* **using** *order-le-less-trans* **by** *auto*
have *i* < *length* (*items* (*bs* ! *k*))
using *a1* **by** *simp*
also have ... = *card* (*set* (*items* (*bs* ! *k*)))
using *wf distinct-card wf-bins-def wf-bin-def* **by** (*metis Suc-eq-plus1 le-imp-less-Suc*)
also have ... ≤ *card* {*x* |*x*. *wf-item* $\mathcal{G}$ *ω x* ∧ *end-item x* = *k*}
using *card-mono fin sub* **by** *blast*
finally have *card* {*x* |*x*. *wf-item* $\mathcal{G}$ *ω x* ∧ *end-item x* = *k*} > *i*
by *blast*
hence *earley-measure* (*k*, $\mathcal{G}$, *ω*, *bs*) (*Suc i*) < *earley-measure* (*k*, $\mathcal{G}$, *ω*, *bs*) *i*
by *simp*
thus *?thesis*
using *1 a1 a3 a4 a-def pass* **by** *simp*
qed
next
assume *a3*: ¬ *a* ∉ *nonterminals* $\mathcal{G}$
let *?bs′* = *upd-bins bs k* (*Predict*$_L$ *k* $\mathcal{G}$ *a*)
have *wf-bins′*: *wf-bins* $\mathcal{G}$ *ω ?bs′*
using *wf-bins-Predict*$_L$ *wf wf-bins-upd-bins k x* **by** *metis*
hence *wf′*: (*k*, $\mathcal{G}$, *ω*, *?bs′*) ∈ *wf-earley-input*

using *wf(1,2,3) wf-earley-input-intro* **by** *fastforce*
have *sub*: *set (items (?bs' ! k))* $\subseteq$ { *x* | *x. wf-item* $\mathcal{G}$ ω *x* $\wedge$ *end-item x* = *k* }
using *wf(1,2) wf-bins'* **unfolding** *wf-bin-def wf-bins-def wf-bin-items-def*
using *order-le-less-trans* **by** *auto*
have *i* < *length (items (?bs' ! k))*
using *a1* **by** *(metis dual-order.strict-trans1 items-def leI length-map length-nth-upd-bin-bins)*
also have ... = *card (set (items (?bs' ! k)))*
using *wf(1,2) wf-bins' distinct-card wf-bins-def wf-bin-def*
by *(metis Suc-eq-plus1 le-imp-less-Suc length-upd-bins)*
also have ... $\leq$ *card* {*x* |*x. wf-item* $\mathcal{G}$ ω *x* $\wedge$ *end-item x* = *k*}
using *card-mono fin sub* **by** *blast*
finally have *card* {*x* |*x. wf-item* $\mathcal{G}$ ω *x* $\wedge$ *end-item x* = *k*} > *i*
by *blast*
hence *earley-measure* (*k*, $\mathcal{G}$, ω, *?bs'*) (*Suc i*) < *earley-measure* (*k*, $\mathcal{G}$, ω, *bs*) *i*
by *simp*
thus *?thesis*
using *1 a1 a-def a3 a-def predict wf'* **by** *simp*
qed
qed
qed
qed

lemma *wf-earley-input-Earley*$_L$*-bin'*:
assumes (*k*, $\mathcal{G}$, ω, *bs*) $\in$ *wf-earley-input*
shows (*k*, $\mathcal{G}$, ω, *Earley*$_L$*-bin' k* $\mathcal{G}$ ω *bs i*) $\in$ *wf-earley-input*
using *assms*
proof (*induction i rule*: *Earley*$_L$*-bin'-induct*[*OF assms(1)*, *case-names Base Complete*$_F$ *Scan*$_F$ *Pass Predict*$_F$])
case (*Complete*$_F$ *k* $\mathcal{G}$ ω *bs i x*)
let *?bs'* = *upd-bins bs k* (*Complete*$_L$ *k x bs i*)
have (*k*, $\mathcal{G}$, ω, *?bs'*) $\in$ *wf-earley-input*
using *Complete*$_F$*.hyps Complete*$_F$*.prems wf-earley-input-Complete*$_L$ **by** *blast*
thus *?case*
using *Complete*$_F$*.IH Complete*$_F$*.hyps* **by** *simp*
next
case (*Scan*$_F$ *k* $\mathcal{G}$ ω *bs i x a*)
let *?bs'* = *upd-bins bs* (*k+1*) (*Scan*$_L$ *k* ω *a x i*)
have (*k*, $\mathcal{G}$, ω, *?bs'*) $\in$ *wf-earley-input*
using *Scan*$_F$*.hyps Scan*$_F$*.prems wf-earley-input-Scan*$_L$ **by** *metis*
thus *?case*
using *Scan*$_F$*.IH Scan*$_F$*.hyps* **by** *simp*
next
case (*Predict*$_F$ *k* $\mathcal{G}$ ω *bs i x a*)
let *?bs'* = *upd-bins bs k* (*Predict*$_L$ *k* $\mathcal{G}$ *a*)
have (*k*, $\mathcal{G}$, ω, *?bs'*) $\in$ *wf-earley-input*
using *Predict*$_F$*.hyps Predict*$_F$*.prems wf-earley-input-Predict*$_L$ **by** *metis*
thus *?case*

using $Predict_F$*.IH* $Predict_F$*.hyps* **by** *simp*
qed *simp-all*

lemma *wf-earley-input-*$Earley_L$*-bin*:
assumes $(k, \mathcal{G}, \omega, bs) \in$ *wf-earley-input*
shows $(k, \mathcal{G}, \omega,$ $Earley_L$*-bin* $k\ \mathcal{G}\ \omega\ bs) \in$ *wf-earley-input*
using *assms* **by** (*simp add*: $Earley_L$*-bin-def wf-earley-input-*$Earley_L$*-bin'*)

lemma *length-bins-*$Earley_L$*-bin'*:
assumes $(k, \mathcal{G}, \omega, bs) \in$ *wf-earley-input*
shows *length* ($Earley_L$*-bin'* $k\ \mathcal{G}\ \omega\ bs\ i$) = *length bs*
by (*metis assms wf-earley-input-*$Earley_L$*-bin' wf-earley-input-elim*)

lemma *length-nth-bin-*$Earley_L$*-bin'*:
assumes $(k, \mathcal{G}, \omega, bs) \in$ *wf-earley-input*
shows *length* (*items* ($Earley_L$*-bin'* $k\ \mathcal{G}\ \omega\ bs\ i$! l)) $\geq$ *length* (*items* (*bs* ! l))
using *length-nth-upd-bin-bins order-trans*
by (*induction i rule*: $Earley_L$*-bin'-induct*[*OF assms*]) (*auto simp*: *items-def*, *blast+*)

lemma *wf-bins-*$Earley_L$*-bin'*:
assumes $(k, \mathcal{G}, \omega, bs) \in$ *wf-earley-input*
shows *wf-bins* $\mathcal{G}\ \omega$ ($Earley_L$*-bin'* $k\ \mathcal{G}\ \omega\ bs\ i$)
using *assms wf-earley-input-*$Earley_L$*-bin' wf-earley-input-elim* **by** *blast*

lemma *wf-bins-*$Earley_L$*-bin*:
assumes $(k, \mathcal{G}, \omega, bs) \in$ *wf-earley-input*
shows *wf-bins* $\mathcal{G}\ \omega$ ($Earley_L$*-bin* $k\ \mathcal{G}\ \omega\ bs$)
using *assms* $Earley_L$*-bin-def wf-bins-*$Earley_L$*-bin'* **by** *metis*

lemma *kth-*$Earley_L$*-bin'-bins*:
assumes $(k, \mathcal{G}, \omega, bs) \in$ *wf-earley-input*
assumes $j <$ *length* (*items* (*bs* ! l))
shows *items* ($Earley_L$*-bin'* $k\ \mathcal{G}\ \omega\ bs\ i$! l) ! j = *items* (*bs* ! l) ! j
using *assms*(*2*)
proof (*induction i rule*: $Earley_L$*-bin'-induct*[*OF assms*(*1*), *case-names Base* $Complete_F$ $Scan_F$ *Pass* $Predict_F$])
case ($Complete_F$ $k\ \mathcal{G}\ \omega\ bs\ i\ x$)
let *?bs'* = *upd-bins bs k* ($Complete_L$ $k\ x\ bs\ i$)
have *items* ($Earley_L$*-bin'* $k\ \mathcal{G}\ \omega$ *?bs'* $(i + 1)$! l) ! j = *items* (*?bs'* ! l) ! j
using $Complete_F$*.IH* $Complete_F$*.prems length-nth-upd-bin-bins items-def order.strict-trans2* **by** (*metis length-map*)
also have ... = *items* (*bs* ! l) ! j
using $Complete_F$*.prems items-nth-idem-upd-bins nth-idem-upd-bins length-map items-def* **by** *metis*
finally show *?case*
using $Complete_F$*.hyps* **by** *simp*
next
case ($Scan_F$ $k\ \mathcal{G}\ \omega\ bs\ i\ x\ a$)

let *?bs'* = *upd-bins bs* (*k+1*) ($Scan_L$ *k* ω *a x i*)
have *items* ($Earley_L$*-bin'* *k* $\mathcal{G}$ ω *?bs'* (*i* + *1*) ! *l*) ! *j* = *items* (*?bs'* ! *l*) ! *j*
using $Scan_F$*.IH* $Scan_F$*.prems length-nth-upd-bin-bins order.strict-trans2 items-def*
by (*metis length-map*)
also have ... = *items* (*bs* ! *l*) ! *j*
using $Scan_F$*.prems items-nth-idem-upd-bins nth-idem-upd-bins length-map items-def*
by *metis*
finally show *?case*
using $Scan_F$*.hyps* **by** *simp*
next
case ($Predict_F$ *k* $\mathcal{G}$ ω *bs i x a*)
let *?bs'* = *upd-bins bs k* ($Predict_L$ *k* $\mathcal{G}$ *a*)
have *items* ($Earley_L$*-bin'* *k* $\mathcal{G}$ ω *?bs'* (*i* + *1*) ! *l*) ! *j* = *items* (*?bs'* ! *l*) ! *j*
using $Predict_F$*.IH* $Predict_F$*.prems length-nth-upd-bin-bins order.strict-trans2*
items-def **by** (*metis length-map*)
also have ... = *items* (*bs* ! *l*) ! *j*
using $Predict_F$*.prems items-nth-idem-upd-bins nth-idem-upd-bins length-map*
items-def **by** *metis*
finally show *?case*
using $Predict_F$*.hyps* **by** *simp*
qed *simp-all*

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 *standard*
fix *x*
assume *x* $\in$ *set* (*items* (*bs* ! *l*))
then obtain *j* **where** $*$: *j* < *length* (*items* (*bs* ! *l*)) *items* (*bs* ! *l*) ! *j* = *x*
using *in-set-conv-nth* **by** *metis*
have *x* = *items* ($Earley_L$*-bin'* *k* $\mathcal{G}$ ω *bs i* ! *l*) ! *j*
using *kth-*$Earley_L$*-bin'-bins assms* $*$ **by** *metis*
moreover have *j* < *length* (*items* ($Earley_L$*-bin'* *k* $\mathcal{G}$ ω *bs i* ! *l*))
using *assms* $*$(*1*) *length-nth-bin-*$Earley_L$*-bin' less-le-trans* **by** *blast*
ultimately show *x* $\in$ *set* (*items* ($Earley_L$*-bin'* *k* $\mathcal{G}$ ω *bs i* ! *l*))
by *simp*
qed

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*
by (*induction i rule*: $Earley_L$*-bin'-induct*[*OF assms*]) (*auto simp*: *upd-bins-def*)

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*))
by (*simp add*: *assms nth-*$Earley_L$*-bin'-eq*)

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

assumes $(k, \mathcal{G}, \omega, bs) \in$ *wf-earley-input*
shows *bins-upto* (*Earley*$_L$*-bin k* $\mathcal{G}$ ω *bs*) *k 0* = *bins-upto bs k 0*
unfolding *bins-upto-def bin-upto-def Earley*$_L$*-bin-def* **using** *set-items-Earley*$_L$*-bin′-eq assms nth-Earley*$_L$*-bin′-eq* **by** *fastforce*

lemma *wf-earley-input-Init*$_L$:
assumes $k \leq$ *length* ω
shows $(k, \mathcal{G}, \omega,$ *Init*$_L$ $\mathcal{G}$ $\omega) \in$ *wf-earley-input*
proof −
let *?rs* = *filter* ($\lambda r.$ *lhs-rule r* = $\mathfrak{S}$ $\mathcal{G}$) (*remdups* ($\mathfrak{R}$ $\mathcal{G}$))
let *?b0* = *map* ($\lambda r.$ (*init-item r 0*, *Null*)) *?rs*
let *?bs* = *replicate* (*length* ω + *1*) ([])
have *distinct* (*items ?b0*)
using *assms* **unfolding** *wf-bin-def wf-item-def items-def*
by (*auto simp*: *init-item-def distinct-map inj-on-def*)
moreover have $\forall x \in$ *set* (*items ?b0*). *wf-item* $\mathcal{G}$ ω *x* $\wedge$ *end-item x* = *0*
using *assms* **unfolding** *wf-bin-def wf-item-def* **by** (*auto simp*: *init-item-def items-def*)
moreover have *wf-bins* $\mathcal{G}$ ω *?bs*
unfolding *wf-bins-def wf-bin-def wf-bin-items-def items-def* **using** *less-Suc-eq-0-disj* **by** *force*
ultimately show *?thesis*
using *assms length-replicate wf-earley-input-intro*
unfolding *wf-bin-def Init*$_L$*-def wf-bin-def wf-bin-items-def wf-bins-def*
by (*metis* (*no-types*, *lifting*) *length-list-update nth-list-update-eq nth-list-update-neq*)
qed

lemma *length-bins-Init*$_L$[*simp*]:
length (*Init*$_L$ $\mathcal{G}$ ω) = *length* ω + *1*
by (*simp add*: *Init*$_L$*-def*)

lemma *wf-earley-input-Earley*$_L$*-bins*[*simp*]:
assumes $k \leq$ *length* ω
shows $(k, \mathcal{G}, \omega,$ *Earley*$_L$*-bins k* $\mathcal{G}$ $\omega) \in$ *wf-earley-input*
using *assms*
proof (*induction k*)
case *0*
have $(k, \mathcal{G}, \omega,$ *Init*$_L$ $\mathcal{G}$ $\omega) \in$ *wf-earley-input*
using *assms wf-earley-input-Init*$_L$ **by** *blast*
thus *?case*
by (*simp add*: *assms wf-earley-input-Init*$_L$ *wf-earley-input-Earley*$_L$*-bin*)
next
case (*Suc k*)
have (*Suc k*, $\mathcal{G}$, ω, *Earley*$_L$*-bins k* $\mathcal{G}$ $\omega) \in$ *wf-earley-input*
using *Suc.IH Suc.prems*(*1*) *Suc-leD assms wf-earley-input-elim wf-earley-input-intro* **by** *metis*
thus *?case*
by (*simp add*: *wf-earley-input-Earley*$_L$*-bin*)
qed

lemma *length-Earley$_L$-bins*[*simp*]:
  assumes $k \leq$ *length* ω
  shows *length* (*Earley$_L$-bins* k $\mathcal{G}$ ω) = *length* (*Init$_L$* $\mathcal{G}$ ω)
  using *assms wf-earley-input-Earley$_L$-bins wf-earley-input-elim* **by** *fastforce*

lemma *wf-bins-Earley$_L$-bins*[*simp*]:
  assumes $k \leq$ *length* ω
  shows *wf-bins* $\mathcal{G}$ ω (*Earley$_L$-bins* k $\mathcal{G}$ ω)
  using *assms wf-earley-input-Earley$_L$-bins wf-earley-input-elim* **by** *fastforce*

lemma *wf-bins-Earley$_L$*:
  wf-bins $\mathcal{G}$ ω (*Earley$_L$* $\mathcal{G}$ ω)
  by (*simp add*: *Earley$_L$-def*)

8.6 Soundness

lemma *Init$_L$-eq-Init$_F$*:
  bins (*Init$_L$* $\mathcal{G}$ ω) = *Init$_F$* $\mathcal{G}$
proof −
  let *?rs* = *filter* (λr. *lhs-rule* $r = \mathfrak{S}$ $\mathcal{G}$) (*remdups* ($\mathfrak{R}$ $\mathcal{G}$))
  let *?b0* = *map* (λr. (*init-item* r *0*, *Null*)) *?rs*
  let *?bs* = *replicate* (*length* ω + *1*) ([])
  have *bins* (*?bs*[*0* := *?b0*]) = *set* (*items* *?b0*)
  proof −
    have *bins* (*?bs*[*0* := *?b0*]) = $\bigcup$ {*set* (*items* ((*?bs*[*0* := *?b0*]) ! k)) |k. $k <$ *length* (*?bs*[*0* := *?b0*])}
      unfolding *bins-def* **by** *blast*
    also have ... = *set* (*items* ((*?bs*[*0* := *?b0*]) ! *0*)) $\cup$ $\bigcup$ {*set* (*items* ((*?bs*[*0* := *?b0*]) ! k)) |k. $k <$ *length* (*?bs*[*0* := *?b0*]) $\wedge$ $k \neq$ *0*}
      by *fastforce*
    also have ... = *set* (*items* (*?b0*))
      by (*auto simp*: *items-def*)
    finally show *?thesis* **.**
  qed
  also have ... = *Init$_F$* $\mathcal{G}$
    by (*auto simp*: *Init$_F$-def items-def lhs-rule-def*)
  finally show *?thesis*
    by (*auto simp*: *Init$_L$-def*)
qed

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 *standard*
  fix y
  assume $*$: $y \in$ *set* (*items* (*Scan$_L$* k ω a x *pre*))

have $x \in bin\ I\ k$
using *kth-bin-sub-bins assms*$(1-4)$ *items-def wf-bin-def wf-bins-def wf-bin-items-def bin-def* **by** *fastforce*
{
assume #: $k < length\ \omega\ \omega!k = a$
hence $y = inc\text{-}item\ x\ (k+1)$
using $*$ **unfolding** $Scan_L$*-def* **by** (*simp add*: *items-def*)
hence $y \in Scan_F\ k\ \omega\ I$
using ‹$x \in bin\ I\ k$› # *assms*(6) **unfolding** $Scan_F$*-def* **by** *blast*
}
thus $y \in Scan_F\ k\ \omega\ I$
using $*$ *assms*(5) **unfolding** $Scan_L$*-def* **by** (*auto simp*: *items-def*)
qed

lemma $Predict_L$*-sub-*$Predict_F$:
assumes *wf-bins* $\mathcal{G}\ \omega$ *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 *standard*
fix y
assume $*$: $y \in set\ (items\ (Predict_L\ k\ \mathcal{G}\ X))$
have $x \in bin\ I\ k$
using *kth-bin-sub-bins assms*$(1-4)$ *items-def wf-bin-def wf-bins-def bin-def wf-bin-items-def* **by** *fast*
let *?rs* = *filter* ($\lambda r.$ *lhs-rule* $r = X$) ($\mathfrak{R}\ \mathcal{G}$)
let *?xs* = *map* ($\lambda r.$ *init-item* $r\ k$) *?rs*
have $y \in set$ *?xs*
using $*$ **unfolding** $Predict_L$*-def items-def* **by** *simp*
then obtain r **where** $y = init\text{-}item\ r\ k\ lhs\text{-}rule\ r = X\ r \in set\ (\mathfrak{R}\ \mathcal{G})$ *next-symbol* $x = Some\ (lhs\text{-}rule\ r)$
using *assms*(5) **by** *auto*
thus $y \in Predict_F\ k\ \mathcal{G}\ I$
unfolding $Predict_F$*-def* **using** ‹$x \in bin\ I\ k$› **by** *blast*
qed

lemma $Complete_L$*-sub-*$Complete_F$:
assumes *wf-bins* $\mathcal{G}\ \omega$ *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 *standard*
fix x
assume $*$: $x \in set\ (items\ (Complete_L\ k\ y\ bs\ red))$
have $y \in bin\ I\ k$
using *kth-bin-sub-bins assms items-def wf-bin-def wf-bins-def bin-def wf-bin-items-def* **by** *fast*
let *?orig* = *bs* ! *start-item* y
let *?xs* = *filter-with-index* ($\lambda x.$ *next-symbol* $x = Some\ (lhs\text{-}item\ y)$) (*items ?orig*)
let *?xs'* = *map* ($\lambda(x, pre).$ (*inc-item* $x\ k$, *PreRed* (*start-item* y, *pre*, *red*) [])) *?xs*
have *0*: *start-item* $y < length\ bs$

using *wf-bins-def wf-bin-def wf-item-def wf-bin-items-def assms*(*1*,*3*,*4*)
by (*metis Orderings.preorder-class.dual-order.strict-trans1 leD not-le-imp-less*)
{
fix *z*
assume ∗: $z \in$ *set* (*map fst ?xs*)
have *next-symbol z* = *Some* (*lhs-item y*)
using ∗ **by** (*simp add*: *filter-with-index-cong-filter*)
moreover have $z \in$ *bin I* (*start-item y*)
using *0* ∗ *assms*(*1*,*2*) *bin-def kth-bin-sub-bins wf-bins-kth-bin filter-with-index-cong-filter*
by (*metis* (*mono-tags*, *lifting*) *filter-is-subset in-mono mem-Collect-eq*)
ultimately have *next-symbol z* = *Some* (*lhs-item y*) $z \in$ *bin I* (*start-item y*)
by *simp-all*
}
hence *1*: ∀ $z \in$ *set* (*map fst ?xs*). *next-symbol z* = *Some* (*lhs-item y*) ∧ $z \in$ *bin I* (*start-item y*)
by *blast*
obtain *z* **where** *z*: *x* = *inc-item z k* $z \in$ *set* (*map fst ?xs*)
using ∗ **unfolding** $Complete_L$*-def* **by** (*auto simp*: *rev-image-eqI items-def*)
moreover have *next-symbol z* = *Some* (*lhs-item y*) $z \in$ *bin I* (*start-item y*)
using *1 z* **by** *blast+*
ultimately show $x \in Complete_F\ k\ I$
using ‹$y \in$ *bin I k*› *assms*(*5*) **unfolding** $Complete_F$*-def next-symbol-def* **by** (*auto split*: *if-splits*)
qed

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* ∀ $x \in I$. *wf-item* $\mathcal{G}$ ω *x* ∀ $x \in I$. *sound-item* $\mathcal{G}$ ω *x*
shows ∀ $x \in$ *set* (*items* ($Scan_L$ *k* ω *a x i*)). *sound-item* $\mathcal{G}$ ω *x*
proof *standard*
fix *y*
assume $y \in$ *set* (*items* ($Scan_L$ *k* ω *a x i*))
hence $y \in Scan_F\ k\ \omega\ I$
by (*meson* $Scan_L$*-sub-*$Scan_F$ *assms*(*1*−*6*) *in-mono*)
thus *sound-item* $\mathcal{G}$ ω *y*
using *sound-Scan assms*(*7*,*8*) **unfolding** $Scan_F$*-def inc-item-def bin-def*
by (*smt* (*verit*, *best*) *item.exhaust-sel mem-Collect-eq*)
qed

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* ∀ $x \in I$. *wf-item* $\mathcal{G}$ ω *x* ∀ $x \in I$. *sound-item* $\mathcal{G}$ ω *x*
shows ∀ $x \in$ *set* (*items* ($Predict_L$ *k* $\mathcal{G}$ *X*)). *sound-item* $\mathcal{G}$ ω *x*
proof *standard*
fix *y*
assume $y \in$ *set* (*items* ($Predict_L$ *k* $\mathcal{G}$ *X*))

hence $y \in \mathit{Predict}_F\ k\ \mathcal{G}\ I$
by (*meson* $\mathit{Predict}_L$*-sub-*$\mathit{Predict}_F$ *assms*(*1*−*5*) *subsetD*)
thus *sound-item* $\mathcal{G}\ \omega\ y$
using *sound-Predict assms*(*6*,*7*) **unfolding** $\mathit{Predict}_F$*-def init-item-def bin-def*
by (*smt* (*verit*, *del-insts*) *item.exhaust-sel mem-Collect-eq*)
qed

lemma *sound-*$\mathit{Complete}_L$:
assumes *wf-bins* $\mathcal{G}\ \omega$ *bs bins bs* $\subseteq I$ $y \in$ *set* (*items* (*bs*!*k*)) $k <$ *length bs*
assumes *next-symbol* $y =$ *None* $\forall x \in I.$ *wf-item* $\mathcal{G}\ \omega\ x$ $\forall x \in I.$ *sound-item* $\mathcal{G}\ \omega$
x
shows $\forall x \in$ *set* (*items* ($\mathit{Complete}_L\ k\ y\ bs\ i$)). *sound-item* $\mathcal{G}\ \omega\ x$
proof *standard*
fix x
assume $x \in$ *set* (*items* ($\mathit{Complete}_L\ k\ y\ bs\ i$))
hence $x \in \mathit{Complete}_F\ k\ I$
using $\mathit{Complete}_L$*-sub-*$\mathit{Complete}_F$ *assms*(*1*−*5*) **by** *blast*
thus *sound-item* $\mathcal{G}\ \omega\ x$
using *sound-Complete assms*(*6*,*7*) **unfolding** $\mathit{Complete}_F$*-def inc-item-def bin-def*
by (*smt* (*verit*, *del-insts*) *item.exhaust-sel mem-Collect-eq*)
qed

lemma *sound-*Earley_L*-bin*′:
assumes (k, $\mathcal{G}$, ω, *bs*) $\in$ *wf-earley-input*
assumes $\forall x \in$ *bins bs*. *sound-item* $\mathcal{G}\ \omega\ x$
shows $\forall x \in$ *bins* (Earley_L*-bin*′ $k\ \mathcal{G}\ \omega$ *bs* i). *sound-item* $\mathcal{G}\ \omega\ x$
using *assms*
proof (*induction i rule*: Earley_L*-bin*′*-induct*[*OF assms*(*1*), *case-names Base Com-*
plete_F Scan_F *Pass* $\mathit{Predict}_F$])
case ($\mathit{Complete}_F\ k\ \mathcal{G}\ \omega$ *bs i x*)
let *?bs*′ = *upd-bins bs k* ($\mathit{Complete}_L\ k\ x\ bs\ i$)
have $x \in$ *set* (*items* (*bs* ! k))
using $\mathit{Complete}_F$*.hyps*(*1*,*2*) **by** *force*
hence $\forall x \in$ *set* (*items* ($\mathit{Complete}_L\ k\ x\ bs\ i$)). *sound-item* $\mathcal{G}\ \omega\ x$
using *sound-*$\mathit{Complete}_L$ $\mathit{Complete}_F$*.hyps*(*3*) $\mathit{Complete}_F$*.prems wf-earley-input-elim*
wf-bins-impl-wf-items **by** *fastforce*
moreover have (k, $\mathcal{G}$, ω, *?bs*′) $\in$ *wf-earley-input*
using $\mathit{Complete}_F$*.hyps* $\mathit{Complete}_F$*.prems*(*1*) *wf-earley-input-*$\mathit{Complete}_L$ **by** *blast*
ultimately have $\forall x \in$ *bins* (Earley_L*-bin*′ $k\ \mathcal{G}\ \omega$ *?bs*′ (i + *1*)). *sound-item* $\mathcal{G}\ \omega$
x
using $\mathit{Complete}_F$*.IH* $\mathit{Complete}_F$*.prems*(*2*) *length-upd-bins bins-upd-bins wf-earley-input-elim*
Suc-eq-plus1 Un-iff le-imp-less-Suc **by** *metis*
thus *?case*
using $\mathit{Complete}_F$*.hyps* **by** *simp*
next
case ($\mathit{Scan}_F\ k\ \mathcal{G}\ \omega$ *bs i x a*)
let *?bs*′ = *upd-bins bs* (k+*1*) ($\mathit{Scan}_L\ k\ \omega\ a\ x\ i$)
have $x \in$ *set* (*items* (*bs* ! k))
using Scan_F*.hyps*(*1*,*2*) **by** *force*

hence $\forall x \in set\ (items\ (Scan_L\ k\ \omega\ a\ x\ i)).\ sound\text{-}item\ \mathcal{G}\ \omega\ x$
using *sound-Scan*$_L$ *Scan*$_F$*.hyps*(*3*,*5*) *Scan*$_F$*.prems*(*1*,*2*) *wf-earley-input-elim wf-bins-impl-wf-items* **by** *fast*
moreover have $(k, \mathcal{G}, \omega, ?bs') \in wf\text{-}earley\text{-}input$
using *Scan*$_F$*.hyps* *Scan*$_F$*.prems*(*1*) *wf-earley-input-Scan*$_L$ **by** *metis*
ultimately have $\forall x \in bins\ (Earley_L\text{-}bin'\ k\ \mathcal{G}\ \omega\ ?bs'\ (i + 1)).\ sound\text{-}item\ \mathcal{G}\ \omega\ x$
using *Scan*$_F$*.IH* *Scan*$_F$*.hyps*(*5*) *Scan*$_F$*.prems*(*2*) *length-upd-bins bins-upd-bins wf-earley-input-elim*
by (*metis UnE add-less-cancel-right*)
thus *?case*
using *Scan*$_F$*.hyps* **by** *simp*
next
case (*Predict*$_F$ $k\ \mathcal{G}\ \omega\ bs\ i\ x\ a$)
let $?bs' = upd\text{-}bins\ bs\ k\ (Predict_L\ k\ \mathcal{G}\ a)$
have $x \in set\ (items\ (bs\ !\ k))$
using *Predict*$_F$*.hyps*(*1*,*2*) **by** *force*
hence $\forall x \in set\ (items(Predict_L\ k\ \mathcal{G}\ a)).\ sound\text{-}item\ \mathcal{G}\ \omega\ x$
using *sound-Predict*$_L$ *Predict*$_F$*.hyps*(*3*) *Predict*$_F$*.prems* *wf-earley-input-elim wf-bins-impl-wf-items* **by** *fast*
moreover have $(k, \mathcal{G}, \omega, ?bs') \in wf\text{-}earley\text{-}input$
using *Predict*$_F$*.hyps* *Predict*$_F$*.prems*(*1*) *wf-earley-input-Predict*$_L$ **by** *metis*
ultimately have $\forall x \in bins\ (Earley_L\text{-}bin'\ k\ \mathcal{G}\ \omega\ ?bs'\ (i + 1)).\ sound\text{-}item\ \mathcal{G}\ \omega\ x$
using *Predict*$_F$*.IH* *Predict*$_F$*.prems*(*2*) *length-upd-bins bins-upd-bins wf-earley-input-elim*
by (*metis Suc-eq-plus1 UnE*)
thus *?case*
using *Predict*$_F$*.hyps* **by** *simp*
qed *simp-all*

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 $\forall x \in bins\ (Earley_L\text{-}bin\ k\ \mathcal{G}\ \omega\ bs).\ sound\text{-}item\ \mathcal{G}\ \omega\ x$
using *sound-Earley*$_L$*-bin*$'$ *assms Earley*$_L$*-bin-def* **by** *metis*

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 $bins\ (Earley_L\text{-}bin'\ k\ \mathcal{G}\ \omega\ bs\ i) \subseteq Earley_F\text{-}bin\ k\ \mathcal{G}\ \omega\ I$
using *assms*
proof (*induction i arbitrary*: *I rule*: *Earley*$_L$*-bin*$'$*-induct*[*OF assms*(*1*), *case-names Base Complete*$_F$ *Scan*$_F$ *Pass Predict*$_F$])
case (*Base* $k\ \mathcal{G}\ \omega\ bs\ i$)
thus *?case*
using *Earley*$_F$*-bin-mono* **by** *fastforce*
next
case (*Complete*$_F$ $k\ \mathcal{G}\ \omega\ bs\ i\ x$)
let $?bs' = upd\text{-}bins\ bs\ k\ (Complete_L\ k\ x\ bs\ i)$

have $x \in set$ (*items* (*bs* ! *k*))
using $Complete_F.hyps(1,2)$ **by** *force*
hence *bins* $?bs' \subseteq I \cup Complete_F\ k\ I$
using $Complete_L$-*sub*-$Complete_F$ $Complete_F.hyps(3)$ $Complete_F.prems(1,2)$ *bins-upd-bins wf-earley-input-elim* **by** *blast*
moreover have $(k, \mathcal{G}, \omega, ?bs') \in$ *wf-earley-input*
using $Complete_F.hyps$ $Complete_F.prems(1)$ *wf-earley-input*-$Complete_L$ **by** *blast*
ultimately have *bins* ($Earley_L$-*bin'* $k\ \mathcal{G}\ \omega\ bs\ i$) $\subseteq Earley_F$-*bin* $k\ \mathcal{G}\ \omega$ ($I \cup Complete_F\ k\ I$)
using $Complete_F.IH$ $Complete_F.hyps$ **by** *simp*
also have ... $\subseteq Earley_F$-*bin* $k\ \mathcal{G}\ \omega$ ($Earley_F$-*bin* $k\ \mathcal{G}\ \omega\ I$)
using $Complete_F$-$Earley_F$-*bin-mono* $Earley_F$-*bin-mono* $Earley_F$-*bin-sub-mono* **by** (*metis Un-subset-iff*)
finally show *?case*
using $Earley_F$-*bin-idem* **by** *blast*
next
case ($Scan_F\ k\ \mathcal{G}\ \omega\ bs\ i\ x\ a$)
let $?bs'$ = *upd-bins* *bs* $(k+1)$ ($Scan_L\ k\ \omega\ a\ x\ i$)
have $x \in set$ (*items* (*bs* ! *k*))
using $Scan_F.hyps(1,2)$ **by** *force*
hence *bins* $?bs' \subseteq I \cup Scan_F\ k\ \omega\ I$
using $Scan_L$-*sub*-$Scan_F$ $Scan_F.hyps(3,5)$ $Scan_F.prems$ *bins-upd-bins wf-earley-input-elim*
by (*metis add-mono1 sup-mono*)
moreover have $(k, \mathcal{G}, \omega, ?bs') \in$ *wf-earley-input*
using $Scan_F.hyps$ $Scan_F.prems(1)$ *wf-earley-input*-$Scan_L$ **by** *metis*
ultimately have *bins* ($Earley_L$-*bin'* $k\ \mathcal{G}\ \omega\ bs\ i$) $\subseteq Earley_F$-*bin* $k\ \mathcal{G}\ \omega$ ($I \cup Scan_F\ k\ \omega\ I$)
using $Scan_F.IH$ $Scan_F.hyps$ **by** *simp*
thus *?case*
using $Scan_F$-$Earley_F$-*bin-mono* $Earley_F$-*bin-mono* $Earley_F$-*bin-sub-mono* $Earley_F$-*bin-idem*
by (*metis Un-subset-iff subset-trans*)
next
case (*Pass* $k\ \mathcal{G}\ \omega\ bs\ i\ x\ a$)
thus *?case*
by *simp*
next
case ($Predict_F\ k\ \mathcal{G}\ \omega\ bs\ i\ x\ a$)
let $?bs'$ = *upd-bins* *bs* *k* ($Predict_L\ k\ \mathcal{G}\ a$)
have $x \in set$ (*items* (*bs* ! *k*))
using $Predict_F.hyps(1,2)$ **by** *force*
hence *bins* $?bs' \subseteq I \cup Predict_F\ k\ \mathcal{G}\ I$
using $Predict_L$-*sub*-$Predict_F$ $Predict_F.hyps(3)$ $Predict_F.prems$ *bins-upd-bins wf-earley-input-elim*
by (*metis sup-mono*)
moreover have $(k, \mathcal{G}, \omega, ?bs') \in$ *wf-earley-input*
using $Predict_F.hyps$ $Predict_F.prems(1)$ *wf-earley-input*-$Predict_L$ **by** *metis*
ultimately have *bins* ($Earley_L$-*bin'* $k\ \mathcal{G}\ \omega\ bs\ i$) $\subseteq Earley_F$-*bin* $k\ \mathcal{G}\ \omega$ ($I \cup Predict_F\ k\ \mathcal{G}\ I$)

using $Predict_F$*.IH* $Predict_F$*.hyps* **by** *simp*
thus *?case*
using $Predict_F$*-*$Earley_F$*-bin-mono* $Earley_F$*-bin-mono* $Earley_F$*-bin-sub-mono* $Earley_F$*-bin-idem*
by (*metis Un-subset-iff subset-trans*)
qed

lemma $Earley_L$*-bin-sub-*$Earley_F$*-bin*:
assumes $(k, \mathcal{G}, \omega, bs) \in$ *wf-earley-input*
assumes *bins bs* $\subseteq I$
shows *bins* ($Earley_L$*-bin* $k\ \mathcal{G}\ \omega\ bs$) $\subseteq$ $Earley_F$*-bin* $k\ \mathcal{G}\ \omega\ I$
using *assms* $Earley_L$*-bin′-sub-*$Earley_F$*-bin* $Earley_L$*-bin-def* **by** *metis*

lemma $Earley_L$*-bins-sub-*$Earley_F$*-bins*:
assumes $k \leq$ *length* ω
shows *bins* ($Earley_L$*-bins* $k\ \mathcal{G}\ \omega$) $\subseteq$ $Earley_F$*-bins* $k\ \mathcal{G}\ \omega$
using *assms*
proof (*induction k*)
case *0*
have $(k, \mathcal{G}, \omega, Init_L\ \mathcal{G}\ \omega) \in$ *wf-earley-input*
using *assms*(*1*) *assms wf-earley-input-*$Init_L$ **by** *blast*
thus *?case*
by (*simp add*: $Init_L$*-eq-*$Init_F$ $Earley_L$*-bin-sub-*$Earley_F$*-bin assms wf-earley-input-*$Init_L$)
next
case (*Suc k*)
have (*Suc k*, $\mathcal{G}, \omega$, $Earley_L$*-bins* $k\ \mathcal{G}\ \omega$) $\in$ *wf-earley-input*
by (*simp add*: *Suc.prems*(*1*) *Suc-leD assms wf-earley-input-intro*)
thus *?case*
by (*simp add*: *Suc.IH Suc.prems*(*1*) *Suc-leD* $Earley_L$*-bin-sub-*$Earley_F$*-bin assms*)
qed

lemma $Earley_L$*-sub-*$Earley_F$:
bins ($Earley_L\ \mathcal{G}\ \omega$) $\subseteq Earley_F\ \mathcal{G}\ \omega$
using $Earley_L$*-bins-sub-*$Earley_F$*-bins* $Earley_F$*-def* $Earley_L$*-def* **by** (*metis dual-order.refl*)

theorem *soundness-*$Earley_L$:
assumes *recognizing* (*bins* ($Earley_L\ \mathcal{G}\ \omega$)) $\mathcal{G}\ \omega$
shows $\mathcal{G} \vdash [\mathfrak{S}\ \mathcal{G}] \Rightarrow^* \omega$
using *assms* $Earley_L$*-sub-*$Earley_F$ *recognizing-def soundness-*$Earley_F$ **by** (*meson subsetD*)

8.7 Completeness

lemma *bin-bins-upto-bins-eq*:
assumes *wf-bins* $\mathcal{G}\ \omega\ bs\ k <$ *length bs* $i \geq$ *length* (*items* (*bs* ! *k*)) $l \leq k$
shows *bin* (*bins-upto bs k i*) $l =$ *bin* (*bins bs*) l
unfolding *bins-upto-def bins-def bin-def* **using** *assms nat-less-le*
apply (*auto simp*: *nth-list-update bin-upto-eq-set-items wf-bins-kth-bin items-def*)
apply (*metis fst-conv image-eqI order.strict-trans2*)

by (*metis fst-conv image-eqI items-def list.set-map wf-bins-kth-bin*)

lemma *impossible-complete-item*:
assumes *sound-item* $\mathcal{G}$ ω *x is-complete x start-item x* = *k end-item x* = *k nonempty-derives* $\mathcal{G}$
shows *False*
proof −
have $\mathcal{G} \vdash$ [*lhs-item x*] $\Rightarrow^*$ []
using *assms*(*1*−*4*) **by** (*simp add*: *slice-empty is-complete-def sound-item-def* β*-item-def*)
thus *?thesis*
by (*meson assms*(*5*) *nonempty-derives-def*)
qed

lemma $Complete_F$*-Un-eq-terminal*:
assumes *next-symbol z* = *Some a a* $\notin$ *nonterminals* $\mathcal{G}$ $\forall x \in I$. *wf-item* $\mathcal{G}$ ω *x wf-item* $\mathcal{G}$ ω *z*
shows $Complete_F$ *k* ($I \cup \{z\}$) = $Complete_F$ *k I*
proof (*rule ccontr*)
assume $Complete_F$ *k* ($I \cup \{z\}$) $\neq$ $Complete_F$ *k I*
hence $Complete_F$ *k I* $\subset$ $Complete_F$ *k* ($I \cup \{z\}$)
using $Complete_F$*-sub-mono* **by** *blast*
then obtain *w x y* **where** $*$:
w $\in$ $Complete_F$ *k* ($I \cup \{z\}$) *w* $\notin$ $Complete_F$ *k I w* = *inc-item x k*
x $\in$ *bin* ($I \cup \{z\}$) (*start-item y*) *y* $\in$ *bin* ($I \cup \{z\}$) *k*
is-complete y next-symbol x = *Some* (*lhs-item y*)
unfolding $Complete_F$*-def* **by** *fast*
show *False*
proof (*cases x* = *z*)
case *True*
have *lhs-item y* $\in$ *nonterminals* $\mathcal{G}$
using $*$(*5*,*6*) *assms*
by (*auto simp*: *wf-item-def bin-def lhs-item-def lhs-rule-def next-symbol-def nonterminals-def*)
thus *?thesis*
using *True* $*$(*7*) *assms* **by** *simp*
next
case *False*
thus *?thesis*
using $*$ *assms*(*1*) **by** (*auto simp*: *next-symbol-def* $Complete_F$*-def bin-def*)
qed
qed

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

assume $Complete_F\ k\ (I \cup \{z\}) \neq Complete_F\ k\ I$
hence $Complete_F\ k\ I \subset Complete_F\ k\ (I \cup \{z\})$
using *$Complete_F$-sub-mono* **by** *blast*
then obtain $x\ x'\ y$ **where** $*$:
$x \in Complete_F\ k\ (I \cup \{z\})\ x \notin Complete_F\ k\ I\ x = inc\text{-}item\ x'\ k$
$x' \in bin\ (I \cup \{z\})\ (start\text{-}item\ y)\ y \in bin\ (I \cup \{z\})\ k$
$is\text{-}complete\ y\ next\text{-}symbol\ x' = Some\ (lhs\text{-}item\ y)$
unfolding *$Complete_F$-def* **by** *fast*
consider $(A)\ x' = z \mid (B)\ y = z$
using $*(2-7)$ *$Complete_F$-def* **by** (*auto simp*: *bin-def*; *blast*)
thus *False*
proof *cases*
case *A*
have $start\text{-}item\ y = k$
using $*(4)$ *A bin-def assms*(*5*) **by** (*metis* (*mono-tags*, *lifting*) *mem-Collect-eq*)
moreover have $end\text{-}item\ y = k$
using $*(5)$ *bin-def* **by** *blast*
moreover have $sound\text{-}item\ \mathcal{G}\ \omega\ y$
using $*(5,6)$ *assms*(*2*,*6*) **by** (*auto simp*: *bin-def next-symbol-def sound-item-def*)
moreover have $wf\text{-}item\ \mathcal{G}\ \omega\ y$
using $*(5)$ *assms*(*1*,*4*) *wf-item-def* **by** (*auto simp*: *bin-def*)
ultimately show *?thesis*
using *impossible-complete-item* $*(6)$ *assms*(*3*) **by** *blast*
next
case *B*
thus *?thesis*
using $*(6)$ *assms*(*6*) **by** (*auto simp*: *next-symbol-def*)
qed
qed

lemma *wf-item-in-kth-bin*:
$wf\text{-}bins\ \mathcal{G}\ \omega\ bs \Longrightarrow x \in bins\ bs \Longrightarrow end\text{-}item\ x = k \Longrightarrow x \in set\ (items\ (bs\ !\ k))$
using *bins-bin-exists wf-bins-kth-bin wf-bins-def* **by** *blast*

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 −
have $\bigwedge l.\ l \leq k \Longrightarrow bin\ (bins\text{-}upto\ bs\ k\ i)\ l = bin\ (bins\ bs)\ l$
using *bin-bins-upto-bins-eq*[*OF assms*] **by** *blast*
moreover have $\forall x \in bins\ bs.\ wf\text{-}item\ \mathcal{G}\ \omega\ x$
using *assms*(*1*) *wf-bins-impl-wf-items* **by** *metis*
ultimately show *?thesis*
unfolding *$Complete_F$-def bin-def wf-item-def wf-item-def* **by** *auto*
qed

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 *standard*
fix w
assume $w \in Complete_F\ k\ (I \cup \{z\})$
then obtain $x\ y$ **where** $*$:
$w = inc\text{-}item\ x\ k\ x \in bin\ (I \cup \{z\})\ (start\text{-}item\ y)\ y \in bin\ (I \cup \{z\})\ k$
$is\text{-}complete\ y\ next\text{-}symbol\ x = Some\ (lhs\text{-}item\ y)$
unfolding $Complete_F\text{-}def$ **by** *blast*
consider $(A)\ x = z \mid (B)\ y = z \mid \neg\ (x = z \vee y = z)$
by *blast*
thus $w \in bins\ bs \cup set\ (items\ (Complete_L\ k\ z\ bs\ red))$
proof *cases*
case A
thus *?thesis*
using $*(5)$ *assms*(*3*) **by** (*auto simp*: *next-symbol-def*)
next
case B
let $?orig = bs\ !\ start\text{-}item\ z$
let $?is = filter\text{-}with\text{-}index\ (\lambda x.\ next\text{-}symbol\ x = Some\ (lhs\text{-}item\ z))\ (items\ ?orig)$
have $x \in bin\ I\ (start\text{-}item\ y)$
using $B\ *(2)\ *(5)$ *assms*(*3*) **by** (*auto simp*: *next-symbol-def bin-def*)
moreover have $bin\ I\ (start\text{-}item\ z) \subseteq set\ (items\ (bs\ !\ start\text{-}item\ z))$
using *wf-item-in-kth-bin assms*(*2*,*4*) *bin-def* **by** *blast*
ultimately have $x \in set\ (map\ fst\ ?is)$
using $*(5)\ B$ **by** (*simp add*: *filter-with-index-cong-filter in-mono*)
thus *?thesis*
unfolding $Complete_L\text{-}def\ *(1)$ **by** (*auto simp*: *rev-image-eqI items-def*)
next
case *3*
thus *?thesis*
using $*$ *assms*(*1*) $Complete_F\text{-}def$ **by** (*auto simp*: *bin-def*; *blast*)
qed
qed

lemma $Complete_L\text{-}eq\text{-}start\text{-}item$:
$bs\ !\ start\text{-}item\ y = bs'\ !\ start\text{-}item\ y \Longrightarrow Complete_L\ k\ y\ bs\ red = Complete_L\ k\ y\ bs'\ red$
by (*auto simp*: $Complete_L\text{-}def$)

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 −
{
fix x
assume $x \in bins\text{-}upto\ bs\ k\ 0$
then obtain l **where** $x \in set\ (items\ (bs\ !\ l))\ l < k$
unfolding *bins-upto-def bin-upto-def* **by** *blast*
hence $end\text{-}item\ x = l$

using *wf-bins-kth-bin assms* **by** *fastforce*
hence *end-item* $x < k$
using ‹$l < k$› **by** *blast*
}
thus *?thesis*
by (*auto simp*: *bin-def*)
qed

lemma *Earley*$_L$*-bin′-mono*:
assumes $(k, \mathcal{G}, \omega, bs) \in$ *wf-earley-input*
shows *bins bs* $\subseteq$ *bins* (*Earley*$_L$*-bin′* k $\mathcal{G}$ ω *bs* i)
using *assms*
proof (*induction i rule*: *Earley*$_L$*-bin′-induct*[*OF assms*(*1*), *case-names Base Complete*$_F$ *Scan*$_F$ *Pass Predict*$_F$])
case (*Complete*$_F$ k $\mathcal{G}$ ω *bs* i x)
let *?bs′* = *upd-bins bs k* (*Complete*$_L$ k x *bs* i)
have *wf*: $(k, \mathcal{G}, \omega, ?bs') \in$ *wf-earley-input*
using *Complete*$_F$*.hyps Complete*$_F$*.prems*(*1*) *wf-earley-input-Complete*$_L$ **by** *blast*
hence *bins bs* $\subseteq$ *bins ?bs′*
using *length-upd-bins bins-upd-bins wf-earley-input-elim* **by** (*metis Un-upper1*)
also have ... $\subseteq$ *bins* (*Earley*$_L$*-bin′* k $\mathcal{G}$ ω *?bs′* $(i + 1)$)
using *wf Complete*$_F$*.IH* **by** *blast*
finally show *?case*
using *Complete*$_F$*.hyps* **by** *simp*
next
case (*Scan*$_F$ k $\mathcal{G}$ ω *bs* i x a)
let *?bs′* = *upd-bins bs* $(k+1)$ (*Scan*$_L$ k ω a x i)
have *wf*: $(k, \mathcal{G}, \omega, ?bs') \in$ *wf-earley-input*
using *Scan*$_F$*.hyps Scan*$_F$*.prems*(*1*) *wf-earley-input-Scan*$_L$ **by** *metis*
hence *bins bs* $\subseteq$ *bins ?bs′*
using *Scan*$_F$*.hyps*(*5*) *length-upd-bins bins-upd-bins wf-earley-input-elim*
by (*metis add-mono1 sup-ge1*)
also have ... $\subseteq$ *bins* (*Earley*$_L$*-bin′* k $\mathcal{G}$ ω *?bs′* $(i + 1)$)
using *wf Scan*$_F$*.IH* **by** *blast*
finally show *?case*
using *Scan*$_F$*.hyps* **by** *simp*
next
case (*Predict*$_F$ k $\mathcal{G}$ ω *bs* i x a)
let *?bs′* = *upd-bins bs k* (*Predict*$_L$ k $\mathcal{G}$ a)
have *wf*: $(k, \mathcal{G}, \omega, ?bs') \in$ *wf-earley-input*
using *Predict*$_F$*.hyps Predict*$_F$*.prems*(*1*) *wf-earley-input-Predict*$_L$ **by** *metis*
hence *bins bs* $\subseteq$ *bins ?bs′*
using *length-upd-bins bins-upd-bins wf-earley-input-elim* **by** (*metis sup-ge1*)
also have ... $\subseteq$ *bins* (*Earley*$_L$*-bin′* k $\mathcal{G}$ ω *?bs′* $(i + 1)$)
using *wf Predict*$_F$*.IH* **by** *blast*
finally show *?case*
using *Predict*$_F$*.hyps* **by** *simp*
qed *simp-all*

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*)
using *assms*
proof (*induction i rule*: $Earley_L$*-bin'-induct*[*OF assms*(*1*), *case-names Base Complete*$_F$ *Scan*$_F$ *Pass Predict*$_F$])
case (*Base k* $\mathcal{G}\ \omega$ *bs i*)
have *bin* (*bins bs*) *k* = *bin* (*bins-upto bs k i*) *k*
using *Base.hyps Base.prems*(*1*) *bin-bins-upto-bins-eq wf-earley-input-elim* **by** *blast*
thus *?case*
using $Scan_F$*-bin-absorb* $Predict_F$*-bin-absorb* $Complete_F$*-bins-upto-eq-bins wf-earley-input-elim* *Base.hyps Base.prems*(*1*,*2*,*3*,*5*) $Earley_F$*-bin-step-def* $Complete_F$*-*$Earley_F$*-bin-step-mono* $Predict_F$*-*$Earley_F$*-bin-step-mono* $Scan_F$*-*$Earley_F$*-bin-step-mono* $Earley_L$*-bin'-mono*
by (*metis* (*no-types*, *lifting*) *Un-assoc sup.orderE*)
next
case ($Complete_F$ *k* $\mathcal{G}\ \omega$ *bs i x*)
let *?bs'* = *upd-bins bs k* ($Complete_L$ *k x bs i*)
have *x*: $x \in$ *set* (*items* (*bs* ! *k*))
using $Complete_F$*.hyps*(*1*,*2*) **by** *auto*
have *wf*: $(k, \mathcal{G}, \omega,$ *?bs'*$) \in$ *wf-earley-input*
using $Complete_F$*.hyps* $Complete_F$*.prems*(*1*) *wf-earley-input-*$Complete_L$ **by** *blast*
hence *sound*: $\forall x \in$ *set* (*items* ($Complete_L$ *k x bs i*)). *sound-item* $\mathcal{G}\ \omega\ x$
using *sound-*$Complete_L$ $Complete_F$*.hyps*(*3*) $Complete_F$*.prems wf-earley-input-elim* *wf-bins-impl-wf-items x*
by (*metis dual-order.refl*)
have $Scan_F$ *k* ω (*bins-upto ?bs' k* (*i* + *1*)) $\subseteq$ *bins ?bs'*
proof −
have $Scan_F$ *k* ω (*bins-upto ?bs' k* (*i* + *1*)) = $Scan_F$ *k* ω (*bins-upto ?bs' k i* $\cup$ {*items* (*?bs'* ! *k*) ! *i*})
using $Complete_F$*.hyps*(*1*) *bins-upto-Suc-Un length-nth-upd-bin-bins items-def*
by (*metis length-map linorder-not-less sup.boundedE sup.order-iff*)
also have ... = $Scan_F$ *k* ω (*bins-upto bs k i* $\cup$ {*x*})
using $Complete_F$*.hyps*(*1*,*2*) $Complete_F$*.prems*(*1*) *items-nth-idem-upd-bins* *bins-upto-kth-nth-idem wf-earley-input-elim*
by (*metis dual-order.refl items-def length-map not-le-imp-less*)
also have ... $\subseteq$ *bins bs* $\cup$ $Scan_F$ *k* ω {*x*}
using $Complete_F$*.prems*(*2*,*3*) $Scan_F$*-Un* $Scan_F$*-*$Earley_F$*-bin-step-mono* **by** *fastforce*
also have ... = *bins bs*
using $Complete_F$*.hyps*(*3*) **by** (*auto simp*: $Scan_F$*-def bin-def*)
finally show *?thesis*
using $Complete_F$*.prems*(*1*) *wf-earley-input-elim bins-upd-bins* **by** *blast*
qed
moreover have $Predict_F$ *k* $\mathcal{G}$ (*bins-upto ?bs' k* (*i* + *1*)) $\subseteq$ *bins ?bs'*
proof −
have $Predict_F$ *k* $\mathcal{G}$ (*bins-upto ?bs' k* (*i* + *1*)) = $Predict_F$ *k* $\mathcal{G}$ (*bins-upto ?bs' k*

$i \cup \{items\ (?bs'\ !\ k)\ !\ i\})$
using *Complete$_F$.hyps(1) bins-upto-Suc-Un length-nth-upd-bin-bins*
by (*metis dual-order.strict-trans1 items-def length-map not-le-imp-less*)
also have ... $=$ *Predict$_F$ k $\mathcal{G}$* (*bins-upto bs k i* $\cup \{x\}$)
using *Complete$_F$.hyps(1,2) Complete$_F$.prems(1) items-nth-idem-upd-bins bins-upto-kth-nth-idem wf-earley-input-elim*
by (*metis dual-order.refl items-def length-map not-le-imp-less*)
also have ... $\subseteq$ *bins bs* $\cup$ *Predict$_F$ k $\mathcal{G}$* $\{x\}$
using *Complete$_F$.prems(2,3) Predict$_F$-Un Predict$_F$-Earley$_F$-bin-step-mono*
by *blast*
also have ... $=$ *bins bs*
using *Complete$_F$.hyps(3)* **by** (*auto simp*: *Predict$_F$-def bin-def*)
finally show *?thesis*
using *Complete$_F$.prems(1) wf-earley-input-elim bins-upd-bins* **by** *blast*
qed
moreover have *Complete$_F$ k* (*bins-upto ?bs' k* $(i + 1)$) $\subseteq$ *bins ?bs'*
proof $-$
have *Complete$_F$ k* (*bins-upto ?bs' k* $(i + 1)$) $=$ *Complete$_F$ k* (*bins-upto ?bs' k* $i \cup \{items\ (?bs'\ !\ k)\ !\ i\})$
using *bins-upto-Suc-Un length-nth-upd-bin-bins Complete$_F$.hyps(1)*
by (*metis* (*no-types*, *opaque-lifting*) *dual-order.trans items-def length-map not-le-imp-less*)
also have ... $=$ *Complete$_F$ k* (*bins-upto bs k i* $\cup \{x\}$)
using *items-nth-idem-upd-bins Complete$_F$.hyps(1,2) bins-upto-kth-nth-idem Complete$_F$.prems(1) wf-earley-input-elim*
by (*metis dual-order.refl items-def length-map not-le-imp-less*)
also have ... $\subseteq$ *bins bs* $\cup$ *set* (*items* (*Complete$_L$ k x bs i*))
using *Complete$_F$-sub-bins-Un-Complete$_L$ Complete$_F$.hyps(3) Complete$_F$.prems(1,2,3) next-symbol-def*
bins-upto-sub-bins wf-bins-kth-bin x Complete$_F$-Earley$_F$-bin-step-mono wf-earley-input-elim
by (*smt* (*verit*, *best*) *option.distinct(1) subset-trans*)
finally show *?thesis*
using *Complete$_F$.prems(1) wf-earley-input-elim bins-upd-bins* **by** *blast*
qed
ultimately have *Earley$_F$-bin-step k $\mathcal{G}$ ω* (*bins ?bs'*) $\subseteq$ *bins* (*Earley$_L$-bin' k $\mathcal{G}$ ω ?bs'* $(i+1)$)
using *Complete$_F$.IH Complete$_F$.prems sound wf Earley$_F$-bin-step-def bins-upto-sub-bins*
wf-earley-input-elim bins-upd-bins
by (*metis UnE sup.boundedI*)
thus *?case*
using *Complete$_F$.hyps Complete$_F$.prems(1) Earley$_L$-bin'-simps(2) Earley$_F$-bin-step-sub-mono bins-upd-bins wf-earley-input-elim*
by (*smt* (*verit*, *best*) *sup.coboundedI2 sup.orderE sup-ge1*)
next
case (*Scan$_F$ k $\mathcal{G}$ ω bs i x a*)
let *?bs'* $=$ *upd-bins bs* $(k+1)$ (*Scan$_L$ k ω a x i*)
have *x*: $x \in$ *set* (*items* (*bs* ! *k*))
using *Scan$_F$.hyps(1,2)* **by** *auto*
hence *sound*: $\forall x \in$ *set* (*items* (*Scan$_L$ k ω a x i*)). *sound-item $\mathcal{G}$ ω x*

using *sound-*$Scan_L$ $Scan_F$*.hyps*(*3*,*5*) $Scan_F$*.prems*(*1*,*2*,*3*) *wf-earley-input-elim wf-bins-impl-wf-items x*
by (*metis dual-order.refl*)
have *wf*: $(k, \mathcal{G}, \omega, ?bs') \in$ *wf-earley-input*
using $Scan_F$*.hyps* $Scan_F$*.prems*(*1*) *wf-earley-input-*$Scan_L$ **by** *metis*
have $Scan_F\ k\ \omega$ (*bins-upto ?bs′ k* $(i + 1)$) $\subseteq$ *bins ?bs′*
proof −
have $Scan_F\ k\ \omega$ (*bins-upto ?bs′ k* $(i + 1)$) $=$ $Scan_F\ k\ \omega$ (*bins-upto ?bs′ k i* $\cup$ $\{$*items* (*?bs′* ! *k*) ! *i*$\}$)
using *bins-upto-Suc-Un* $Scan_F$*.hyps*(*1*) *nth-idem-upd-bins*
by (*metis Suc-eq-plus1 items-def length-map lessI less-not-refl not-le-imp-less*)
also have ... $=$ $Scan_F\ k\ \omega$ (*bins-upto bs k i* $\cup \{x\}$)
using $Scan_F$*.hyps*(*1*,*2*,*5*) $Scan_F$*.prems*(*1*,*2*) *nth-idem-upd-bins bins-upto-kth-nth-idem wf-earley-input-elim*
by (*metis add-mono-thms-linordered-field*(*1*) *items-def length-map less-add-one linorder-le-less-linear not-add-less1*)
also have ... $\subseteq$ *bins bs* $\cup$ $Scan_F\ k\ \omega\ \{x\}$
using $Scan_F$*.prems*(*2*,*3*) $Scan_F$*-Un* $Scan_F$*-*$Earley_F$*-bin-step-mono* **by** *fastforce*
finally have $*$: $Scan_F\ k\ \omega$ (*bins-upto ?bs′ k* $(i + 1)$) $\subseteq$ *bins bs* $\cup$ $Scan_F\ k\ \omega\ \{x\}$ **.**
show *?thesis*
proof *cases*
assume *a1*: $\omega!k = a$
hence $Scan_F\ k\ \omega\ \{x\} = \{$*inc-item x* $(k{+}1)\}$
using $Scan_F$*.hyps*(*1*−*3*,*5*) $Scan_F$*.prems*(*1*,*2*) *wf-earley-input-elim* **apply** (*auto simp*: $Scan_F$*-def bin-def*)
using *wf-bins-kth-bin x* **by** *blast*
hence $Scan_F\ k\ \omega$ (*bins-upto ?bs′ k* $(i + 1)$) $\subseteq$ *bins bs* $\cup$ $\{$*inc-item x* $(k{+}1)\}$
using $*$ **by** *blast*
also have ... $=$ *bins bs* $\cup$ *set* (*items* ($Scan_L\ k\ \omega\ a\ x\ i$))
using *a1* $Scan_F$*.hyps*(*5*) **by** (*auto simp*: $Scan_L$*-def items-def*)
also have ... $=$ *bins ?bs′*
using $Scan_F$*.hyps*(*5*) $Scan_F$*.prems*(*1*) *wf-earley-input-elim bins-upd-bins* **by** (*metis add-mono1*)
finally show *?thesis* **.**
next
assume *a1*: $\neg\ \omega!k = a$
hence $Scan_F\ k\ \omega\ \{x\} = \{\}$
using $Scan_F$*.hyps*(*3*) **by** (*auto simp*: $Scan_F$*-def bin-def*)
hence $Scan_F\ k\ \omega$ (*bins-upto ?bs′ k* $(i + 1)$) $\subseteq$ *bins bs*
using $*$ **by** *blast*
also have ... $\subseteq$ *bins ?bs′*
using $Scan_F$*.hyps*(*5*) $Scan_F$*.prems*(*1*) *wf-earley-input-elim bins-upd-bins*
by (*metis Un-left-absorb add-strict-right-mono subset-Un-eq*)
finally show *?thesis* **.**
qed
qed
moreover have $Predict_F\ k\ \mathcal{G}$ (*bins-upto ?bs′ k* $(i + 1)$) $\subseteq$ *bins ?bs′*
proof −

have $Predict_F$ *k* $\mathcal{G}$ (*bins-upto ?bs′ k* (*i* + *1*)) = $Predict_F$ *k* $\mathcal{G}$ (*bins-upto ?bs′ k*
i ∪ {*items* (*?bs′* ! *k*) ! *i*})
using *bins-upto-Suc-Un* $Scan_F$*.hyps*(*1*) *nth-idem-upd-bins*
by (*metis Suc-eq-plus1 dual-order.refl items-def length-map lessI linorder-not-less*)
also have ... = $Predict_F$ *k* $\mathcal{G}$ (*bins-upto bs k i* ∪ {*x*})
using $Scan_F$*.hyps*(*1*,*2*,*5*) $Scan_F$*.prems*(*1*,*2*) *nth-idem-upd-bins bins-upto-kth-nth-idem*
wf-earley-input-elim
by (*metis add-strict-right-mono items-def le-add1 length-map less-add-one*
linorder-not-le)
also have ... ⊆ *bins bs* ∪ $Predict_F$ *k* $\mathcal{G}$ {*x*}
using $Scan_F$*.prems*(*2*,*3*) $Predict_F$*-Un* $Predict_F$*-*$Earley_F$*-bin-step-mono* **by**
fastforce
also have ... = *bins bs*
using $Scan_F$*.hyps*(*3*,*4*) $Scan_F$*.prems*(*1*) *wf-earley-input-elim*
apply (*auto simp*: $Predict_F$*-def bin-def lhs-rule-def*)
by (*smt* (*verit*) *UnCI in-set-zipE nonterminals-def zip-map-fst-snd*)
finally show *?thesis*
using $Scan_F$*.hyps*(*5*) $Scan_F$*.prems*(*1*) **by** (*simp add*: *bins-upd-bins sup.coboundedI1*
wf-earley-input-elim)
qed
moreover have $Complete_F$ *k* (*bins-upto ?bs′ k* (*i* + *1*)) ⊆ *bins ?bs′*
proof −
have $Complete_F$ *k* (*bins-upto ?bs′ k* (*i* + *1*)) = $Complete_F$ *k* (*bins-upto ?bs′ k*
i ∪ {*items* (*?bs′* ! *k*) ! *i*})
using *bins-upto-Suc-Un* $Scan_F$*.hyps*(*1*) *nth-idem-upd-bins*
by (*metis Suc-eq-plus1 items-def length-map lessI less-not-refl not-le-imp-less*)
also have ... = $Complete_F$ *k* (*bins-upto bs k i* ∪ {*x*})
using $Scan_F$*.hyps*(*1*,*2*,*5*) $Scan_F$*.prems*(*1*,*2*) *nth-idem-upd-bins bins-upto-kth-nth-idem*
wf-earley-input-elim
by (*metis add-mono1 items-def length-map less-add-one linorder-not-le not-add-less1*)
also have ... = $Complete_F$ *k* (*bins-upto bs k i*)
using $Complete_F$*-Un-eq-terminal* $Scan_F$*.hyps*(*3*,*4*) $Scan_F$*.prems bins-upto-sub-bins*
subset-iff
wf-bins-impl-wf-items wf-bins-kth-bin wf-item-def x wf-earley-input-elim
by (*smt* (*verit*, *ccfv-threshold*))
finally show *?thesis*
using $Scan_F$*.hyps*(*5*) $Scan_F$*.prems*(*1*,*2*,*3*) $Complete_F$*-*$Earley_F$*-bin-step-mono*
by (*auto simp*: *bins-upd-bins wf-earley-input-elim*, *blast*)
qed
ultimately have $Earley_F$*-bin-step k* $\mathcal{G}$ ω (*bins ?bs′*) ⊆ *bins* ($Earley_L$*-bin′ k* $\mathcal{G}$ ω
?bs′ (*i*+*1*))
using $Scan_F$*.IH* $Scan_F$*.prems* $Scan_F$*.hyps*(*5*) *sound wf* $Earley_F$*-bin-step-def*
bins-upto-sub-bins wf-earley-input-elim
bins-upd-bins **by** (*metis UnE add-mono1 le-supI*)
thus *?case*
using $Earley_F$*-bin-step-sub-mono* $Earley_L$*-bin′-simps*(*3*) $Scan_F$*.hyps* $Scan_F$*.prems*(*1*)
wf-earley-input-elim bins-upd-bins
by (*smt* (*verit*, *ccfv-SIG*) *add-mono1 sup.coboundedI sup.coboundedI2 sup.orderE*)
next

case $(Pass\ k\ \mathcal{G}\ \omega\ bs\ i\ x\ a)$
have x: $x \in set\ (items\ (bs\ !\ k))$
using *Pass.hyps(1,2)* **by** *auto*
have $Scan_F\ k\ \omega\ (bins\text{-}upto\ bs\ k\ (i + 1)) \subseteq bins\ bs$
using *$Scan_F$-def Pass.hyps(5)* **by** *auto*
moreover have $Predict_F\ k\ \mathcal{G}\ (bins\text{-}upto\ bs\ k\ (i + 1)) \subseteq bins\ bs$
proof −
have $Predict_F\ k\ \mathcal{G}\ (bins\text{-}upto\ bs\ k\ (i + 1)) = Predict_F\ k\ \mathcal{G}\ (bins\text{-}upto\ bs\ k\ i \cup \{items\ (bs\ !\ k)\ !\ i\})$
using *bins-upto-Suc-Un Pass.hyps(1)* **by** *(metis items-def length-map not-le-imp-less)*
also have ... $= Predict_F\ k\ \mathcal{G}\ (bins\text{-}upto\ bs\ k\ i \cup \{x\})$
using *Pass.hyps(1,2,5) nth-idem-upd-bins bins-upto-kth-nth-idem* **by** *simp*
also have ... $\subseteq bins\ bs \cup Predict_F\ k\ \mathcal{G}\ \{x\}$
using *Pass.prems(2) $Predict_F$-Un $Predict_F$-$Earley_F$-bin-step-mono* **by** *blast*
also have ... $= bins\ bs$
using *Pass.hyps(3,4) Pass.prems(1) wf-earley-input-elim*
apply *(auto simp: $Predict_F$-def bin-def lhs-rule-def)*
by *(smt (verit, ccfv-SIG) UnCI fst-conv imageI list.set-map nonterminals-def)*
finally show *?thesis*
using *bins-upd-bins Pass.hyps(5) Pass.prems(3)* **by** *auto*
qed
moreover have $Complete_F\ k\ (bins\text{-}upto\ bs\ k\ (i + 1)) \subseteq bins\ bs$
proof −
have $Complete_F\ k\ (bins\text{-}upto\ bs\ k\ (i + 1)) = Complete_F\ k\ (bins\text{-}upto\ bs\ k\ i \cup \{x\})$
using *bins-upto-Suc-Un Pass.hyps(1,2)*
by *(metis items-def length-map not-le-imp-less)*
also have ... $= Complete_F\ k\ (bins\text{-}upto\ bs\ k\ i)$
using *$Complete_F$-Un-eq-terminal Pass.hyps Pass.prems bins-upto-sub-bins subset-iff wf-bins-impl-wf-items wf-item-def wf-bins-kth-bin x wf-earley-input-elim* **by** *(smt (verit, best))*
finally show *?thesis*
using *Pass.prems(1,2) $Complete_F$-$Earley_F$-bin-step-mono wf-earley-input-elim* **by** *blast*
qed
ultimately have $Earley_F\text{-}bin\text{-}step\ k\ \mathcal{G}\ \omega\ (bins\ bs) \subseteq bins\ (Earley_L\text{-}bin'\ k\ \mathcal{G}\ \omega\ bs\ (i+1))$
using *Pass.IH Pass.prems $Earley_F$-bin-step-def bins-upto-sub-bins wf-earley-input-elim*
by *(metis le-sup-iff)*
thus *?case*
using *bins-upd-bins Pass.hyps Pass.prems* **by** *simp*
next
case $(Predict_F\ k\ \mathcal{G}\ \omega\ bs\ i\ x\ a)$
let $?bs' = upd\text{-}bins\ bs\ k\ (Predict_L\ k\ \mathcal{G}\ a)$
have $k \geq length\ \omega \lor \neg\ \omega!k = a$
using *$Predict_F$.hyps(4) $Predict_F$.prems(4) is-word-def*
by *(metis Set.set-insert insert-disjoint(1) not-le-imp-less nth-mem)*
have x: $x \in set\ (items\ (bs\ !\ k))$

using *Predict*$_F$*.hyps*(*1*,*2*) **by** *auto*
hence *sound*: $\forall$ *x* $\in$ *set* (*items*(*Predict*$_L$ *k* $\mathcal{G}$ *a*)). *sound-item* $\mathcal{G}$ ω *x*
using *sound-Predict*$_L$ *Predict*$_F$*.hyps*(*3*) *Predict*$_F$*.prems wf-earley-input-elim wf-bins-impl-wf-items* **by** *fast*
have *wf*: (*k*, $\mathcal{G}$, ω, *?bs'*) $\in$ *wf-earley-input*
using *Predict*$_F$*.hyps Predict*$_F$*.prems*(*1*) *wf-earley-input-Predict*$_L$ **by** *metis*
have *len*: *i* $<$ *length* (*items* (*?bs'* ! *k*))
using *length-nth-upd-bin-bins Predict*$_F$*.hyps*(*1*)
by (*metis dual-order.strict-trans1 items-def length-map linorder-not-less*)
have *Scan*$_F$ *k* ω (*bins-upto ?bs' k* (*i* + *1*)) $\subseteq$ *bins ?bs'*
proof –
have *Scan*$_F$ *k* ω (*bins-upto ?bs' k* (*i* + *1*)) = *Scan*$_F$ *k* ω (*bins-upto ?bs' k i* $\cup$ {*items* (*?bs'* ! *k*) ! *i*})
using *Predict*$_F$*.hyps*(*1*) *bins-upto-Suc-Un* **by** (*metis items-def len length-map*)
also have ... = *Scan*$_F$ *k* ω (*bins-upto bs k i* $\cup$ {*x*})
using *Predict*$_F$*.hyps*(*1*,*2*) *Predict*$_F$*.prems*(*1*) *items-nth-idem-upd-bins bins-upto-kth-nth-idem wf-earley-input-elim*
by (*metis dual-order.refl items-def length-map not-le-imp-less*)
also have ... $\subseteq$ *bins bs* $\cup$ *Scan*$_F$ *k* ω {*x*}
using *Predict*$_F$*.prems*(*2*,*3*) *Scan*$_F$*-Un Scan*$_F$*-Earley*$_F$*-bin-step-mono* **by** *fastforce*
also have ... = *bins bs*
using *Predict*$_F$*.hyps*(*3*) ‹*length* ω $\leq$ *k* $\vee$ ω ! *k* $\neq$ *a*› **by** (*auto simp*: *Scan*$_F$*-def bin-def*)
finally show *?thesis*
using *Predict*$_F$*.prems*(*1*) *wf-earley-input-elim bins-upd-bins* **by** *blast*
qed
moreover have *Predict*$_F$ *k* $\mathcal{G}$ (*bins-upto ?bs' k* (*i* + *1*)) $\subseteq$ *bins ?bs'*
proof –
have *Predict*$_F$ *k* $\mathcal{G}$ (*bins-upto ?bs' k* (*i* + *1*)) = *Predict*$_F$ *k* $\mathcal{G}$ (*bins-upto ?bs' k i* $\cup$ {*items* (*?bs'* ! *k*) ! *i*})
using *Predict*$_F$*.hyps*(*1*) *bins-upto-Suc-Un* **by** (*metis items-def len length-map*)
also have ... = *Predict*$_F$ *k* $\mathcal{G}$ (*bins-upto bs k i* $\cup$ {*x*})
using *Predict*$_F$*.hyps*(*1*,*2*) *Predict*$_F$*.prems*(*1*) *items-nth-idem-upd-bins bins-upto-kth-nth-idem wf-earley-input-elim*
by (*metis dual-order.refl items-def length-map not-le-imp-less*)
also have ... $\subseteq$ *bins bs* $\cup$ *Predict*$_F$ *k* $\mathcal{G}$ {*x*}
using *Predict*$_F$*.prems*(*2*,*3*) *Predict*$_F$*-Un Predict*$_F$*-Earley*$_F$*-bin-step-mono* **by** *fastforce*
also have ... = *bins bs* $\cup$ *set* (*items* (*Predict*$_L$ *k* $\mathcal{G}$ *a*))
using *Predict*$_F$*.hyps Predict*$_F$*.prems*(*1*–*3*) *wf-earley-input-elim*
apply (*auto simp*: *Predict*$_F$*-def Predict*$_L$*-def bin-def items-def*)
using *wf-bins-kth-bin x* **by** *blast*
finally show *?thesis*
using *Predict*$_F$*.prems*(*1*) *wf-earley-input-elim bins-upd-bins* **by** *blast*
qed
moreover have *Complete*$_F$ *k* (*bins-upto ?bs' k* (*i* + *1*)) $\subseteq$ *bins ?bs'*
proof –
have *Complete*$_F$ *k* (*bins-upto ?bs' k* (*i* + *1*)) = *Complete*$_F$ *k* (*bins-upto ?bs' k*

$i \cup \{$*items* (*?bs'* ! *k*) ! *i*$\})$
using *bins-upto-Suc-Un len* **by** (*metis items-def length-map*)
also have ... = $Complete_F$ *k* (*bins-upto bs k i* $\cup \{x\})$
using *items-nth-idem-upd-bins* $Predict_F$*.hyps*(*1*,*2*) $Predict_F$*.prems*(*1*) *bins-upto-kth-nth-idem wf-earley-input-elim*
by (*metis dual-order.refl items-def length-map not-le-imp-less*)
also have ... = $Complete_F$ *k* (*bins-upto bs k i*)
using $Complete_F$*-Un-eq-nonterminal* $Predict_F$*.prems bins-upto-sub-bins* $Predict_F$*.hyps*(*3*)
subset-eq wf-bins-kth-bin x wf-bins-impl-wf-items wf-item-def wf-earley-input-elim
by (*smt* (*verit*, *ccfv-SIG*) *option.simps*(*3*))
also have ... $\subseteq$ *bins bs*
using $Complete_F$*-*$Earley_F$*-bin-step-mono* $Predict_F$*.prems*(*2*) **by** *blast*
finally show *?thesis*
using *bins-upd-bins* $Predict_F$*.prems*(*1*,*2*,*3*) *wf-earley-input-elim* **by** *blast*
qed
ultimately have $Earley_F$*-bin-step k* $\mathcal{G}$ ω (*bins ?bs'*) $\subseteq$ *bins* ($Earley_L$*-bin' k* $\mathcal{G}$ ω *?bs'* (*i*+*1*))
using $Predict_F$*.IH* $Predict_F$*.prems sound wf* $Earley_F$*-bin-step-def bins-upto-sub-bins bins-upd-bins wf-earley-input-elim* **by** (*metis UnE le-supI*)
hence $Earley_F$*-bin-step k* $\mathcal{G}$ ω (*bins ?bs'*) $\subseteq$ *bins* ($Earley_L$*-bin' k* $\mathcal{G}$ ω *bs i*)
using $Predict_F$*.hyps* $Earley_L$*-bin'-simps*(*5*) **by** *simp*
moreover have $Earley_F$*-bin-step k* $\mathcal{G}$ ω (*bins bs*) $\subseteq$ $Earley_F$*-bin-step k* $\mathcal{G}$ ω (*bins ?bs'*)
using $Earley_F$*-bin-step-sub-mono* $Predict_F$*.prems*(*1*) *wf-earley-input-elim bins-upd-bins* **by** (*metis Un-upper1*)
ultimately show *?case*
by *blast*
qed

lemma $Earley_F$*-bin-step-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*
assumes $\forall x \in$ *bins bs. sound-item* $\mathcal{G}$ ω *x is-word* $\mathcal{G}$ ω *nonempty-derives* $\mathcal{G}$
shows $Earley_F$*-bin-step k* $\mathcal{G}$ ω (*bins bs*) $\subseteq$ *bins* ($Earley_L$*-bin k* $\mathcal{G}$ ω *bs*)
using *assms* $Earley_F$*-bin-step-sub-*$Earley_L$*-bin'* $Earley_L$*-bin-def* **by** *metis*

lemma *bins-eq-items-*$Complete_L$:
assumes *bins-eq-items as bs start-item x* < *length as*
shows *items* ($Complete_L$ *k x as i*) = *items* ($Complete_L$ *k x bs i*)
proof −
let *?orig-a* = *as* ! *start-item x*
let *?orig-b* = *bs* ! *start-item x*
have *items ?orig-a* = *items ?orig-b*
using *assms* **by** (*metis* (*no-types*, *opaque-lifting*) *bins-eq-items-def length-map nth-map*)
thus *?thesis*
unfolding $Complete_L$*-def* **by** *simp*
qed

lemma *Earley$_L$-bin′-bins-eq*:
  assumes (k, $\mathcal{G}$, ω, *as*) $\in$ *wf-earley-input*
  assumes *bins-eq-items as bs wf-bins* $\mathcal{G}$ ω *as*
  shows *bins-eq-items* (*Earley$_L$-bin′* k $\mathcal{G}$ ω *as i*) (*Earley$_L$-bin′* k $\mathcal{G}$ ω *bs i*)
  using *assms*
proof (*induction i arbitrary*: *bs rule*: *Earley$_L$-bin′-induct*[*OF assms*(*1*), *case-names Base Complete$_F$ Scan$_F$ Pass Predict$_F$*])
  case (*Base k* $\mathcal{G}$ ω *as i*)
  have *Earley$_L$-bin′* k $\mathcal{G}$ ω *as i* = *as*
    by (*simp add*: *Base.hyps*)
  moreover have *Earley$_L$-bin′* k $\mathcal{G}$ ω *bs i* = *bs*
    using *Base.hyps Base.prems*(*1*,*2*) **unfolding** *bins-eq-items-def*
    by (*metis Earley$_L$-bin′-simps*(*1*) *length-map nth-map wf-earley-input-elim*)
  ultimately show *?case*
    using *Base.prems*(*2*) **by** *presburger*
next
  case (*Complete$_F$ k* $\mathcal{G}$ ω *as i x*)
  let *?as′* = *upd-bins as k* (*Complete$_L$ k x as i*)
  let *?bs′* = *upd-bins bs k* (*Complete$_L$ k x bs i*)
  have *k*: k < *length as*
    using *Complete$_F$.prems*(*1*) *wf-earley-input-elim* **by** *blast*
  hence *wf-x*: *wf-item* $\mathcal{G}$ ω *x*
    using *Complete$_F$.hyps*(*1*,*2*) *Complete$_F$.prems*(*3*) *wf-bins-kth-bin* **by** *fastforce*
  have (k, $\mathcal{G}$, ω, *?as′*) $\in$ *wf-earley-input*
   using *Complete$_F$.hyps Complete$_F$.prems*(*1*) *wf-earley-input-Complete$_L$* **by** *blast*
  moreover have *bins-eq-items ?as′ ?bs′*
    using *Complete$_F$.hyps*(*1*,*2*) *Complete$_F$.prems*(*2*,*3*) *bins-eq-items-dist-upd-bins bins-eq-items-Complete$_L$*
        k wf-x wf-bins-kth-bin wf-item-def **by** (*metis dual-order.strict-trans2 leI nth-mem*)
  ultimately have *bins-eq-items* (*Earley$_L$-bin′* k $\mathcal{G}$ ω *?as′* (*i* + *1*)) (*Earley$_L$-bin′* k $\mathcal{G}$ ω *?bs′* (*i* + *1*))
    using *Complete$_F$.IH wf-earley-input-elim* **by** *blast*
  moreover have *Earley$_L$-bin′* k $\mathcal{G}$ ω *as i* = *Earley$_L$-bin′* k $\mathcal{G}$ ω *?as′* (*i+1*)
    using *Complete$_F$.hyps* **by** *simp*
  moreover have *Earley$_L$-bin′* k $\mathcal{G}$ ω *bs i* = *Earley$_L$-bin′* k $\mathcal{G}$ ω *?bs′* (*i+1*)
    using *Complete$_F$.hyps Complete$_F$.prems* **unfolding** *bins-eq-items-def*
   by (*metis Earley$_L$-bin′-simps*(*2*) *map-eq-imp-length-eq nth-map wf-earley-input-elim*)
  ultimately show *?case*
    by *argo*
next
  case (*Scan$_F$ k* $\mathcal{G}$ ω *as i x a*)
  let *?as′* = *upd-bins as* (*k+1*) (*Scan$_L$ k* ω *a x i*)
  let *?bs′* = *upd-bins bs* (*k+1*) (*Scan$_L$ k* ω *a x i*)
  have (k, $\mathcal{G}$, ω, *?as′*) $\in$ *wf-earley-input*
    using *Scan$_F$.hyps Scan$_F$.prems*(*1*) *wf-earley-input-Scan$_L$* **by** *fast*
  moreover have *bins-eq-items ?as′ ?bs′*
   using *Scan$_F$.hyps*(*5*) *Scan$_F$.prems*(*1*,*2*) *bins-eq-items-dist-upd-bins add-mono1*

wf-earley-input-elim **by** *metis*
ultimately have *bins-eq-items* ($Earley_L$*-bin′ k* $\mathcal{G}$ ω *?as′* ($i + 1$)) ($Earley_L$*-bin′* *k* $\mathcal{G}$ ω *?bs′* ($i + 1$))
using $Scan_F$*.IH wf-earley-input-elim* **by** *blast*
moreover have $Earley_L$*-bin′ k* $\mathcal{G}$ ω *as i* = $Earley_L$*-bin′ k* $\mathcal{G}$ ω *?as′* (i+1)
using $Scan_F$*.hyps* **by** *simp*
moreover have $Earley_L$*-bin′ k* $\mathcal{G}$ ω *bs i* = $Earley_L$*-bin′ k* $\mathcal{G}$ ω *?bs′* (i+1)
using $Scan_F$*.hyps* $Scan_F$*.prems* **unfolding** *bins-eq-items-def*
by (*smt* (*verit*, *ccfv-threshold*) $Earley_L$*-bin′-simps*(*3*) *length-map nth-map wf-earley-input-elim*)
ultimately show *?case*
by *argo*
next
case (*Pass k* $\mathcal{G}$ ω *as i x a*)
have *bins-eq-items* ($Earley_L$*-bin′ k* $\mathcal{G}$ ω *as* ($i + 1$)) ($Earley_L$*-bin′ k* $\mathcal{G}$ ω *bs* ($i + 1$))
using *Pass.prems Pass.IH* **by** *blast*
moreover have $Earley_L$*-bin′ k* $\mathcal{G}$ ω *as i* = $Earley_L$*-bin′ k* $\mathcal{G}$ ω *as* (i+1)
using *Pass.hyps* **by** *simp*
moreover have $Earley_L$*-bin′ k* $\mathcal{G}$ ω *bs i* = $Earley_L$*-bin′ k* $\mathcal{G}$ ω *bs* (i+1)
using *Pass.hyps Pass.prems* **unfolding** *bins-eq-items-def*
by (*metis* $Earley_L$*-bin′-simps*(*4*) *map-eq-imp-length-eq nth-map wf-earley-input-elim*)
ultimately show *?case*
by *argo*
next
case ($Predict_F$ *k* $\mathcal{G}$ ω *as i x a*)
let *?as′* = *upd-bins as k* ($Predict_L$ *k* $\mathcal{G}$ *a*)
let *?bs′* = *upd-bins bs k* ($Predict_L$ *k* $\mathcal{G}$ *a*)
have (*k*, $\mathcal{G}$, ω, *?as′*) $\in$ *wf-earley-input*
using $Predict_F$*.hyps* $Predict_F$*.prems*(*1*) *wf-earley-input-*$Predict_L$ **by** *fast*
moreover have *bins-eq-items ?as′ ?bs′*
using $Predict_F$*.prems*(*1*,*2*) *bins-eq-items-dist-upd-bins wf-earley-input-elim* **by** *blast*
ultimately have *bins-eq-items* ($Earley_L$*-bin′ k* $\mathcal{G}$ ω *?as′* ($i + 1$)) ($Earley_L$*-bin′* *k* $\mathcal{G}$ ω *?bs′* ($i + 1$))
using $Predict_F$*.IH wf-earley-input-elim* **by** *blast*
moreover have $Earley_L$*-bin′ k* $\mathcal{G}$ ω *as i* = $Earley_L$*-bin′ k* $\mathcal{G}$ ω *?as′* (i+1)
using $Predict_F$*.hyps* **by** *simp*
moreover have $Earley_L$*-bin′ k* $\mathcal{G}$ ω *bs i* = $Earley_L$*-bin′ k* $\mathcal{G}$ ω *?bs′* (i+1)
using $Predict_F$*.hyps* $Predict_F$*.prems* **unfolding** *bins-eq-items-def*
by (*metis* $Earley_L$*-bin′-simps*(*5*) *length-map nth-map wf-earley-input-elim*)
ultimately show *?case*
by *argo*
qed

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

using *assms*
proof (*induction i arbitrary*: *j rule*: $Earley_L$*-bin′-induct*[*OF assms*(*1*), *case-names Base* $Complete_F$ $Scan_F$ *Pass* $Predict_F$])
case ($Complete_F$ *k* $\mathcal{G}$ ω *bs i x*)
let *?bs′* = *upd-bins bs k* ($Complete_L$ *k x bs i*)
have *x*: $x \in$ *set* (*items* (*bs* ! *k*))
using $Complete_F$*.hyps*(*1*,*2*) **by** *auto*
have *wf*: (*k*, $\mathcal{G}$, ω, *?bs′*) $\in$ *wf-earley-input*
using $Complete_F$*.hyps* $Complete_F$*.prems*(*1*) *wf-earley-input-*$Complete_L$ **by** *blast*
hence $\forall x \in$ *set* (*items* ($Complete_L$ *k x bs i*)). *sound-item* $\mathcal{G}$ ω *x*
using *sound-*$Complete_L$ $Complete_F$*.hyps*(*3*) $Complete_F$*.prems* *wf-earley-input-elim wf-bins-impl-wf-items x*
by (*metis dual-order.refl*)
hence *sound*: $\forall x \in$ *bins ?bs′*. *sound-item* $\mathcal{G}$ ω *x*
by (*metis* $Complete_F$*.prems*(*1*,*3*) *UnE bins-upd-bins wf-earley-input-elim*)
show *?case*
proof *cases*
assume *i*+*1* $\leq$ *j*
thus *?thesis*
using *wf sound* $Complete_F$ $Earley_L$*-bin′-simps*(*2*) **by** *metis*
next
assume $\neg$ *i*+*1* $\leq$ *j*
hence *i* = *j*
using $Complete_F$*.prems*(*2*) **by** *simp*
have *bins* ($Earley_L$*-bin′ k* $\mathcal{G}$ ω ($Earley_L$*-bin′ k* $\mathcal{G}$ ω *bs i*) *j*) = *bins* ($Earley_L$*-bin′ k* $\mathcal{G}$ ω ($Earley_L$*-bin′ k* $\mathcal{G}$ ω *?bs′* (*i*+*1*)) *j*)
using $Earley_L$*-bin′-simps*(*2*) $Complete_F$*.hyps*(*1*−*3*) **by** *simp*
also have ... = *bins* ($Earley_L$*-bin′ k* $\mathcal{G}$ ω ($Earley_L$*-bin′ k* $\mathcal{G}$ ω *?bs′* (*i*+*1*)) (*j*+*1*))
proof −
let *?bs′′* = $Earley_L$*-bin′ k* $\mathcal{G}$ ω *?bs′* (*i*+*1*)
have *length* (*items* (*?bs′′* ! *k*)) $\geq$ *length* (*items* (*bs* ! *k*))
using *length-nth-bin-*$Earley_L$*-bin′ length-nth-upd-bin-bins order-trans wf* $Complete_F$*.hyps* $Complete_F$*.prems*(*1*)
by (*smt* (*verit*, *ccfv-threshold*) $Earley_L$*-bin′-simps*(*2*))
hence *0*: $\neg$ *length* (*items* (*?bs′′* ! *k*)) $\leq$ *j*
using ‹*i* = *j*› $Complete_F$*.hyps*(*1*) **by** *linarith*
have *x* = *items* (*?bs′* ! *k*) ! *j*
using ‹*i* = *j*› *items-nth-idem-upd-bins* $Complete_F$*.hyps*(*1*,*2*)
by (*metis items-def length-map not-le-imp-less*)
hence *1*: *x* = *items* (*?bs′′* ! *k*) ! *j*
using ‹*i* = *j*› *kth-*$Earley_L$*-bin′-bins* $Complete_F$*.hyps* $Complete_F$*.prems*(*1*) $Earley_L$*-bin′-simps*(*2*) *leI* **by** *metis*
have *bins* ($Earley_L$*-bin′ k* $\mathcal{G}$ ω *?bs′′ j*) = *bins* ($Earley_L$*-bin′ k* $\mathcal{G}$ ω (*upd-bins ?bs′′ k* ($Complete_L$ *k x ?bs′′ i*)) (*j*+*1*))
using $Earley_L$*-bin′-simps*(*2*) *0 1* $Complete_F$*.hyps*(*1*,*3*) $Complete_F$*.prems*(*2*) ‹*i* = *j*› **by** *auto*
moreover have *bins-eq-items* (*upd-bins ?bs′′ k* ($Complete_L$ *k x ?bs′′ i*)) *?bs′′*
proof −

have $k < length\ bs$
using $Complete_F.prems(1)$ *wf-earley-input-elim* **by** *blast*
have 0: $set\ (Complete_L\ k\ x\ bs\ i) = set\ (Complete_L\ k\ x\ ?bs''\ i)$
proof (*cases start-item* $x = k$)
case *True*
thus *?thesis*
using *impossible-complete-item kth-bin-sub-bins* $Complete_F.hyps(3)$ $Complete_F.prems$ *wf-earley-input-elim*
wf-bins-kth-bin x next-symbol-def **by** (*metis option.distinct*(*1*) *subsetD*)
next
case *False*
hence *start-item* $x < k$
using x $Complete_F.prems(1)$ *wf-bins-kth-bin wf-item-def nat-less-le* **by** (*metis wf-earley-input-elim*)
hence bs ! *start-item* $x = ?bs''$! *start-item* x
using *False nth-idem-upd-bins* $nth\text{-}Earley_L\text{-}bin'\text{-}eq$ *wf* **by** *metis*
thus *?thesis*
using $Complete_L\text{-}eq\text{-}start\text{-}item$ **by** *metis*
qed
have $set\ (items\ (Complete_L\ k\ x\ bs\ i)) \subseteq set\ (items\ (?bs'\ !\ k))$
by (*simp add*: ‹$k < length\ bs$› *upd-bins-def set-items-upds-bin*)
hence $set\ (items\ (Complete_L\ k\ x\ ?bs''\ i)) \subseteq set\ (items\ (?bs'\ !\ k))$
using 0 **by** (*simp add*: *items-def*)
also have ... $\subseteq set\ (items\ (?bs''\ !\ k))$
by (*simp add*: *wf* $nth\text{-}bin\text{-}sub\text{-}Earley_L\text{-}bin'$)
finally show *?thesis*
using *bins-eq-items-upd-bins* **by** *blast*
qed
moreover have $(k, \mathcal{G}, \omega, upd\text{-}bins\ ?bs''\ k\ (Complete_L\ k\ x\ ?bs''\ i)) \in$ *wf-earley-input*
using $wf\text{-}earley\text{-}input\text{-}Earley_L\text{-}bin'$ $wf\text{-}earley\text{-}input\text{-}Complete_L$ $Complete_F.hyps$ $Complete_F.prems(1)$
‹$length\ (items\ (bs\ !\ k)) \le length\ (items\ (?bs''\ !\ k))$› $kth\text{-}Earley_L\text{-}bin'\text{-}bins$ $0\ 1$ **by** *blast*
ultimately show *?thesis*
using $Earley_L\text{-}bin'\text{-}bins\text{-}eq$ *bins-eq-items-imp-eq-bins wf-earley-input-elim* **by** *blast*
qed
also have ... $= bins\ (Earley_L\text{-}bin'\ k\ \mathcal{G}\ \omega\ ?bs'\ (i + 1))$
using $Complete_F.IH[OF\ wf\ \text{-}\ sound\ Complete_F.prems(4)]$ ‹$i = j$› **by** *blast*
finally show *?thesis*
using $Complete_F.hyps$ **by** *simp*
qed
next
case $(Scan_F\ k\ \mathcal{G}\ \omega\ bs\ i\ x\ a)$
let $?bs' = upd\text{-}bins\ bs\ (k+1)\ (Scan_L\ k\ \omega\ a\ x\ i)$
have x: $x \in set\ (items\ (bs\ !\ k))$
using $Scan_F.hyps(1,2)$ **by** *auto*
hence $\forall x \in set\ (items\ (Scan_L\ k\ \omega\ a\ x\ i)).\ sound\text{-}item\ \mathcal{G}\ \omega\ x$

using *sound-Scan*$_L$ *Scan*$_F$*.hyps*(*3*,*5*) *Scan*$_F$*.prems*(*1*,*2*,*3*) *wf-earley-input-elim wf-bins-impl-wf-items x*
by (*metis dual-order.refl*)
hence *sound*: $\forall\, x \in$ *bins ?bs'*. *sound-item* $\mathcal{G}$ ω *x*
using *Scan*$_F$*.hyps*(*5*) *Scan*$_F$*.prems*(*1*,*3*) *bins-upd-bins wf-earley-input-elim*
by (*metis UnE add-less-cancel-right*)
have *wf*: (*k*, $\mathcal{G}$, ω, *?bs'*) $\in$ *wf-earley-input*
using *Scan*$_F$*.hyps Scan*$_F$*.prems*(*1*) *wf-earley-input-Scan*$_L$ **by** *metis*
show *?case*
proof *cases*
assume $i+1 \le j$
thus *?thesis*
using *sound Scan*$_F$ **by** (*metis Earley*$_L$*-bin'-simps*(*3*) *wf-earley-input-Scan*$_L$)
next
assume $\neg$ $i+1 \le j$
hence $i = j$
using *Scan*$_F$*.prems*(*2*) **by** *auto*
have *bins* (*Earley*$_L$*-bin' k* $\mathcal{G}$ ω (*Earley*$_L$*-bin' k* $\mathcal{G}$ ω *bs i*) *j*) = *bins* (*Earley*$_L$*-bin' k* $\mathcal{G}$ ω (*Earley*$_L$*-bin' k* $\mathcal{G}$ ω *?bs'* (*i*+*1*)) *j*)
using *Scan*$_F$*.hyps* **by** *simp*
also have ... = *bins* (*Earley*$_L$*-bin' k* $\mathcal{G}$ ω (*Earley*$_L$*-bin' k* $\mathcal{G}$ ω *?bs'* (*i*+*1*)) (*j*+*1*))
proof −
let *?bs''* = *Earley*$_L$*-bin' k* $\mathcal{G}$ ω *?bs'* (*i*+*1*)
have *length* (*items* (*?bs''* ! *k*)) $\ge$ *length* (*items* (*bs* ! *k*))
using *length-nth-bin-Earley*$_L$*-bin' length-nth-upd-bin-bins order-trans Scan*$_F$*.hyps Scan*$_F$*.prems*(*1*) *Earley*$_L$*-bin'-simps*(*3*)
by (*smt* (*verit*, *ccfv-SIG*))
hence *bins* (*Earley*$_L$*-bin' k* $\mathcal{G}$ ω *?bs'' j*) = *bins* (*Earley*$_L$*-bin' k* $\mathcal{G}$ ω (*upd-bins ?bs''* (*k*+*1*) (*Scan*$_L$ *k* ω *a x i*)) (*j*+*1*))
using ‹$i = j$› *kth-Earley*$_L$*-bin'-bins nth-idem-upd-bins Earley*$_L$*-bin'-simps*(*3*) *Scan*$_F$*.hyps Scan*$_F$*.prems*(*1*) **by** (*smt* (*verit*, *best*) *leI le-trans*)
moreover have *bins-eq-items* (*upd-bins ?bs''* (*k*+*1*) (*Scan*$_L$ *k* ω *a x i*)) *?bs''*
proof −
have $k+1 <$ *length bs*
using *Scan*$_F$*.hyps*(*5*) *Scan*$_F$*.prems wf-earley-input-elim* **by** *fastforce*+
hence *set* (*items* (*Scan*$_L$ *k* ω *a x i*)) $\subseteq$ *set* (*items* (*?bs'* ! (*k*+*1*)))
by (*simp add*: *upd-bins-def set-items-upds-bin*)
also have ... $\subseteq$ *set* (*items* (*?bs''* ! (*k*+*1*)))
using *wf nth-bin-sub-Earley*$_L$*-bin'* **by** *blast*
finally show *?thesis*
using *bins-eq-items-upd-bins* **by** *blast*
qed
moreover have (*k*, $\mathcal{G}$, ω, *upd-bins ?bs''* (*k*+*1*) (*Scan*$_L$ *k* ω *a x i*)) $\in$ *wf-earley-input*
using *wf-earley-input-Earley*$_L$*-bin' wf-earley-input-Scan*$_L$ *Scan*$_F$*.hyps Scan*$_F$*.prems*(*1*) ‹*length* (*items* (*bs* ! *k*)) $\le$ *length* (*items* (*?bs''* ! *k*))› *kth-Earley*$_L$*-bin'-bins*
by (*smt* (*verit*, *ccfv-SIG*) *Earley*$_L$*-bin'-simps*(*3*) *linorder-not-le order.trans*)
ultimately show *?thesis*

using *Earley$_L$-bin$'$-bins-eq bins-eq-items-imp-eq-bins wf-earley-input-elim* **by** *blast*
qed
also have ... = *bins* (*Earley$_L$-bin$'$ k $\mathcal{G}$ ω ?bs$'$* (*i* + *1*))
using ‹*i* = *j*› *Scan$_F$.IH Scan$_F$.prems Scan$_F$.hyps sound wf-earley-input-Scan$_L$* **by** *fast*
finally show *?thesis*
using *Scan$_F$.hyps* **by** *simp*
qed
next
case (*Pass k $\mathcal{G}$ ω bs i x a*)
show *?case*
proof *cases*
assume *i+1* $\leq$ *j*
thus *?thesis*
using *Pass* **by** (*metis Earley$_L$-bin$'$-simps*(*4*))
next
assume $\neg$ *i+1* $\leq$ *j*
show *?thesis*
using *Pass Earley$_L$-bin$'$-simps*(*1*,*4*) *kth-Earley$_L$-bin$'$-bins* **by** (*metis Suc-eq-plus1 Suc-leI antisym-conv2 not-le-imp-less*)
qed
next
case (*Predict$_F$ k $\mathcal{G}$ ω bs i x a*)
let *?bs$'$* = *upd-bins bs k* (*Predict$_L$ k $\mathcal{G}$ a*)
have *x*: *x* $\in$ *set* (*items* (*bs* ! *k*))
using *Predict$_F$.hyps*(*1*,*2*) **by** *auto*
hence $\forall$ *x* $\in$ *set* (*items*(*Predict$_L$ k $\mathcal{G}$ a*)). *sound-item $\mathcal{G}$ ω x*
using *sound-Predict$_L$ Predict$_F$.hyps*(*3*) *Predict$_F$.prems wf-earley-input-elim wf-bins-impl-wf-items* **by** *fast*
hence *sound*: $\forall$ *x* $\in$ *bins ?bs$'$*. *sound-item $\mathcal{G}$ ω x*
using *Predict$_F$.prems*(*1*,*3*) *UnE bins-upd-bins wf-earley-input-elim* **by** *metis*
have *wf*: (*k*, $\mathcal{G}$, ω, *?bs$'$*) $\in$ *wf-earley-input*
using *Predict$_F$.hyps Predict$_F$.prems*(*1*) *wf-earley-input-Predict$_L$* **by** *metis*
have *len*: *i* < *length* (*items* (*?bs$'$* ! *k*))
using *length-nth-upd-bin-bins Predict$_F$.hyps*(*1*) *Orderings.preorder-class.dual-order.strict-trans1 linorder-not-less*
by (*metis items-def length-map*)
show *?case*
proof *cases*
assume *i+1* $\leq$ *j*
thus *?thesis*
using *sound wf Predict$_F$* **by** (*metis Earley$_L$-bin$'$-simps*(*5*))
next
assume $\neg$ *i+1* $\leq$ *j*
hence *i* = *j*
using *Predict$_F$.prems*(*2*) **by** *auto*
have *bins* (*Earley$_L$-bin$'$ k $\mathcal{G}$ ω* (*Earley$_L$-bin$'$ k $\mathcal{G}$ ω bs i*) *j*) = *bins* (*Earley$_L$-bin$'$ k $\mathcal{G}$ ω* (*Earley$_L$-bin$'$ k $\mathcal{G}$ ω ?bs$'$* (*i+1*)) *j*)

using $Predict_F.hyps$ **by** *simp*
also have ... = *bins* ($Earley_L$*-bin'* k $\mathcal{G}$ ω ($Earley_L$*-bin'* k $\mathcal{G}$ ω *?bs'* (i+*1*)) (j+*1*))
proof −
let *?bs''* = $Earley_L$*-bin'* k $\mathcal{G}$ ω *?bs'* (i+*1*)
have *length* (*items* (*?bs''* ! k)) ≥ *length* (*items* (*bs* ! k))
using *length-nth-bin-*$Earley_L$*-bin' length-nth-upd-bin-bins order-trans wf*
by (*metis* (*no-types*, *lifting*) *items-def length-map*)
hence *bins* ($Earley_L$*-bin'* k $\mathcal{G}$ ω *?bs''* j) = *bins* ($Earley_L$*-bin'* k $\mathcal{G}$ ω (*upd-bins* *?bs''* k ($Predict_L$ k $\mathcal{G}$ a)) (j+*1*))
using ‹$i = j$› *kth-*$Earley_L$*-bin'-bins nth-idem-upd-bins* $Earley_L$*-bin'-simps*(*5*) $Predict_F$*.hyps* $Predict_F$*.prems*(*1*) *length-bins-*$Earley_L$*-bin'*
wf-bins-$Earley_L$*-bin' wf-bins-kth-bin wf-item-def x* **by** (*smt* (*verit*, *ccfv-SIG*) *linorder-not-le order.trans*)
moreover have *bins-eq-items* (*upd-bins* *?bs''* k ($Predict_L$ k $\mathcal{G}$ a)) *?bs''*
proof −
have k < *length bs*
using *wf-earley-input-elim*[*OF* $Predict_F$*.prems*(*1*)] **by** *blast*
hence *set* (*items* ($Predict_L$ k $\mathcal{G}$ a)) ⊆ *set* (*items* (*?bs'* ! k))
by (*simp add*: *upd-bins-def set-items-upds-bin*)
also have ... ⊆ *set* (*items* (*?bs''* ! k))
using *wf nth-bin-sub-*$Earley_L$*-bin'* **by** *blast*
finally show *?thesis*
using *bins-eq-items-upd-bins* **by** *blast*
qed
moreover have (k, $\mathcal{G}$, ω, *upd-bins* *?bs''* k ($Predict_L$ k $\mathcal{G}$ a)) ∈ *wf-earley-input*
using *wf-earley-input-*$Earley_L$*-bin' wf-earley-input-*$Predict_L$ $Predict_F$*.hyps* $Predict_F$*.prems*(*1*)
‹*length* (*items* (*bs* ! k)) ≤ *length* (*items* (*?bs''* ! k))› *kth-*$Earley_L$*-bin'-bins*
by (*smt* (*verit*, *best*) $Earley_L$*-bin'-simps*(*5*) *dual-order.trans not-le-imp-less*)
ultimately show *?thesis*
using $Earley_L$*-bin'-bins-eq bins-eq-items-imp-eq-bins wf-earley-input-elim* **by** *blast*
qed
also have ... = *bins* ($Earley_L$*-bin'* k $\mathcal{G}$ ω *?bs'* (i + *1*))
using ‹$i = j$› $Predict_F$*.IH* $Predict_F$*.prems sound wf* **by** (*metis order-refl*)
finally show *?thesis*
using $Predict_F$*.hyps* **by** *simp*
qed
qed *simp*

lemma $Earley_L$*-bin-idem*:
assumes (k, $\mathcal{G}$, ω, *bs*) ∈ *wf-earley-input*
assumes ∀ x ∈ *bins bs*. *sound-item* $\mathcal{G}$ ω x *nonempty-derives* $\mathcal{G}$
shows *bins* ($Earley_L$*-bin* k $\mathcal{G}$ ω ($Earley_L$*-bin* k $\mathcal{G}$ ω *bs*)) = *bins* ($Earley_L$*-bin* k $\mathcal{G}$ ω *bs*)
using *assms* $Earley_L$*-bin'-idem* $Earley_L$*-bin-def le0* **by** *metis*

lemma *funpower-*$Earley_F$*-bin-step-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 *funpower* (*Earley*$_F$*-bin-step k* $\mathcal{G}$ ω) *n* (*bins bs*) $\subseteq$ *bins* (*Earley*$_L$*-bin k* $\mathcal{G}$ ω *bs*)
using *assms*
proof (*induction n*)
case *0*
thus *?case*
using *Earley*$_L$*-bin′-mono Earley*$_L$*-bin-def* **by** (*simp add*: *Earley*$_L$*-bin′-mono Earley*$_L$*-bin-def*)
next
case (*Suc n*)
have *0*: *Earley*$_F$*-bin-step k* $\mathcal{G}$ ω (*bins-upto* (*Earley*$_L$*-bin k* $\mathcal{G}$ ω *bs*) *k 0*) $\subseteq$ *bins* (*Earley*$_L$*-bin k* $\mathcal{G}$ ω *bs*)
using *Earley*$_L$*-bin′-mono bins-upto-k0-Earley*$_L$*-bin′-eq assms*(*1*,*2*) *Earley*$_L$*-bin-def order-trans*
by (*metis* (*no-types*, *lifting*))
have *funpower* (*Earley*$_F$*-bin-step k* $\mathcal{G}$ ω) (*Suc n*) (*bins bs*) $\subseteq$ *Earley*$_F$*-bin-step k* $\mathcal{G}$ ω (*bins* (*Earley*$_L$*-bin k* $\mathcal{G}$ ω *bs*))
using *Earley*$_F$*-bin-step-sub-mono Suc* **by** (*metis funpower.simps*(*2*))
also have ... $\subseteq$ *bins* (*Earley*$_L$*-bin k* $\mathcal{G}$ ω (*Earley*$_L$*-bin k* $\mathcal{G}$ ω *bs*))
using *Earley*$_F$*-bin-step-sub-Earley*$_L$*-bin Suc.prems wf-bins-Earley*$_L$*-bin sound-Earley*$_L$*-bin 0 wf-earley-input-Earley*$_L$*-bin* **by** *blast*
also have ... $\subseteq$ *bins* (*Earley*$_L$*-bin k* $\mathcal{G}$ ω *bs*)
using *Earley*$_L$*-bin-idem Suc.prems* **by** *blast*
finally show *?case* **.**
qed

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*)
using *assms funpower-Earley*$_F$*-bin-step-sub-Earley*$_L$*-bin Earley*$_F$*-bin-def elem-limit-simp*
by *fastforce*

lemma *Earley*$_F$*-bins-sub-Earley*$_L$*-bins*:
assumes $k \leq$ *length* ω
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)
using *assms*
proof (*induction k*)
case *0*
hence *Earley*$_F$*-bin 0* $\mathcal{G}$ ω (*Init*$_F$ $\mathcal{G}$) $\subseteq$ *bins* (*Earley*$_L$*-bin 0* $\mathcal{G}$ ω (*Init*$_L$ $\mathcal{G}$ ω))
using *Earley*$_F$*-bin-sub-Earley*$_L$*-bin Init*$_L$*-eq-Init*$_F$ *length-bins-Init*$_L$ *Init*$_L$*-eq-Init*$_F$ *sound-Init bins-upto-empty*

$Earley_F$*-bin-step-empty bins-upto-sub-bins wf-earley-input-Init*$_L$ *wf-earley-input-elim*
by (*smt* (*verit*, *ccfv-threshold*) *Init*$_F$*-sub-Earley basic-trans-rules*(*31*) *sound-Earley wf-bins-impl-wf-items*)
thus *?case*
by *simp*
next
case (*Suc k*)
have *wf*: (*Suc k*, $\mathcal{G}$, ω, *Earley*$_L$*-bins k* $\mathcal{G}$ ω) $\in$ *wf-earley-input*
by (*simp add*: *Suc.prems*(*1*) *Suc-leD assms*(*2*) *wf-earley-input-intro*)
have *sub*: *Earley*$_F$*-bin-step* (*Suc k*) $\mathcal{G}$ ω (*bins-upto* (*Earley*$_L$*-bins k* $\mathcal{G}$ ω) (*Suc k*) *0*) $\subseteq$ *bins* (*Earley*$_L$*-bins k* $\mathcal{G}$ ω)
proof −
have *bin* (*bins-upto* (*Earley*$_L$*-bins k* $\mathcal{G}$ ω) (*Suc k*) *0*) (*Suc k*) = {}
using *kth-bin-bins-upto-empty wf Suc.prems wf-earley-input-elim* **by** *blast*
hence *Earley*$_F$*-bin-step* (*Suc k*) $\mathcal{G}$ ω (*bins-upto* (*Earley*$_L$*-bins k* $\mathcal{G}$ ω) (*Suc k*) *0*) = *bins-upto* (*Earley*$_L$*-bins k* $\mathcal{G}$ ω) (*Suc k*) *0*
unfolding *Earley*$_F$*-bin-step-def Scan*$_F$*-def Complete*$_F$*-def Predict*$_F$*-def bin-def* **by** *blast*
also have ... $\subseteq$ *bins* (*Earley*$_L$*-bins k* $\mathcal{G}$ ω)
using *wf Suc.prems bins-upto-sub-bins wf-earley-input-elim* **by** *blast*
finally show *?thesis* **.**
qed
have *sound*: $\forall x \in$ *bins* (*Earley*$_L$*-bins k* $\mathcal{G}$ ω). *sound-item* $\mathcal{G}$ ω x
using *Suc Earley*$_L$*-bins-sub-Earley*$_F$*-bins* **by** (*metis Suc-leD Earley*$_F$*-bins-sub-Earley in-mono sound-Earley wf-Earley*)
have *Earley*$_F$*-bins* (*Suc k*) $\mathcal{G}$ ω $\subseteq$ *Earley*$_F$*-bin* (*Suc k*) $\mathcal{G}$ ω (*bins* (*Earley*$_L$*-bins k* $\mathcal{G}$ ω))
using *Suc Earley*$_F$*-bin-sub-mono* **by** *simp*
also have ... $\subseteq$ *bins* (*Earley*$_L$*-bin* (*Suc k*) $\mathcal{G}$ ω (*Earley*$_L$*-bins k* $\mathcal{G}$ ω))
using *Earley*$_F$*-bin-sub-Earley*$_L$*-bin wf sub sound Suc.prems* **by** *fastforce*
finally show *?case*
by *simp*
qed

lemma *Earley*$_F$*-sub-Earley*$_L$:
assumes *is-word* $\mathcal{G}$ ω ε*-free* $\mathcal{G}$
shows *Earley*$_F$ $\mathcal{G}$ ω $\subseteq$ *bins* (*Earley*$_L$ $\mathcal{G}$ ω)
using *assms Earley*$_F$*-bins-sub-Earley*$_L$*-bins Earley*$_F$*-def Earley*$_L$*-def*
by (*metis* ε*-free-impl-nonempty-derives dual-order.refl*)

theorem *completeness-Earley*$_L$:
assumes $\mathcal{G} \vdash [\mathfrak{S}\ \mathcal{G}] \Rightarrow^* \omega$ *is-word* $\mathcal{G}$ ω ε*-free* $\mathcal{G}$
shows *recognizing* (*bins* (*Earley*$_L$ $\mathcal{G}$ ω)) $\mathcal{G}$ ω
using *assms Earley*$_F$*-sub-Earley*$_L$ *Earley*$_L$*-sub-Earley*$_F$ *completeness-Earley*$_F$ **by** (*metis subset-antisym*)

8.8 Correctness

theorem *Earley-eq-Earley*$_L$:

assumes *is-word* $\mathcal{G}$ ω *ε-free* $\mathcal{G}$
shows *Earley* $\mathcal{G}$ ω = *bins* (*Earley*$_L$ $\mathcal{G}$ ω)
using *assms Earley*$_F$*-sub-Earley*$_L$ *Earley*$_L$*-sub-Earley*$_F$ *Earley-eq-Earley*$_F$ **by** *blast*

lemma *correctness-recognizer*:
assumes *is-word* $\mathcal{G}$ ω *ε-free* $\mathcal{G}$
shows *recognizer* $\mathcal{G}$ ω $\longleftrightarrow$ $\mathcal{G} \vdash [\mathfrak{S}\ \mathcal{G}] \Rightarrow^* \omega$ (**is** *?L* $\longleftrightarrow$ *?R*)
proof *standard*
assume *?L*
then obtain x **where** $x \in$ *set* (*items* (*Earley*$_L$ $\mathcal{G}$ ω ! *length* ω)) *is-finished* $\mathcal{G}$ ω x
using *assms*(*1*) **unfolding** *recognizer-def* **by** *blast*
moreover have $x \in$ *bins* (*Earley*$_L$ $\mathcal{G}$ ω)
using *assms*(*2*) *kth-bin-sub-bins* ‹$x \in$ *set* (*items* (*Earley*$_L$ $\mathcal{G}$ ω ! *length* ω))›
by (*metis* (*no-types*, *lifting*) *Earley*$_L$*-def dual-order.refl length-Earley*$_L$*-bins length-bins-Init*$_L$ *less-add-one subsetD*)
ultimately show *?R*
using *recognizing-def soundness-Earley*$_L$ **by** *blast*
next
assume *?R*
thus *?L*
using *assms wf-item-in-kth-bin recognizing-def is-finished-def*
by (*metis completeness-Earley*$_L$ *recognizer-def wf-bins-Earley*$_L$)
qed

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

9 Earley parser

9.1 Pointer lemmas

definition *predicts* :: *'a item* ⇒ *bool* **where**
predicts x ≡ *start-item* x = *end-item* x ∧ *dot-item* x = *0*

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

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

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

definition *sound-pre-ptr* :: $'a$ *list* $\Rightarrow$ $'a$ *bins* $\Rightarrow$ *nat* $\Rightarrow$ $'a$ *item* $\times$ *pointer* $\Rightarrow$ *bool* **where**
sound-pre-ptr ω *bs k e* $\equiv$ $\forall$ *pre*. *snd e* $=$ *Pre pre* $\longrightarrow$
$k > 0 \wedge pre <$ *length* $(bs!(k-1)) \wedge$ *scans* ω k (*fst* $(bs!(k-1)!pre)$) (*fst e*)

definition *sound-prered-ptr* :: $'a$ *bins* $\Rightarrow$ *nat* $\Rightarrow$ $'a$ *item* $\times$ *pointer* $\Rightarrow$ *bool* **where**
sound-prered-ptr bs k e $\equiv$ $\forall$ *p ps k' pre red*. *snd e* $=$ *PreRed p ps* $\wedge$ (k', pre, red) $\in$ *set* $(p\#ps)$ $\longrightarrow$
$k' < k \wedge pre <$ *length* $(bs!k') \wedge red <$ *length* $(bs!k) \wedge$ *completes k* (*fst* $(bs!k'!pre)$) (*fst e*) (*fst* $(bs!k!red)$)

definition *sound-ptrs* :: $'a$ *list* $\Rightarrow$ $'a$ *bins* $\Rightarrow$ *bool* **where**
sound-ptrs ω *bs* $\equiv$ $\forall$ $k <$ *length bs*. $\forall$ $e \in$ *set* $(bs!k)$.
sound-null-ptr e $\wedge$ *sound-pre-ptr* ω *bs k e* $\wedge$ *sound-prered-ptr bs k e*

definition *mono-red-ptr* :: $'a$ *bins* $\Rightarrow$ *bool* **where**
mono-red-ptr bs $\equiv$ $\forall$ $k <$ *length bs*. $\forall$ $i <$ *length* $(bs!k)$.
$\forall$ *k' pre red ps*. *snd* $(bs!k!i)$ $=$ *PreRed* (k', pre, red) *ps* $\longrightarrow$ $red < i$

lemma *nth-item-upd-bin*:
$n <$ *length es* $\Longrightarrow$ *fst* (*upd-bin e es* ! n) $=$ *fst* $(es!n)$
by (*induction es arbitrary*: *e n*) (*auto simp*: *less-Suc-eq-0-disj split*: *prod.splits pointer.splits*)

lemma *upd-bin-append*:
fst e $\notin$ *set* (*items es*) $\Longrightarrow$ *upd-bin e es* $=$ *es* @ $[e]$
by (*induction es arbitrary*: *e*) (*auto simp*: *items-def split*: *prod.splits pointer.splits*)

lemma *upd-bin-null-pre*:
fst e $\in$ *set* (*items es*) $\Longrightarrow$ *snd e* $=$ *Null* $\vee$ *snd e* $=$ *Pre pre* $\Longrightarrow$ *upd-bin e es* $=$ *es*
by (*induction es arbitrary*: *e*) (*auto simp*: *items-def split*: *prod.splits*, *fastforce*+)

lemma *upd-bin-prered-nop*:
assumes *distinct* (*items es*) $i <$ *length es*
assumes *fst e* $=$ *fst* $(es!i)$ *snd e* $=$ *PreRed p ps* $\nexists$ *p ps*. *snd* $(es!i)$ $=$ *PreRed p ps*
shows *upd-bin e es* $=$ *es*
using *assms*
by (*induction es arbitrary*: *e i*) (*auto simp*: *less-Suc-eq-0-disj items-def split*: *prod.splits pointer.splits*)

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

using *assms*
proof (*induction es arbitrary*: *e i es'*)
case (*Cons e' es*)
show *?case*
proof *cases*
assume ∗: *fst e = fst e'*
show *?thesis*
proof (*cases* $\exists$ *x xp xs y yp ys. e* = (*x*, *PreRed xp xs*) $\wedge$ *e'* = (*y*, *PreRed yp ys*))
case *True*
then obtain *x xp xs y yp ys* **where** *ee'*: *e* = (*x*, *PreRed xp xs*) *e'* = (*y*, *PreRed yp ys*) *x = y*
using ∗ **by** *auto*
have *simp*: *upd-bin e* (*e'* # *es'*) = (*x*, *PreRed yp* (*xp* # *xs* @ *ys*)) # *es'*
using *True ee'* **by** *simp*
show *?thesis*
using *Cons simp ee'* **apply** (*auto simp*: *items-def*)
using *less-Suc-eq-0-disj* **by** *fastforce+*
next
case *False*
hence *upd-bin e* (*e'* # *es'*) = *e'* # *es'*
using ∗ **by** (*auto split*: *pointer.splits prod.splits*)
thus *?thesis*
using *False* ∗ *Cons.prems*(*1*,*2*,*3*,*4*,*5*) **by** (*auto simp*: *less-Suc-eq-0-disj items-def split*: *prod.splits*)
qed
next
assume ∗: *fst e* $\neq$ *fst e'*
have *simp*: *upd-bin e* (*e'* # *es*) = *e'* # *upd-bin e es*
using ∗ **by** (*auto split*: *pointer.splits prod.splits*)
have *0*: *distinct* (*items es*)
using *Cons.prems*(*1*) **unfolding** *items-def* **by** *simp*
have *1*: *i−1* < *length es*
using *Cons.prems*(*2*,*3*) ∗ **by** (*metis One-nat-def leI less-diff-conv2 less-one list.size*(*4*) *nth-Cons-0*)
have *2*: *fst e = fst* (*es*!(*i−1*))
using *Cons.prems*(*3*) ∗ **by** (*metis nth-Cons'*)
have *3*: *snd e = PreRed p rs*
using *Cons.prems*(*4*) **by** *simp*
have *4*: *snd* (*es*!(*i−1*)) = *PreRed p' rs'*
using *Cons.prems*(*3*,*5*) ∗ **by** (*metis nth-Cons'*)
have *snd* (*upd-bin e es*!(*i−1*)) = *PreRed p'* (*p* # *rs* @ *rs'*) $\wedge$
($\forall$ *j* < *length* (*upd-bin e es*). *i−1* $\neq$ *j* $\longrightarrow$ (*upd-bin e es*) ! *j* = *es* ! *j*)
using *Cons.IH*[*OF 0 1 2 3 4*] **by** *blast*
hence *snd* ((*e'* # *upd-bin e es*) ! *i*) = *PreRed p'* (*p* # *rs* @ *rs'*) $\wedge$
($\forall$ *j* < *length* (*e'* # *upd-bin e es*). *i* $\neq$ *j* $\longrightarrow$ (*e'* # *upd-bin e es*) ! *j* = (*e'* # *es*) ! *j*)
using ∗ *Cons.prems*(*2*,*3*) *less-Suc-eq-0-disj* **by** *auto*
moreover have *e'* # *upd-bin e es* = *es'*
using *Cons.prems*(*6*) *simp* **by** *auto*

ultimately show *?thesis*
by (*metis 0 1 2 3 4 Cons.IH Cons.prems*(*6*) *length-Cons*)
qed
qed *simp*

lemma *sound-ptrs-upd-bin*:
assumes *sound-ptrs* ω *bs* $k <$ *length bs es* $=$ *bs*!*k distinct* (*items es*)
assumes *sound-null-ptr e sound-pre-ptr* ω *bs k e sound-prered-ptr bs k e*
shows *sound-ptrs* ω (*bs*[*k* := *upd-bin e es*])
unfolding *sound-ptrs-def*
proof (*standard*, *standard*, *standard*)
fix *idx elem*
let *?bs* = *bs*[*k* := *upd-bin e es*]
assume *a0*: *idx* < *length ?bs*
assume *a1*: *elem* $\in$ *set* (*?bs* ! *idx*)
show *sound-null-ptr elem* $\wedge$ *sound-pre-ptr* ω *?bs idx elem* $\wedge$ *sound-prered-ptr ?bs idx elem*
proof *cases*
assume *a2*: *idx* = *k*
have *elem* $\in$ *set es* $\Longrightarrow$ *sound-pre-ptr* ω *bs idx elem*
using *a0 a2 assms*(*1*−*3*) *sound-ptrs-def* **by** *blast*
hence *pre-es*: *elem* $\in$ *set es* $\Longrightarrow$ *sound-pre-ptr* ω *?bs idx elem*
using *a2* **unfolding** *sound-pre-ptr-def* **by** *force*
have *elem* = *e* $\Longrightarrow$ *sound-pre-ptr* ω *bs idx elem*
using *a2 assms*(*6*) **by** *auto*
hence *pre-e*: *elem* = *e* $\Longrightarrow$ *sound-pre-ptr* ω *?bs idx elem*
using *a2* **unfolding** *sound-pre-ptr-def* **by** *force*
have *elem* $\in$ *set es* $\Longrightarrow$ *sound-prered-ptr bs idx elem*
using *a0 a2 assms*(*1*−*3*) *sound-ptrs-def* **by** *blast*
hence *prered-es*: *elem* $\in$ *set es* $\Longrightarrow$ *sound-prered-ptr* (*bs*[*k* := *upd-bin e es*]) *idx elem*
using *a2 assms*(*2*,*3*) *length-upd-bin nth-item-upd-bin* **unfolding** *sound-prered-ptr-def*
by (*smt* (*verit*, *ccfv-SIG*) *dual-order.strict-trans1 nth-list-update*)
have *elem* = *e* $\Longrightarrow$ *sound-prered-ptr bs idx elem*
using *a2 assms*(*7*) **by** *auto*
hence *prered-e*: *elem* = *e* $\Longrightarrow$ *sound-prered-ptr ?bs idx elem*
using *a2 assms*(*2*,*3*) *length-upd-bin nth-item-upd-bin* **unfolding** *sound-prered-ptr-def*
by (*smt* (*verit*, *best*) *dual-order.strict-trans1 nth-list-update*)
consider (*A*) *fst e* $\notin$ *set* (*items es*) |
(*B*) *fst e* $\in$ *set* (*items es*) $\wedge$ ($\exists$ *pre*. *snd e* = *Null* $\vee$ *snd e* = *Pre pre*) |
(*C*) *fst e* $\in$ *set* (*items es*) $\wedge$ $\neg$ ($\exists$ *pre*. *snd e* = *Null* $\vee$ *snd e* = *Pre pre*)
by *blast*
thus *?thesis*
proof *cases*
case *A*
hence *elem* $\in$ *set* (*es* @ [*e*])
using *a1 a2 upd-bin-append assms*(*2*) **by** *fastforce*
thus *?thesis*
using *assms*(*1*−*3*,*5*) *pre-e pre-es prered-e prered-es sound-ptrs-def* **by** *auto*

next
case *B*
hence *elem* ∈ *set es*
using *a1 a2 upd-bin-null-pre assms*(*2*) **by** *fastforce*
thus *?thesis*
using *assms*(*1*−*3*) *pre-es prered-es sound-ptrs-def* **by** *blast*
next
case *C*
then obtain *i p ps* **where** *C*: *i* < *length es* ∧ *fst e* = *fst* (*es*!*i*) ∧ *snd e* = *PreRed p ps*
by (*metis assms*(*4*) *distinct-Ex1 items-def length-map nth-map pointer.exhaust*)
show *?thesis*
proof *cases*
assume ∄ *p′ ps′*. *snd* (*es*!*i*) = *PreRed p′ ps′*
hence *C*: *elem* ∈ *set es*
using *a1 a2 C upd-bin-prered-nop assms*(*2*,*4*) **by** (*metis nth-list-update-eq*)
thus *?thesis*
using *assms*(*1*−*3*) *sound-ptrs-def pre-es prered-es* **by** *blast*
next
assume ¬ (∄ *p′ ps′*. *snd* (*es*!*i*) = *PreRed p′ ps′*)
then obtain *p′ ps′* **where** *D*: *snd* (*es*!*i*) = *PreRed p′ ps′*
by *blast*
hence *0*: *snd* (*upd-bin e es*!*i*) = *PreRed p′* (*p*#*ps*@*ps′*) ∧ (∀ *j* < *length* (*upd-bin e es*). *i*≠*j* ⟶ *upd-bin e es*!*j* = *es*!*j*)
using *C assms*(*4*) *upd-bin-prered-upd* **by** *blast*
obtain *j* **where** *1*: *j* < *length es* ∧ *elem* = *upd-bin e es*!*j*
using *a1 a2 assms*(*2*) *C items-def bin-eq-items-upd-bin* **by** (*metis in-set-conv-nth length-map nth-list-update-eq nth-map*)
show *?thesis*
proof *cases*
assume *a3*: *i*=*j*
hence *a3*: *snd elem* = *PreRed p′* (*p*#*ps*@*ps′*)
using *0 1* **by** *blast*
have *sound-null-ptr elem*
using *a3* **unfolding** *sound-null-ptr-def* **by** *simp*
moreover have *sound-pre-ptr ω ?bs idx elem*
using *a3* **unfolding** *sound-pre-ptr-def* **by** *simp*
moreover have *sound-prered-ptr ?bs idx elem*
unfolding *sound-prered-ptr-def*
proof (*standard*, *standard*, *standard*, *standard*, *standard*, *standard*)
fix *P PS k′ pre red*
assume *a4*: *snd elem* = *PreRed P PS* ∧ (*k′*, *pre*, *red*) ∈ *set* (*P*#*PS*)
show *k′* < *idx* ∧ *pre* < *length* (*bs*[*k* := *upd-bin e es*]!*k′*) ∧ *red* < *length* (*bs*[*k* := *upd-bin e es*]!*idx*) ∧
completes idx (*fst* (*bs*[*k* := *upd-bin e es*]!*k′*!*pre*)) (*fst elem*) (*fst* (*bs*[*k* := *upd-bin e es*]!*idx*!*red*))
proof *cases*
assume *a5*: (*k′*, *pre*, *red*) ∈ *set* (*p*#*ps*)
show *?thesis*

using *0 1 C a2 a4 a5 prered-es assms*(*2*,*3*,*7*) *sound-prered-ptr-def length-upd-bin nth-item-upd-bin*
by (*smt* (*verit*) *dual-order.strict-trans1 nth-list-update-eq nth-list-update-neq nth-mem*)
next
assume *a5*: (k', *pre*, *red*) $\notin$ *set* (*p#ps*)
hence *a5*: (k', *pre*, *red*) $\in$ *set* (*p'#ps'*)
using *a3 a4* **by** *auto*
have $k' <$ *idx* $\wedge$ *pre* $<$ *length* (*bs*!k') $\wedge$ *red* $<$ *length* (*bs*!*idx*) $\wedge$
completes idx (*fst* (*bs*!k'!*pre*)) (*fst e*) (*fst* (*bs*!*idx*!*red*))
using *assms*(*1*−*3*) *C D a2 a5* **unfolding** *sound-ptrs-def sound-prered-ptr-def* **by** (*metis nth-mem*)
thus *?thesis*
using *0 1 C a4 assms*(*2*,*3*) *length-upd-bin nth-item-upd-bin prered-es sound-prered-ptr-def*
by (*smt* (*verit*, *best*) *dual-order.strict-trans1 nth-list-update-eq nth-list-update-neq nth-mem*)
qed
qed
ultimately show *?thesis*
by *blast*
next
assume *a3*: *i*≠*j*
hence *elem* $\in$ *set es*
using *0 1* **by** (*metis length-upd-bin nth-mem order-less-le-trans*)
thus *?thesis*
using *assms*(*1*−*3*) *pre-es prered-es sound-ptrs-def* **by** *blast*
qed
qed
qed
next
assume *a2*: *idx* $\neq$ *k*
have *null*: *sound-null-ptr elem*
using *a0 a1 a2 assms*(*1*) *sound-ptrs-def* **by** *auto*
have *sound-pre-ptr* ω *bs idx elem*
using *a0 a1 a2 assms*(*1*,*2*) **unfolding** *sound-ptrs-def* **by** *simp*
hence *pre*: *sound-pre-ptr* ω *?bs idx elem*
using *assms*(*2*,*3*) *length-upd-bin nth-item-upd-bin* **unfolding** *sound-pre-ptr-def*
using *dual-order.strict-trans1 nth-list-update* **by** (*metis* (*no-types*, *lifting*))
have *sound-prered-ptr bs idx elem*
using *a0 a1 a2 assms*(*1*,*2*) **unfolding** *sound-ptrs-def* **by** *simp*
hence *prered*: *sound-prered-ptr ?bs idx elem*
using *assms*(*2*,*3*) *length-upd-bin nth-item-upd-bin* **unfolding** *sound-prered-ptr-def*
by (*smt* (*verit*, *best*) *dual-order.strict-trans1 nth-list-update*)
show *?thesis*
using *null pre prered* **by** *blast*
qed
qed

lemma *mono-red-ptr-upd-bin*:
  assumes *mono-red-ptr bs k* $<$ *length bs es* $=$ *bs*!*k distinct* (*items es*)
  assumes $\forall$ *k′ pre red ps*. *snd e* $=$ *PreRed* (*k′*, *pre*, *red*) *ps* $\longrightarrow$ *red* $<$ *length es*
  shows *mono-red-ptr* (*bs*[*k* := *upd-bin e es*])
  unfolding *mono-red-ptr-def*
proof (*standard*, *standard*)
  fix *idx*
  let *?bs* $=$ *bs*[*k* := *upd-bin e es*]
  assume *a0*: *idx* $<$ *length ?bs*
  show $\forall$ *i* $<$ *length* (*?bs*!*idx*). $\forall$ *k′ pre red ps*. *snd* (*?bs*!*idx*!*i*) $=$ *PreRed* (*k′*, *pre*, *red*) *ps* $\longrightarrow$ *red* $<$ *i*
  proof *cases*
    assume *a1*: *idx*$=$*k*
    consider (*A*) *fst e* $\notin$ *set* (*items es*) |
      (*B*) *fst e* $\in$ *set* (*items es*) $\wedge$ ($\exists$ *pre*. *snd e* $=$ *Null* $\vee$ *snd e* $=$ *Pre pre*) |
      (*C*) *fst e* $\in$ *set* (*items es*) $\wedge$ $\neg$ ($\exists$ *pre*. *snd e* $=$ *Null* $\vee$ *snd e* $=$ *Pre pre*)
      by *blast*
    thus *?thesis*
    proof *cases*
      case *A*
      hence *upd-bin e es* $=$ *es* @ [*e*]
        using *upd-bin-append* **by** *blast*
      thus *?thesis*
        using *a1 assms*(*1*$-$*3*,*5*) *mono-red-ptr-def*
        by (*metis length-append-singleton less-antisym nth-append nth-append-length nth-list-update-eq*)
    next
      case *B*
      hence *upd-bin e es* $=$ *es*
        using *upd-bin-null-pre* **by** *blast*
      thus *?thesis*
        using *a1 assms*(*1*$-$*3*) *mono-red-ptr-def* **by** *force*
    next
      case *C*
      then obtain *i p ps* **where** *C*: *i* $<$ *length es fst e* $=$ *fst* (*es*!*i*) *snd e* $=$ *PreRed p ps*
        by (*metis in-set-conv-nth items-def length-map nth-map pointer*.*exhaust*)
      show *?thesis*
      proof *cases*
        assume $\nexists$ *p′ ps′*. *snd* (*es*!*i*) $=$ *PreRed p′ ps′*
        hence *upd-bin e es* $=$ *es*
          using *upd-bin-prered-nop C assms*(*4*) **by** *blast*
        thus *?thesis*
          using *a1 assms*(*1*$-$*3*) *mono-red-ptr-def* **by** (*metis nth-list-update-eq*)
      next
        assume $\neg$ ($\nexists$ *p′ ps′*. *snd* (*es*!*i*) $=$ *PreRed p′ ps′*)
        then obtain *p′ ps′* **where** *D*: *snd* (*es*!*i*) $=$ *PreRed p′ ps′*
          by *blast*
        have *0*: *snd* (*upd-bin e es*!*i*) $=$ *PreRed p′* (*p*#*ps*@*ps′*) $\wedge$

$(\forall j < \textit{length} (\textit{upd-bin } e \; es).\ i \neq j \longrightarrow \textit{upd-bin } e \; es!j = es!j) \wedge$
length (*upd-bin e es*) = *length es*
using *C D assms*(*4*) *upd-bin-prered-upd* **by** *blast*
show *?thesis*
proof (*standard*, *standard*, *standard*, *standard*, *standard*, *standard*, *standard*)
fix *j k′ pre red PS*
assume *a2*: $j < \textit{length} (\textit{?bs}!\textit{idx})$
assume *a3*: *snd* (*?bs*!*idx*!*j*) = *PreRed* (*k′*, *pre*, *red*) *PS*
have *1*: *?bs*!*idx* = *upd-bin e es*
by (*simp add*: *a1 assms*(*2*))
show $\textit{red} < j$
proof *cases*
assume *a4*: *i=j*
show *?thesis*
using *0 1 C*(*1*) *D a3 a4 assms*(*1−3*) **unfolding** *mono-red-ptr-def* **by** (*metis pointer.inject*(*2*))
next
assume *a4*: $i \neq j$
thus *?thesis*
using *0 1 a2 a3 assms*(*1*) *assms*(*2*) *assms*(*3*) *mono-red-ptr-def* **by** *force*
qed
qed
qed
qed
next
assume *a1*: $\textit{idx} \neq k$
show *?thesis*
using *a0 a1 assms*(*1*) *mono-red-ptr-def* **by** *fastforce*
qed
qed

lemma *sound-mono-ptrs-upds-bin*:
assumes *sound-ptrs* ω *bs mono-red-ptr bs* $k < \textit{length bs}$ $b = bs!k$ *distinct* (*items b*)
assumes $\forall e \in \textit{set es}.\ \textit{sound-null-ptr } e \wedge \textit{sound-pre-ptr } \omega \; bs \; k \; e \wedge \textit{sound-prered-ptr } bs \; k \; e$
assumes $\forall e \in \textit{set es}.\ \forall k' \; \textit{pre red ps}.\ \textit{snd } e = \textit{PreRed } (k', \textit{pre}, \textit{red}) \; ps \longrightarrow \textit{red} < \textit{length } b$
shows *sound-ptrs* ω (*bs*[*k* := *upds-bin es b*]) $\wedge$ *mono-red-ptr* (*bs*[*k* := *upds-bin es b*])
using *assms*
proof (*induction es arbitrary*: *b bs*)
case (*Cons e es*)
let *?bs* = *bs*[*k* := *upd-bin e b*]
have *0*: *sound-ptrs* ω *?bs*
using *sound-ptrs-upd-bin Cons.prems*(*1*,*3−5*,*6*) **by** (*metis list.set-intros*(*1*))
have *1*: *mono-red-ptr ?bs*
using *mono-red-ptr-upd-bin Cons.prems*(*2−5*,*7*) **by** (*metis list.set-intros*(*1*))

have *2*: $k < length\ ?bs$
using *Cons.prems*(*3*) **by** *simp*
have *3*: $upd\text{-}bin\ e\ b = ?bs!k$
using *Cons.prems*(*3*) **by** *simp*
have *4*: $\forall e' \in set\ es.\ sound\text{-}null\text{-}ptr\ e' \wedge sound\text{-}pre\text{-}ptr\ \omega\ ?bs\ k\ e' \wedge sound\text{-}prered\text{-}ptr\ ?bs\ k\ e'$
using *Cons.prems*(*3*,*4*,*6*) *length-upd-bin nth-item-upd-bin sound-pre-ptr-def sound-prered-ptr-def*
by (*smt* (*verit*, *ccfv-threshold*) *list.set-intros*(*2*) *nth-list-update order-less-le-trans*)
have *5*: $\forall e' \in set\ es.\ \forall k'\ pre\ red\ ps.\ snd\ e' = PreRed\ (k', pre, red)\ ps \longrightarrow red < length\ (upd\text{-}bin\ e\ b)$
by (*meson Cons.prems*(*7*) *length-upd-bin order-less-le-trans set-subset-Cons subsetD*)
have $sound\text{-}ptrs\ \omega\ ((bs[k := upd\text{-}bin\ e\ b])[k := upds\text{-}bin\ es\ (upd\text{-}bin\ e\ b)]) \wedge mono\text{-}red\text{-}ptr\ (bs[k := upd\text{-}bin\ e\ b,\ k := upds\text{-}bin\ es\ (upd\text{-}bin\ e\ b)])$
using *Cons.IH*[*OF 0 1 2 3 - -*] *distinct-upd-bin Cons.prems*(*4*,*5*,*6*) *items-def 4 5* **by** *blast*
thus *?case*
by *simp*
qed *simp*

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-red-ptr bs*
assumes *nonempty-derives* $\mathcal{G}$
shows $sound\text{-}ptrs\ \omega\ (Earley_L\text{-}bin'\ k\ \mathcal{G}\ \omega\ bs\ i) \wedge mono\text{-}red\text{-}ptr\ (Earley_L\text{-}bin'\ k\ \mathcal{G}\ \omega\ bs\ i)$
using *assms*
proof (*induction i rule*: *Earley*$_L$*-bin'-induct*[*OF assms*(*1*), *case-names Base Complete*$_F$ *Scan*$_F$ *Pass Predict*$_F$])
case ($Complete_F\ k\ \mathcal{G}\ \omega\ bs\ i\ x$)
let $?bs' = upd\text{-}bins\ bs\ k\ (Complete_L\ k\ x\ bs\ i)$
have *x*: $x \in set\ (items\ (bs\ !\ k))$
using *Complete*$_F$*.hyps*(*1*,*2*) **by** *force*
hence $\forall x \in set\ (items\ (Complete_L\ k\ x\ bs\ i)).\ sound\text{-}item\ \mathcal{G}\ \omega\ x$
using *sound-Complete*$_L$ *Complete*$_F$*.hyps*(*3*) *Complete*$_F$*.prems wf-earley-input-elim wf-bins-impl-wf-items x*
by (*metis dual-order.refl*)
hence *sound*: $\forall x \in bins\ ?bs'.\ sound\text{-}item\ \mathcal{G}\ \omega\ x$
by (*metis Complete*$_F$*.prems*(*1*,*3*) *UnE bins-upd-bins wf-earley-input-elim*)
have *0*: $k < length\ bs$
using *Complete*$_F$*.prems*(*1*) *wf-earley-input-elim* **by** *auto*
have *1*: $\forall e \in set\ (Complete_L\ k\ x\ bs\ i).\ sound\text{-}null\text{-}ptr\ e$
unfolding *Complete*$_L$*-def sound-null-ptr-def* **by** *auto*
have *2*: $\forall e \in set\ (Complete_L\ k\ x\ bs\ i).\ sound\text{-}pre\text{-}ptr\ \omega\ bs\ k\ e$
unfolding *Complete*$_L$*-def sound-pre-ptr-def* **by** *auto*
{
fix *e*

assume *a0*: $e \in$ *set* (*Complete*$_L$ *k x bs i*)
fix *p ps k′ pre red*
assume *a1*: *snd e* = *PreRed p ps* (*k′*, *pre*, *red*) $\in$ *set* (*p#ps*)
have *k′* = *start-item x*
using *a0 a1* **unfolding** *Complete*$_L$*-def* **by** *auto*
moreover have *wf-item* $\mathcal{G}$ ω *x end-item x* = *k*
using *Complete*$_F$*.prems*(*1*) *x wf-earley-input-elim wf-bins-kth-bin* **by** *blast+*
ultimately have *0*: *k′* $\leq$ *k*
using *wf-item-def* **by** *blast*
have *1*: *k′* $\neq$ *k*
proof (*rule ccontr*)
assume $\neg$ *k′* $\neq$ *k*
have *sound-item* $\mathcal{G}$ ω *x*
using *Complete*$_F$*.prems*(*1*,*3*) *x kth-bin-sub-bins wf-earley-input-elim* **by** (*metis subset-eq*)
moreover have *is-complete x*
using *Complete*$_F$*.hyps*(*3*) **by** (*auto simp*: *next-symbol-def split*: *if-splits*)
moreover have *start-item x* = *k*
using ‹$\neg$ *k′* $\neq$ *k*› ‹*k′* = *start-item x*› **by** *auto*
ultimately show *False*
using *impossible-complete-item Complete*$_F$*.prems*(*1*,*5*) *wf-earley-input-elim* ‹*end-item x* = *k*› ‹*wf-item* $\mathcal{G}$ ω *x*› **by** *blast*
qed
have *2*: *pre* < *length* (*bs*!*k′*)
using *a0 a1 index-filter-with-index-lt-length* **unfolding** *Complete*$_L$*-def* **by** (*auto simp*: *items-def*; *fastforce*)
have *3*: *red* < *i*+*1*
using *a0 a1* **unfolding** *Complete*$_L$*-def* **by** *auto*

have *fst e* = *inc-item* (*fst* (*bs*!*k′*!*pre*)) *k*
using *a0 a1 0 2 Complete*$_F$*.hyps*(*1*,*2*,*3*) *Complete*$_F$*.prems*(*1*) ‹*k′* = *start-item x*› **unfolding** *Complete*$_L$*-def*
by (*auto simp*: *items-def*, *metis filter-with-index-nth nth-map*)
moreover have *is-complete* (*fst* (*bs*!*k*!*red*))
using *a0 a1 0 2 Complete*$_F$*.hyps*(*1*,*2*,*3*) *Complete*$_F$*.prems*(*1*) ‹*k′* = *start-item x*› **unfolding** *Complete*$_L$*-def*
by (*auto simp*: *next-symbol-def items-def split*: *if-splits*)
moreover have *start-item* (*fst* (*bs*!*k*!*red*)) = *end-item* (*fst* (*bs*!*k′*!*pre*))
using *a0 a1 0 2 Complete*$_F$*.hyps*(*1*,*2*,*3*) *Complete*$_F$*.prems*(*1*) ‹*k′* = *start-item x*› **unfolding** *Complete*$_L$*-def*
apply (*auto simp*: *items-def*)
by (*metis dual-order.strict-trans index-filter-with-index-lt-length items-def le-neq-implies-less nth-map nth-mem wf-bins-kth-bin wf-earley-input-elim*)
moreover have ($\exists$ *N*. *next-symbol* (*fst* (*bs* ! *k′* ! *pre*)) = *Some N* $\wedge$ *N* = *lhs-item* (*fst* (*bs* ! *k* ! *red*)))
using *a0 a1 0 2 Complete*$_F$*.hyps*(*1*,*2*,*3*) *Complete*$_F$*.prems*(*1*) ‹*k′* = *start-item x*› **unfolding** *Complete*$_L$*-def*
by (*auto simp*: *items-def*, *metis* (*mono-tags*, *lifting*) *filter-with-index-P filter-with-index-nth nth-map*)

ultimately have *4*: *completes* k (*fst* (*bs*!k'!*pre*)) (*fst* e) (*fst* (*bs*!k!*red*))
unfolding *completes-def* **by** *blast*
have $k' < k$ *pre* < *length* (*bs*!k') *red* < *i+1* *completes* k (*fst* (*bs*!k'!*pre*)) (*fst* e) (*fst* (*bs*!k!*red*))
using *0 1 2 3 4* **by** *simp-all*
}
hence $\forall e \in$ *set* (*Complete*$_L$ k x *bs* i). $\forall$ p *ps* k' *pre red*. *snd* e = *PreRed* p *ps* $\wedge$ (k', *pre*, *red*) $\in$ *set* (p#*ps*) $\longrightarrow$
$k' < k$ $\wedge$ *pre* < *length* (*bs*!k') $\wedge$ *red* < *i+1* $\wedge$ *completes* k (*fst* (*bs*!k'!*pre*)) (*fst* e) (*fst* (*bs*!k!*red*))
by *force*
hence *3*: $\forall e \in$ *set* (*Complete*$_L$ k x *bs* i). *sound-prered-ptr bs k e*
unfolding *sound-prered-ptr-def* **using** *Complete*$_F$.*hyps*(*1*) *items-def*
by (*smt* (*verit*, *del-insts*) *le-antisym le-eq-less-or-eq le-trans length-map length-pos-if-in-set less-imp-add-positive less-one nat-add-left-cancel-le nat-neq-iff plus-nat.add-0*)
have *sound-ptrs* ω *?bs'* $\wedge$ *mono-red-ptr ?bs'*
using *sound-mono-ptrs-upds-bin*[*OF Complete*$_F$.*prems*(*2*) *Complete*$_F$.*prems*(*4*) *0*] *1 2 3 sound-prered-ptr-def*
Complete$_F$.*prems*(*1*) *upd-bins-def wf-earley-input-elim wf-bin-def wf-bins-def*
by (*smt* (*verit*, *ccfv-SIG*) *list.set-intros*(*1*))
moreover have (k, $\mathcal{G}$, ω, *?bs'*) $\in$ *wf-earley-input*
using *Complete*$_F$.*hyps Complete*$_F$.*prems*(*1*) *wf-earley-input-Complete*$_L$ **by** *blast*
ultimately have *sound-ptrs* ω (*Earley*$_L$-*bin'* k $\mathcal{G}$ ω *?bs'* (*i+1*)) $\wedge$ *mono-red-ptr* (*Earley*$_L$-*bin'* k $\mathcal{G}$ ω *?bs'* (*i+1*))
using *Complete*$_F$.*IH Complete*$_F$.*prems*(*4*−*5*) *sound* **by** *blast*
thus *?case*
using *Complete*$_F$.*hyps* **by** *simp*
next
case (*Scan*$_F$ k $\mathcal{G}$ ω *bs* i x a)
let *?bs'* = *upd-bins bs* (*k+1*) (*Scan*$_L$ k ω a x i)
have $x \in$ *set* (*items* (*bs* ! k))
using *Scan*$_F$.*hyps*(*1*,*2*) **by** *force*
hence $\forall x \in$ *set* (*items* (*Scan*$_L$ k ω a x i)). *sound-item* $\mathcal{G}$ ω x
using *sound-Scan*$_L$ *Scan*$_F$.*hyps*(*3*,*5*) *Scan*$_F$.*prems*(*1*,*2*,*3*) *wf-earley-input-elim wf-bins-impl-wf-items wf-bins-impl-wf-items* **by** *fast*
hence *sound*: $\forall x \in$ *bins ?bs'*. *sound-item* $\mathcal{G}$ ω x
using *Scan*$_F$.*hyps*(*5*) *Scan*$_F$.*prems*(*1*,*3*) *bins-upd-bins wf-earley-input-elim*
by (*metis UnE add-less-cancel-right*)
have *0*: *k+1* < *length bs*
using *Scan*$_F$.*hyps*(*5*) *Scan*$_F$.*prems*(*1*) *wf-earley-input-elim* **by** *force*
have *1*: $\forall e \in$ *set* (*Scan*$_L$ k ω a x i). *sound-null-ptr* e
unfolding *Scan*$_L$-*def sound-null-ptr-def* **by** *auto*
have *2*: $\forall e \in$ *set* (*Scan*$_L$ k ω a x i). *sound-pre-ptr* ω *bs* (*k+1*) e
using *Scan*$_F$.*hyps*(*1*,*2*,*3*) **unfolding** *sound-pre-ptr-def Scan*$_L$-*def scans-def items-def* **by** *auto*
have *3*: $\forall e \in$ *set* (*Scan*$_L$ k ω a x i). *sound-prered-ptr bs* (*k+1*) e
unfolding *Scan*$_L$-*def sound-prered-ptr-def* **by** *simp*
have *sound-ptrs* ω *?bs'* $\wedge$ *mono-red-ptr ?bs'*
using *sound-mono-ptrs-upds-bin*[*OF Scan*$_F$.*prems*(*2*) *Scan*$_F$.*prems*(*4*) *0*] *0 1 2*

3 sound-prered-ptr-def
Scan$_F$.prems(1) upd-bins-def wf-earley-input-elim wf-bin-def wf-bins-def
by (*smt* (*verit*, *ccfv-threshold*) *list.set-intros*(*1*))
moreover have (k, $\mathcal{G}$, ω, *?bs'*) $\in$ *wf-earley-input*
using *Scan$_F$.hyps Scan$_F$.prems(1) wf-earley-input-Scan$_L$* **by** *metis*
ultimately have *sound-ptrs* ω (*Earley$_L$-bin' k* $\mathcal{G}$ ω *?bs'* (*i+1*)) $\wedge$ *mono-red-ptr* (*Earley$_L$-bin' k* $\mathcal{G}$ ω *?bs'* (*i+1*))
using *Scan$_F$.IH Scan$_F$.prems(4−5) sound* **by** *blast*
thus *?case*
using *Scan$_F$.hyps* **by** *simp*
next
case (*Predict$_F$ k* $\mathcal{G}$ ω *bs i x a*)
let *?bs'* = *upd-bins bs k* (*Predict$_L$ k* $\mathcal{G}$ *a*)
have $x \in$ *set* (*items* (*bs* ! *k*))
using *Predict$_F$.hyps(1,2)* **by** *force*
hence $\forall x \in$ *set* (*items*(*Predict$_L$ k* $\mathcal{G}$ *a*)). *sound-item* $\mathcal{G}$ ω *x*
using *sound-Predict$_L$ Predict$_F$.hyps(3) Predict$_F$.prems wf-earley-input-elim wf-bins-impl-wf-items* **by** *fast*
hence *sound*: $\forall x \in$ *bins ?bs'*. *sound-item* $\mathcal{G}$ ω *x*
using *Predict$_F$.prems(1,3) UnE bins-upd-bins wf-earley-input-elim* **by** *metis*
have *0*: $k <$ *length bs*
using *Predict$_F$.prems(1) wf-earley-input-elim* **by** *force*
have *1*: $\forall e \in$ *set* (*Predict$_L$ k* $\mathcal{G}$ *a*). *sound-null-ptr e*
unfolding *sound-null-ptr-def Predict$_L$-def predicts-def* **by** (*auto simp*: *init-item-def*)
have *2*: $\forall e \in$ *set* (*Predict$_L$ k* $\mathcal{G}$ *a*). *sound-pre-ptr* ω *bs k e*
unfolding *sound-pre-ptr-def Predict$_L$-def* **by** *simp*
have *3*: $\forall e \in$ *set* (*Predict$_L$ k* $\mathcal{G}$ *a*). *sound-prered-ptr bs k e*
unfolding *sound-prered-ptr-def Predict$_L$-def* **by** *simp*
have *sound-ptrs* ω *?bs'* $\wedge$ *mono-red-ptr ?bs'*
using *sound-mono-ptrs-upds-bin*[*OF Predict$_F$.prems(2) Predict$_F$.prems(4) 0*] *0 1 2 3 sound-prered-ptr-def*
Predict$_F$.prems(1) upd-bins-def wf-earley-input-elim wf-bin-def wf-bins-def
by (*smt* (*verit*, *ccfv-threshold*) *list.set-intros*(*1*))
moreover have (k, $\mathcal{G}$, ω, *?bs'*) $\in$ *wf-earley-input*
using *Predict$_F$.hyps Predict$_F$.prems(1) wf-earley-input-Predict$_L$* **by** *metis*
ultimately have *sound-ptrs* ω (*Earley$_L$-bin' k* $\mathcal{G}$ ω *?bs'* (*i+1*)) $\wedge$ *mono-red-ptr* (*Earley$_L$-bin' k* $\mathcal{G}$ ω *?bs'* (*i+1*))
using *Predict$_F$.IH Predict$_F$.prems(4−5) sound* **by** *blast*
thus *?case*
using *Predict$_F$.hyps* **by** *simp*
qed *simp-all*

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

using *assms sound-mono-ptrs-Earley*$_L$*-bin*$'$ *Earley*$_L$*-bin-def* **by** *metis*

lemma *sound-ptrs-Init*$_L$:
sound-ptrs ω (*Init*$_L$ $\mathcal{G}$ ω)
unfolding *sound-ptrs-def sound-null-ptr-def sound-pre-ptr-def sound-prered-ptr-def predicts-def scans-def completes-def Init*$_L$*-def*
by (*auto simp*: *init-item-def less-Suc-eq-0-disj*)

lemma *mono-red-ptr-Init*$_L$:
mono-red-ptr (*Init*$_L$ $\mathcal{G}$ ω)
unfolding *mono-red-ptr-def Init*$_L$*-def*
by (*auto simp*: *init-item-def less-Suc-eq-0-disj*)

lemma *sound-mono-ptrs-Earley*$_L$*-bins*:
assumes $k \leq$ *length* ω *nonempty-derives* $\mathcal{G}$
shows *sound-ptrs* ω (*Earley*$_L$*-bins* k $\mathcal{G}$ ω) $\wedge$ *mono-red-ptr* (*Earley*$_L$*-bins* k $\mathcal{G}$ ω)
using *assms*
proof (*induction* k)
case *0*
have (*0*, $\mathcal{G}$, ω, (*Init*$_L$ $\mathcal{G}$ ω)) $\in$ *wf-earley-input*
using *assms*(*2*) *wf-earley-input-Init*$_L$ **by** *blast*
moreover have $\forall x \in$ *bins* (*Init*$_L$ $\mathcal{G}$ ω). *sound-item* $\mathcal{G}$ ω x
by (*metis Init*$_L$*-eq-Init*$_F$ *Init*$_F$*-sub-Earley sound-Earley subsetD wf-Earley*)
ultimately show *?case*
using *sound-mono-ptrs-Earley*$_L$*-bin sound-ptrs-Init*$_L$ *mono-red-ptr-Init*$_L$ *0.prems*
by *fastforce*
next
case (*Suc k*)
have (*Suc k*, $\mathcal{G}$, ω, *Earley*$_L$*-bins* k $\mathcal{G}$ ω) $\in$ *wf-earley-input*
by (*simp add*: *Suc.prems*(*1*) *Suc-leD assms*(*2*) *wf-earley-input-intro*)
moreover have *sound-ptrs* ω (*Earley*$_L$*-bins* k $\mathcal{G}$ ω)
using *Suc* **by** *simp*
moreover have $\forall x \in$ *bins* (*Earley*$_L$*-bins* k $\mathcal{G}$ ω). *sound-item* $\mathcal{G}$ ω x
by (*meson Suc.prems*(*1*) *Suc-leD Earley*$_L$*-bins-sub-Earley*$_F$*-bins Earley*$_F$*-bins-sub-Earley assms*(*2*)
sound-Earley subsetD wf-bins-Earley$_L$*-bins wf-bins-impl-wf-items*)
ultimately show *?case*
using *Suc.prems sound-mono-ptrs-Earley*$_L$*-bin Suc.IH* **by** *fastforce*
qed

lemma *sound-mono-ptrs-Earley*$_L$:
assumes *nonempty-derives* $\mathcal{G}$
shows *sound-ptrs* ω (*Earley*$_L$ $\mathcal{G}$ ω) $\wedge$ *mono-red-ptr* (*Earley*$_L$ $\mathcal{G}$ ω)
using *assms sound-mono-ptrs-Earley*$_L$*-bins Earley*$_L$*-def* **by** (*metis dual-order.refl*)

9.2 Common Definitions

datatype *'a tree* =
Leaf 'a

| *Branch* *'a* *'a tree list*

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

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

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

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

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

9.3 foldl lemmas

lemma *foldl-add-nth*:
k < *length xs* ⟹ *foldl* (+) *z* (*map length* (*take k xs*)) + *length* (*xs*!*k*) = *foldl* (+) *z* (*map length* (*take* (*k*+*1*) *xs*))
proof (*induction xs arbitrary*: *k z*)
case (*Cons x xs*)
then show *?case*
proof (*cases k* = *0*)
case *False*
thus *?thesis*
using *Cons* **by** (*auto simp add*: *take-Cons'*)
qed *simp*
qed *simp*

lemma *foldl-acc-mono*:
a ≤ *b* ⟹ *foldl* (+) *a xs* ≤ *foldl* (+) *b xs* **for** *a* :: *nat*
by (*induction xs arbitrary*: *a b*) *auto*

lemma *foldl-ge-z-nth*:
j < *length xs* ⟹ *z* + *length* (*xs*!*j*) ≤ *foldl* (+) *z* (*map length* (*take* (*j*+*1*) *xs*))
proof (*induction xs arbitrary*: *j z*)
case (*Cons x xs*)
show *?case*

proof (*cases* $j = 0$)
case *False*
have $z + length\ ((x \# xs) ! j) = z + length\ (xs!(j-1))$
using *False* **by** *simp*
also have ... $\leq foldl\ (+)\ z\ (map\ length\ (take\ (j-1+1)\ xs))$
using *Cons False* **by** (*metis add-diff-inverse-nat length-Cons less-one nat-add-left-cancel-less plus-1-eq-Suc*)
also have ... $= foldl\ (+)\ z\ (map\ length\ (take\ j\ xs))$
using *False* **by** *simp*
also have ... $\leq foldl\ (+)\ (z + length\ x)\ (map\ length\ (take\ j\ xs))$
using *foldl-acc-mono* **by** *force*
also have ... $= foldl\ (+)\ z\ (map\ length\ (take\ (j+1)\ (x\#xs)))$
by *simp*
finally show *?thesis*
by *blast*
qed *simp*
qed *simp*

lemma *foldl-add-nth-ge*:
$i \leq j \Longrightarrow j < length\ xs \Longrightarrow foldl\ (+)\ z\ (map\ length\ (take\ i\ xs)) + length\ (xs!j) \leq foldl\ (+)\ z\ (map\ length\ (take\ (j+1)\ xs))$
proof (*induction xs arbitrary*: *i j z*)
case (*Cons x xs*)
show *?case*
proof (*cases* $i = 0$)
case *True*
have $foldl\ (+)\ z\ (map\ length\ (take\ i\ (x \# xs))) + length\ ((x \# xs) ! j) = z + length\ ((x \# xs) ! j)$
using *True* **by** *simp*
also have ... $\leq foldl\ (+)\ z\ (map\ length\ (take\ (j+1)\ (x\#xs)))$
using *foldl-ge-z-nth Cons.prems*(*2*) **by** *blast*
finally show *?thesis*
by *blast*
next
case *False*
have $i-1 \leq j-1$
by (*simp add*: *Cons.prems*(*1*) *diff-le-mono*)
have $j-1 < length\ xs$
using *Cons.prems*(*1*,*2*) *False* **by** *fastforce*
have $foldl\ (+)\ z\ (map\ length\ (take\ i\ (x \# xs))) + length\ ((x \# xs) ! j) =$
$foldl\ (+)\ (z + length\ x)\ (map\ length\ (take\ (i-1)\ xs)) + length\ ((x\#xs)!j)$
using *False* **by** (*simp add*: *take-Cons'*)
also have ... $= foldl\ (+)\ (z + length\ x)\ (map\ length\ (take\ (i-1)\ xs)) + length\ (xs!(j-1))$
using *Cons.prems*(*1*) *False* **by** *auto*
also have ... $\leq foldl\ (+)\ (z + length\ x)\ (map\ length\ (take\ (j-1+1)\ xs))$
using *Cons.IH* ‹$i - 1 \leq j - 1$› ‹$j - 1 < length\ xs$› **by** *blast*
also have ... $= foldl\ (+)\ (z + length\ x)\ (map\ length\ (take\ j\ xs))$
using *Cons.prems*(*1*) *False* **by** *fastforce*

```
    also have ... = foldl (+) z (map length (take (j+1) (x#xs)))
      by fastforce
    finally show ?thesis
      by blast
  qed
qed simp

lemma foldl-ge-acc:
  foldl (+) z (map length xs) ≥ z
  by (induction xs arbitrary: z) (auto elim: add-leE)

lemma foldl-take-mono:
  i ≤ j ⟹ foldl (+) z (map length (take i xs)) ≤ foldl (+) z (map length (take j
xs))
proof (induction xs arbitrary: z i j)
  case (Cons x xs)
  show ?case
  proof (cases i = 0)
    case True
    have foldl (+) z (map length (take i (x # xs))) = z
      using True by simp
    also have ... ≤ foldl (+) z (map length (take j (x # xs)))
      by (simp add: foldl-ge-acc)
    ultimately show ?thesis
      by simp
  next
    case False
    then show ?thesis
      using Cons by (simp add: take-Cons')
  qed
qed simp
```

9.4 Parse tree

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

$t \leftarrow$ *build-tree' bs* ω *k' pre*;
case t of
Branch N ts $\Rightarrow$
do {
$t \leftarrow$ *build-tree' bs* ω *k red*;
Some (*Branch N* (*ts* @ [*t*]))
}
| - $\Rightarrow$ *undefined* — impossible case
})
))

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

definition *build-tree* :: *'a cfg* $\Rightarrow$ *'a list* $\Rightarrow$ *'a bins* $\Rightarrow$ *'a tree option* **where**
build-tree $\mathcal{G}$ ω *bs* = (
let k = *length bs* − *1 in* (
case filter-with-index (λx. *is-finished* $\mathcal{G}$ ω *x*) (*items* (*bs*!*k*)) *of*
[] $\Rightarrow$ *None*
| (-, *i*)#- $\Rightarrow$ *build-tree' bs* ω *k i*
))

lemma *build-tree'-simps*[*simp*]:
e = *bs*!*k*!*i* $\Longrightarrow$ *snd e* = *Null* $\Longrightarrow$ *build-tree' bs* ω *k i* = *Some* (*Branch* (*lhs-item* (*fst e*)) [])
e = *bs*!*k*!*i* $\Longrightarrow$ *snd e* = *Pre pre* $\Longrightarrow$ *build-tree' bs* ω (*k*−*1*) *pre* = *None* $\Longrightarrow$
build-tree' bs ω *k i* = *None*
e = *bs*!*k*!*i* $\Longrightarrow$ *snd e* = *Pre pre* $\Longrightarrow$ *build-tree' bs* ω (*k*−*1*) *pre* = *Some* (*Branch N ts*) $\Longrightarrow$
build-tree' bs ω *k i* = *Some* (*Branch N* (*ts* @ [*Leaf* (ω!(*k*−*1*))]))
e = *bs*!*k*!*i* $\Longrightarrow$ *snd e* = *Pre pre* $\Longrightarrow$ *build-tree' bs* ω (*k*−*1*) *pre* = *Some* (*Leaf a*) $\Longrightarrow$
build-tree' bs ω *k i* = *undefined*
e = *bs*!*k*!*i* $\Longrightarrow$ *snd e* = *PreRed* (*k'*, *pre*, *red*) *reds* $\Longrightarrow$ *build-tree' bs* ω *k' pre* = *None* $\Longrightarrow$
build-tree' bs ω *k i* = *None*
e = *bs*!*k*!*i* $\Longrightarrow$ *snd e* = *PreRed* (*k'*, *pre*, *red*) *reds* $\Longrightarrow$ *build-tree' bs* ω *k' pre* = *Some* (*Branch N ts*) $\Longrightarrow$
build-tree' bs ω *k red* = *None* $\Longrightarrow$ *build-tree' bs* ω *k i* = *None*
e = *bs*!*k*!*i* $\Longrightarrow$ *snd e* = *PreRed* (*k'*, *pre*, *red*) *reds* $\Longrightarrow$ *build-tree' bs* ω *k' pre* = *Some* (*Leaf a*) $\Longrightarrow$
build-tree' bs ω *k i* = *undefined*
e = *bs*!*k*!*i* $\Longrightarrow$ *snd e* = *PreRed* (*k'*, *pre*, *red*) *reds* $\Longrightarrow$ *build-tree' bs* ω *k' pre* = *Some* (*Branch N ts*) $\Longrightarrow$
build-tree' bs ω *k red* = *Some t* $\Longrightarrow$
build-tree' bs ω *k i* = *Some* (*Branch N* (*ts* @ [*t*]))
by (*subst build-tree'.simps*, *simp*)+

definition *wf-tree-input* :: (*'a bins* $\times$ *'a list* $\times$ *nat* $\times$ *nat*) *set* **where**
wf-tree-input = {

$(bs, \omega, k, i) \mid bs\ \omega\ k\ i.$
  sound-ptrs ω *bs* $\wedge$
  mono-red-ptr bs $\wedge$
  $k <$ *length bs* $\wedge$
  $k \leq$ *length* ω $\wedge$
  $i <$ *length* $(bs!k)$
}

fun *build-tree′-measure* :: ('a bins × 'a list × nat × nat) ⇒ nat **where**
  build-tree′-measure (bs, ω, k, i) = *foldl* (+) *0* (*map length* (*take k bs*)) + *i*

lemma *wf-tree-input-pre*:
  assumes $(bs, \omega, k, i) \in$ *wf-tree-input*
  assumes $e = bs!k!i$ *snd e = Pre pre*
  shows $(bs, \omega, (k-1), pre) \in$ *wf-tree-input*
  using *assms* **unfolding** *wf-tree-input-def*
  apply (*auto simp*: *sound-ptrs-def sound-pre-ptr-def*)
  apply (*metis nth-mem*)
  done

lemma *wf-tree-input-prered-pre*:
  assumes $(bs, \omega, k, i) \in$ *wf-tree-input*
  assumes $e = bs!k!i$ *snd e = PreRed* (k', pre, red) *ps*
  shows $(bs, \omega, k', pre) \in$ *wf-tree-input*
  using *assms* **unfolding** *wf-tree-input-def*
  apply (*auto simp*: *sound-ptrs-def sound-prered-ptr-def*)
  apply (*metis fst-conv snd-conv*)+
  apply (*metis dual-order.strict-trans nth-mem*)
  apply *fastforce*
  by (*metis nth-mem*)

lemma *wf-tree-input-prered-red*:
  assumes $(bs, \omega, k, i) \in$ *wf-tree-input*
  assumes $e = bs!k!i$ *snd e = PreRed* (k', pre, red) *ps*
  shows $(bs, \omega, k, red) \in$ *wf-tree-input*
  using *assms* **unfolding** *wf-tree-input-def*
  apply (*auto simp add*: *sound-ptrs-def sound-prered-ptr-def*)
  apply (*metis fst-conv snd-conv nth-mem*)+
  done

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

shows *P bs* ω *k i*
using *assms*(*1*)
proof (*induction n*$\equiv$*build-tree*$'$*-measure* (*bs*, ω, *k*, *i*) *arbitrary*: *k i rule*: *nat-less-induct*)
case *1*
obtain *e* **where** *entry*: *e* = *bs*!*k*!*i*
by *simp*
consider (*Null*) *snd e* = *Null*
| (*Pre*) $\exists$ *pre*. *snd e* = *Pre pre*
| (*PreRed*) $\exists$ *k*$'$ *pre red reds*. *snd e* = *PreRed* (*k*$'$, *pre*, *red*) *reds*
by (*metis pointer*.*exhaust surj-pair*)
thus *?case*
proof *cases*
case *Null*
thus *?thesis*
using *assms*(*2*) *entry* **by** *fastforce*
next
case *Pre*
then obtain *pre* **where** *pre*: *snd e* = *Pre pre*
by *blast*
define *n* **where** *n*: *n* = *build-tree*$'$*-measure* (*bs*, ω, (*k*−*1*), *pre*)
have *0* < *k pre* < *length* (*bs*!(*k*−*1*))
using *1*(*2*) *entry pre* **unfolding** *wf-tree-input-def sound-ptrs-def sound-pre-ptr-def*
by (*smt* (*verit*) *mem-Collect-eq nth-mem prod*.*inject*)+
have *k* < *length bs*
using *1*(*2*) **unfolding** *wf-tree-input-def* **by** *blast*+
have *foldl* (+) *0* (*map length* (*take k bs*)) + *i* − (*foldl* (+) *0* (*map length* (*take* (*k*−*1*) *bs*)) + *pre*) =
foldl (+) *0* (*map length* (*take* (*k*−*1*) *bs*)) + *length* (*bs*!(*k*−*1*)) + *i* − (*foldl* (+) *0* (*map length* (*take* (*k*−*1*) *bs*)) + *pre*)
using *foldl-add-nth*[*of* ‹*k*−*1*› *bs 0*] **by** (*simp add*: ‹*0* < *k*› ‹*k* < *length bs*› *less-imp-diff-less*)
also have ... = *length* (*bs*!(*k*−*1*)) + *i* − *pre*
by *simp*
also have ... > *0*
using ‹*pre* < *length* (*bs*!(*k*−*1*))› **by** *auto*
finally have *build-tree*$'$*-measure* (*bs*, ω, *k*, *i*) − *build-tree*$'$*-measure* (*bs*, ω, (*k*−*1*), *pre*) > *0*
by *simp*
hence *P bs* ω (*k*−*1*) *pre*
using *1 n wf-tree-input-pre entry pre zero-less-diff* **by** *blast*
thus *?thesis*
using *assms*(*2*) *entry pre pointer*.*distinct*(*5*) *pointer*.*inject*(*1*) **by** *presburger*
next
case *PreRed*
then obtain *k*$'$ *pre red ps* **where** *prered*: *snd e* = *PreRed* (*k*$'$, *pre*, *red*) *ps*
by *blast*
have *k*$'$ < *k pre* < *length* (*bs*!*k*$'$)
using *1*(*2*) *entry prered* **unfolding** *wf-tree-input-def sound-ptrs-def sound-prered-ptr-def*
apply *simp-all*

apply (*metis nth-mem*)+
done
have *red* < *i*
using *1*(*2*) *entry prered* **unfolding** *wf-tree-input-def mono-red-ptr-def* **by** *blast*
have *k* < *length bs i* < *length* (*bs*!*k*)
using *1*(*2*) **unfolding** *wf-tree-input-def* **by** *blast*+
define *n-pre* **where** *n-pre*: *n-pre* = *build-tree′-measure* (*bs*, ω, *k′*, *pre*)
have *0* < *length* (*bs*!*k′*) + *i* − *pre*
by (*simp add*: ‹*pre* < *length* (*bs*!*k′*)› *add.commute trans-less-add2*)
also have ... = *foldl* (+) *0* (*map length* (*take k′ bs*)) + *length* (*bs*!*k′*) + *i* − (*foldl* (+) *0* (*map length* (*take k′ bs*)) + *pre*)
by *simp*
also have ... ≤ *foldl* (+) *0* (*map length* (*take* (*k′*+*1*) *bs*)) + *i* − (*foldl* (+) *0* (*map length* (*take k′ bs*)) + *pre*)
using *foldl-add-nth-ge*[*of k′ k′ bs 0*] ‹*k* < *length bs*› ‹*k′* < *k*› **by** *simp*
also have ... ≤ *foldl* (+) *0* (*map length* (*take k bs*)) + *i* − (*foldl* (+) *0* (*map length* (*take k′ bs*)) + *pre*)
using *foldl-take-mono* **by** (*metis Suc-eq-plus1 Suc-leI* ‹*k′* < *k*› *add.commute add-le-cancel-left diff-le-mono*)
finally have *build-tree′-measure* (*bs*, ω, *k*, *i*) − *build-tree′-measure* (*bs*, ω, *k′*, *pre*) > *0*
by *simp*
hence *x*: *P bs* ω *k′ pre*
using *1*(*1*) *zero-less-diff* **by** (*metis 1.prems entry prered wf-tree-input-prered-pre*)
define *n-red* **where** *n-red*: *n-red* = *build-tree′-measure* (*bs*, ω, *k*, *red*)
have *build-tree′-measure* (*bs*, ω, *k*, *i*) − *build-tree′-measure* (*bs*, ω, *k*, *red*) > *0*
using ‹*red* < *i*› **by** *simp*
hence *y*: *P bs* ω *k red*
using *1.hyps 1.prems entry prered wf-tree-input-prered-red zero-less-diff* **by** *blast*
show *?thesis*
using *assms*(*2*) *x y entry prered*
by (*smt* (*verit*, *best*) *Pair-inject filter-cong pointer.distinct*(*5*) *pointer.inject*(*2*))
qed
qed

lemma *build-tree′-termination*:
assumes (*bs*, ω, *k*, *i*) ∈ *wf-tree-input*
shows ∃ *N ts*. *build-tree′ bs* ω *k i* = *Some* (*Branch N ts*)
proof −
have ∃ *N ts*. *build-tree′ bs* ω *k i* = *Some* (*Branch N ts*)
apply (*induction rule*: *build-tree′-induct*[*OF assms*(*1*)])
subgoal premises *IH* **for** *bs* ω *k i*
proof −
define *e* **where** *entry*: *e* = *bs*!*k*!*i*
consider (*Null*) *snd e* = *Null*
| (*Pre*) ∃ *pre*. *snd e* = *Pre pre*
| (*PreRed*) ∃ *k′ pre red ps*. *snd e* = *PreRed* (*k′*, *pre*, *red*) *ps*

by (*metis pointer.exhaust surj-pair*)
thus *?thesis*
proof *cases*
case *Null*
thus *?thesis*
using *build-tree′-simps*(*1*) *entry* **by** *simp*
next
case *Pre*
then obtain *pre* **where** *pre*: *snd e* = *Pre pre*
by *blast*
obtain *N ts* **where** *Nts*: *build-tree′ bs ω* (*k*−*1*) *pre* = *Some* (*Branch N ts*)
using *IH*(*1*) *entry pre* **by** *blast*
have *build-tree′ bs ω k i* = *Some* (*Branch N* (*ts* @ [*Leaf* (*ω*!(*k*−*1*))]))
using *build-tree′-simps*(*3*) *entry pre Nts* **by** *simp*
thus *?thesis*
by *simp*
next
case *PreRed*
then obtain *k′ pre red ps* **where** *prered*: *snd e* = *PreRed* (*k′*, *pre*, *red*) *ps*
by *blast*
then obtain *N ts* **where** *Nts*: *build-tree′ bs ω k′ pre* = *Some* (*Branch N ts*)
using *IH*(*2*) *entry prered* **by** *blast*
obtain *t-red* **where** *t-red*: *build-tree′ bs ω k red* = *Some t-red*
using *IH*(*3*) *entry prered Nts* **by** (*metis option.exhaust*)
have *build-tree′ bs ω k i* = *Some* (*Branch N* (*ts* @ [*t-red*]))
using *build-tree′-simps*(*8*) *entry prered Nts t-red* **by** *auto*
thus *?thesis*
by *blast*
qed
qed
done
thus *?thesis*
by *blast*
qed

lemma *wf-item-tree-build-tree′*:
assumes (*bs*, *ω*, *k*, *i*) ∈ *wf-tree-input*
assumes *wf-bins* $\mathcal{G}$ *ω bs*
assumes *build-tree′ bs ω k i* = *Some t*
shows *wf-item-tree* $\mathcal{G}$ (*fst* (*bs*!*k*!*i*)) *t*
proof −
have *wf-item-tree* $\mathcal{G}$ (*fst* (*bs*!*k*!*i*)) *t*
using *assms*
apply (*induction arbitrary*: *t rule*: *build-tree′-induct*[*OF assms*(*1*)])
subgoal premises *prems* **for** *bs ω k i t*
proof −
define *e* **where** *entry*: *e* = *bs*!*k*!*i*
have *bounds*: *k* < *length bs k* ≤ *length ω i* < *length* (*bs*!*k*)
using *prems*(*4*) *wf-tree-input-def* **by** *force*+

consider (*Null*) *snd e* = *Null*
| (*Pre*) ∃ *pre*. *snd e* = *Pre pre*
| (*PreRed*) ∃ *k′ pre red ps*. *snd e* = *PreRed* (*k′*, *pre*, *red*) *ps*
by (*metis pointer.exhaust surj-pair*)
thus *?thesis*
proof *cases*
case *Null*
hence *build-tree′ bs ω k i* = *Some* (*Branch* (*lhs-item* (*fst e*)) [])
using *entry* **by** *simp*
have *simp*: *t* = *Branch* (*lhs-item* (*fst e*)) []
using *build-tree′-simps*(*1*) *Null prems*(*6*) *entry* **by** *simp*
have *sound-ptrs ω bs*
using *prems*(*4*) **unfolding** *wf-tree-input-def* **by** *blast*
hence *predicts* (*fst e*)
using *Null nth-mem entry bounds* **unfolding** *sound-ptrs-def sound-null-ptr-def* **by** *blast*
hence *dot-item* (*fst e*) = *0*
unfolding *predicts-def* **by** *blast*
thus *?thesis*
using *simp entry* **by** *simp*
next
case *Pre*
then obtain *pre* **where** *pre*: *snd e* = *Pre pre*
by *blast*
obtain *N ts* **where** *Nts*: *build-tree′ bs ω* (*k*−*1*) *pre* = *Some* (*Branch N ts*)
using *build-tree′-termination entry pre prems*(*4*) *wf-tree-input-pre* **by** *blast*
have *simp*: *build-tree′ bs ω k i* = *Some* (*Branch N* (*ts* @ [*Leaf* (*ω*!(*k*−*1*))]))
using *build-tree′-simps*(*3*) *entry pre Nts* **by** *simp*
have *sound-ptrs ω bs*
using *prems*(*4*) **unfolding** *wf-tree-input-def* **by** *blast*
hence *pre* < *length* (*bs*!(*k*−*1*))
using *entry pre bounds* **unfolding** *sound-ptrs-def sound-pre-ptr-def* **by** (*metis nth-mem*)
moreover have *k*−*1* < *length bs*
by (*simp add*: *bounds less-imp-diff-less*)
ultimately have *IH*: *wf-item-tree 𝒢* (*fst* (*bs*!(*k*−*1*)!*pre*)) (*Branch N ts*)
using *prems*(*1*,*2*,*4*,*5*) *entry pre Nts* **by** (*meson wf-tree-input-pre*)
have *scans*: *scans ω k* (*fst* (*bs*!(*k*−*1*)!*pre*)) (*fst e*)
using *entry pre bounds* ‹*sound-ptrs ω bs*› **unfolding** *sound-ptrs-def sound-pre-ptr-def* **by** *simp*
hence ∗:
lhs-item (*fst* (*bs*!(*k*−*1*)!*pre*)) = *lhs-item* (*fst e*)
rhs-item (*fst* (*bs*!(*k*−*1*)!*pre*)) = *rhs-item* (*fst e*)
dot-item (*fst* (*bs*!(*k*−*1*)!*pre*)) + *1* = *dot-item* (*fst e*)
next-symbol (*fst* (*bs*!(*k*−*1*)!*pre*)) = *Some* (*ω*!(*k*−*1*))
unfolding *scans-def inc-item-def* **by** (*simp-all add*: *lhs-item-def rhs-item-def*)
have *map root* (*ts* @ [*Leaf* (*ω*!(*k*−*1*))]) = *map root ts* @ [*ω*!(*k*−*1*)]
by *simp*
also have ... = *take* (*dot-item* (*fst* (*bs*!(*k*−*1*)!*pre*))) (*rhs-item* (*fst* (*bs*!(*k*−*1*)!*pre*)))

@ $[\omega!(k-1)]$
using *IH* **by** *simp*
also have ... = *take* (*dot-item* (*fst* (*bs*!(*k*−*1*)!*pre*))) (*rhs-item* (*fst* *e*)) @ $[\omega!(k-1)]$
using ∗(*2*) **by** *simp*
also have ... = *take* (*dot-item* (*fst* *e*)) (*rhs-item* (*fst* *e*))
using ∗(*2*−*4*) **by** (*auto simp*: *next-symbol-def is-complete-def split*: *if-splits*; *metis leI take-Suc-conv-app-nth*)
finally have *map root* (*ts* @ [*Leaf* (ω!(*k*−*1*))]) = *take* (*dot-item* (*fst* *e*)) (*rhs-item* (*fst* *e*)) **.**
hence *wf-item-tree* $\mathcal{G}$ (*fst* *e*) (*Branch* *N* (*ts* @ [*Leaf* (ω!(*k*−*1*))]))
using *IH* ∗(*1*) **by** *simp*
thus *?thesis*
using *entry prems*(*6*) *simp* **by** *auto*
next
case *PreRed*
then obtain *k′ pre red ps* **where** *pred*: *snd e* = *PreRed* (*k′*, *pre*, *red*) *ps*
by *blast*
obtain *N ts* **where** *Nts*: *build-tree′ bs* ω *k′ pre* = *Some* (*Branch N ts*)
using *build-tree′-termination entry prems*(*4*) *pred wf-tree-input-prered-pre* **by** *blast*
obtain *N-red ts-red* **where** *Nts-red*: *build-tree′ bs* ω *k red* = *Some* (*Branch N-red ts-red*)
using *build-tree′-termination entry prems*(*4*) *pred wf-tree-input-prered-red* **by** *blast*
have *simp*: *build-tree′ bs* ω *k i* = *Some* (*Branch N* (*ts* @ [*Branch N-red ts-red*]))
using *build-tree′-simps*(*8*) *entry pred Nts Nts-red* **by** *auto*
have *sound-ptrs* ω *bs*
using *prems*(*4*) *wf-tree-input-def* **by** *fastforce*
have *bounds′*: *k′* < *k pre* < *length* (*bs*!*k′*) *red* < *length* (*bs*!*k*)
using *pred entry bounds* ‹*sound-ptrs* ω *bs*›
unfolding *sound-prered-ptr-def sound-ptrs-def* **by** *fastforce*+
have *completes*: *completes k* (*fst* (*bs*!*k′*!*pre*)) (*fst* *e*) (*fst* (*bs*!*k*!*red*))
using *pred entry bounds* ‹*sound-ptrs* ω *bs*›
unfolding *sound-ptrs-def sound-prered-ptr-def* **by** *force*
have ∗:
lhs-item (*fst* (*bs*!*k′*!*pre*)) = *lhs-item* (*fst* *e*)
rhs-item (*fst* (*bs*!*k′*!*pre*)) = *rhs-item* (*fst* *e*)
dot-item (*fst* (*bs*!*k′*!*pre*)) + *1* = *dot-item* (*fst* *e*)
next-symbol (*fst* (*bs*!*k′*!*pre*)) = *Some* (*lhs-item* (*fst* (*bs*!*k*!*red*)))
is-complete (*fst* (*bs*!*k*!*red*))
using *completes* **unfolding** *completes-def inc-item-def*
by (*auto simp*: *lhs-item-def rhs-item-def is-complete-def*)
have (*bs*, ω, *k′*, *pre*) ∈ *wf-tree-input*
using *wf-tree-input-prered-pre*[*OF prems*(*4*) *entry pred*] **by** *blast*
hence *IH-pre*: *wf-item-tree* $\mathcal{G}$ (*fst* (*bs*!*k′*!*pre*)) (*Branch N ts*)
using *prems*(*2*)[*OF entry pred* - *prems*(*5*)] *Nts bounds*(*1*,*2*) *order-less-trans prems*(*6*) **by** *blast*

have (*bs*, ω, *k*, *red*) $\in$ *wf-tree-input*
using *wf-tree-input-prered-red*[*OF prems*(*4*) *entry prered*] **by** *blast*
hence *IH-r*: *wf-item-tree* $\mathcal{G}$ (*fst* (*bs*!*k*!*red*)) (*Branch N-red ts-red*)
using *bounds*$'$(*3*) *entry prems*(*3*,*5*,*6*) *prered Nts-red* **by** *blast*
have *map root* (*ts* @ [*Branch N-red ts-red*]) = *map root ts* @ [*root* (*Branch N-red ts*)]
by *simp*
also have ... = *take* (*dot-item* (*fst* (*bs*!*k*$'$!*pre*))) (*rhs-item* (*fst* (*bs*!*k*$'$!*pre*))) @ [*root* (*Branch N-red ts*)]
using *IH-pre* **by** *simp*
also have ... = *take* (*dot-item* (*fst* (*bs*!*k*$'$!*pre*))) (*rhs-item* (*fst* (*bs*!*k*$'$!*pre*))) @ [*lhs-item* (*fst* (*bs*!*k*!*red*))]
using *IH-r* **by** *simp*
also have ... = *take* (*dot-item* (*fst e*)) (*rhs-item* (*fst e*))
using $*$ **by** (*auto simp*: *next-symbol-def is-complete-def split*: *if-splits*; *metis leI take-Suc-conv-app-nth*)
finally have *roots*: *map root* (*ts* @ [*Branch N-red ts*]) = *take* (*dot-item* (*fst e*)) (*rhs-item* (*fst e*)) **by** *simp*
have *wf-item* $\mathcal{G}$ ω (*fst* (*bs*!*k*!*red*))
using *bounds bounds*$'$(*3*) *prems*(*5*) *wf-bins-def wf-bin-def wf-bin-items-def*
by (*metis items-def length-map nth-map nth-mem*)
moreover have *N-red* = *lhs-item* (*fst* (*bs*!*k*!*red*))
using *IH-r* **by** *fastforce*
moreover have *map root ts-red* = *rhs-item* (*fst* (*bs*!*k*!*red*))
using *IH-r* $*$(*5*) **by** (*auto simp*: *is-complete-def*)
ultimately have $\exists$ *r* $\in$ *set* ($\mathfrak{R}$ $\mathcal{G}$). *N-red* = *lhs-rule r* $\wedge$ *map root ts-red* = *rhs-rule r*
unfolding *wf-item-def rhs-item-def lhs-item-def* **by** *blast*
hence *wf-rule-tree* $\mathcal{G}$ (*Branch N-red ts-red*)
using *IH-r* **by** *simp*
hence *wf-item-tree* $\mathcal{G}$ (*fst* (*bs*!*k*!*i*)) (*Branch N* (*ts* @ [*Branch N-red ts-red*]))
using $*$(*1*) *roots IH-pre entry* **by** *simp*
thus *?thesis*
using *Nts-red prems*(*6*) *simp* **by** *auto*
qed
qed
done
thus *?thesis*
using *assms*(*2*) **by** *blast*
qed

lemma *wf-yield-build-tree*$'$:
assumes (*bs*, ω, *k*, *i*) $\in$ *wf-tree-input*
assumes *wf-bins* $\mathcal{G}$ ω *bs*
assumes *build-tree*$'$ *bs* ω *k i* = *Some t*
shows *wf-yield* ω (*fst* (*bs*!*k*!*i*)) *t*
proof $-$
have *wf-yield* ω (*fst* (*bs*!*k*!*i*)) *t*
using *assms*

apply (*induction arbitrary*: *t rule*: *build-tree′-induct*[*OF assms*(*1*)])
subgoal premises *prems* **for** *bs ω k i t*
proof −
define *e* **where** *entry*: *e* = *bs*!*k*!*i*
have *bounds*: *k* < *length bs k* ≤ *length ω i* < *length* (*bs*!*k*)
using *prems*(*4*) *wf-tree-input-def* **by** *force*+
consider (*Null*) *snd e* = *Null*
| (*Pre*) ∃ *pre*. *snd e* = *Pre pre*
| (*PreRed*) ∃ *k′ pre red reds*. *snd e* = *PreRed* (*k′*, *pre*, *red*) *reds*
by (*metis pointer*.*exhaust surj-pair*)
thus *?thesis*
proof *cases*
case *Null*
hence *build-tree′ bs ω k i* = *Some* (*Branch* (*lhs-item* (*fst e*)) [])
using *entry* **by** *simp*
have *simp*: *t* = *Branch* (*lhs-item* (*fst e*)) []
using *build-tree′-simps*(*1*) *Null prems*(*6*) *entry* **by** *simp*
have *sound-ptrs ω bs*
using *prems*(*4*) **unfolding** *wf-tree-input-def* **by** *blast*
hence *predicts* (*fst e*)
using *Null bounds nth-mem entry* **unfolding** *sound-ptrs-def sound-null-ptr-def* **by** *blast*
thus *?thesis*
unfolding *wf-yield-def predicts-def* **using** *simp entry* **by** (*auto simp*: *slice-empty*)
next
case *Pre*
then obtain *pre* **where** *pre*: *snd e* = *Pre pre*
by *blast*
obtain *N ts* **where** *Nts*: *build-tree′ bs ω* (*k*−*1*) *pre* = *Some* (*Branch N ts*)
using *build-tree′-termination entry pre prems*(*4*) *wf-tree-input-pre* **by** *blast*
have *simp*: *build-tree′ bs ω k i* = *Some* (*Branch N* (*ts* @ [*Leaf* (*ω*!(*k*−*1*))]))
using *build-tree′-simps*(*3*) *entry pre Nts* **by** *simp*
have *sound-ptrs ω bs*
using *prems*(*4*) **unfolding** *wf-tree-input-def* **by** *blast*
hence *bounds′*: *k* > *0 pre* < *length* (*bs*!(*k*−*1*))
using *entry pre bounds* **unfolding** *sound-ptrs-def sound-pre-ptr-def* **by** (*metis nth-mem*)+
moreover have *k*−*1* < *length bs*
by (*simp add*: *bounds less-imp-diff-less*)
ultimately have *IH*: *wf-yield ω* (*fst* (*bs*!(*k*−*1*)!*pre*)) (*Branch N ts*)
using *prems*(*1*) *entry pre Nts wf-tree-input-pre prems*(*4*,*5*,*6*) **by** *fastforce*
have *scans*: *scans ω k* (*fst* (*bs*!(*k*−*1*)!*pre*)) (*fst e*)
using *entry pre bounds* ‹*sound-ptrs ω bs*› **unfolding** *sound-ptrs-def sound-pre-ptr-def* **by** *simp*
have *wf*:
start-item (*fst* (*bs*!(*k*−*1*)!*pre*)) ≤ *end-item* (*fst* (*bs*!(*k*−*1*)!*pre*))
end-item (*fst* (*bs*!(*k*−*1*)!*pre*)) = *k*−*1*
end-item (*fst e*) = *k*

using *entry prems*(*5*) *bounds′ bounds* **unfolding** *wf-bins-def wf-bin-def wf-bin-items-def items-def wf-item-def*
by (*auto*, *meson less-imp-diff-less nth-mem*)
have *yield* (*Branch N* (*ts* @ [*Leaf* (*ω*!(*k−1*))])) = *concat* (*map yield* (*ts* @ [*Leaf* (*ω*!(*k−1*))]))
by *simp*
also have ... = *concat* (*map yield ts*) @ [*ω*!(*k−1*)]
by *simp*
also have ... = *slice ω* (*start-item* (*fst* (*bs*!(*k−1*)!*pre*))) (*end-item* (*fst* (*bs*!(*k−1*)!*pre*))) @ [*ω*!(*k−1*)]
using *IH* **by** (*simp add*: *wf-yield-def*)
also have ... = *slice ω* (*start-item* (*fst* (*bs*!(*k−1*)!*pre*))) (*end-item* (*fst* (*bs*!(*k−1*)!*pre*)) + *1*)
using *slice-append-nth wf* ‹*k* > *0*›
by (*metis One-nat-def Suc-pred bounds*(*2*) *le-neq-implies-less lessI less-imp-diff-less*)
also have ... = *slice ω* (*start-item* (*fst e*)) (*end-item* (*fst* (*bs*!(*k−1*)!*pre*)) + *1*)
using *scans* **unfolding** *scans-def inc-item-def* **by** *simp*
also have ... = *slice ω* (*start-item* (*fst e*)) *k*
using *scans wf* **unfolding** *scans-def* **by** (*metis Suc-diff-1 Suc-eq-plus1 bounds′*(*1*))
also have ... = *slice ω* (*start-item* (*fst e*)) (*end-item* (*fst e*))
using *wf* **by** *auto*
finally show *?thesis*
using *wf-yield-def entry prems*(*6*) *simp* **by** *force*
next
case *PreRed*
then obtain *k′ pre red ps* **where** *prered*: *snd e* = *PreRed* (*k′*, *pre*, *red*) *ps*
by *blast*
obtain *N ts* **where** *Nts*: *build-tree′ bs ω k′ pre* = *Some* (*Branch N ts*)
using *build-tree′-termination entry prems*(*4*) *prered wf-tree-input-prered-pre* **by** *blast*
obtain *N-red ts-red* **where** *Nts-red*: *build-tree′ bs ω k red* = *Some* (*Branch N-red ts-red*)
using *build-tree′-termination entry prems*(*4*) *prered wf-tree-input-prered-red* **by** *blast*
have *simp*: *build-tree′ bs ω k i* = *Some* (*Branch N* (*ts* @ [*Branch N-red ts-red*]))
using *build-tree′-simps*(*8*) *entry prered Nts Nts-red* **by** *auto*
have *sound-ptrs ω bs*
using *prems*(*4*) *wf-tree-input-def* **by** *fastforce*
have *bounds′*: *k′* < *k pre* < *length* (*bs*!*k′*) *red* < *length* (*bs*!*k*)
using *prered entry bounds* ‹*sound-ptrs ω bs*›
unfolding *sound-ptrs-def sound-prered-ptr-def* **by** *fastforce*+
have *completes*: *completes k* (*fst* (*bs*!*k′*!*pre*)) (*fst e*) (*fst* (*bs*!*k*!*red*))
using *prered entry bounds* ‹*sound-ptrs ω bs*›
unfolding *sound-ptrs-def sound-prered-ptr-def* **by** *fastforce*
have (*bs*, *ω*, *k′*, *pre*) ∈ *wf-tree-input*
using *wf-tree-input-prered-pre*[*OF prems*(*4*) *entry prered*] **by** *blast*

hence *IH-pre*: *wf-yield* ω (*fst* (*bs*!*k'*!*pre*)) (*Branch N ts*)
using *prems*(*2*)[*OF entry prered - prems*(*5*)] *Nts bounds'*(*1*,*2*) *prems*(*6*)
by (*meson dual-order.strict-trans1 nat-less-le*)
have (*bs*, ω, *k*, *red*) $\in$ *wf-tree-input*
using *wf-tree-input-prered-red*[*OF prems*(*4*) *entry prered*] **by** *blast*
hence *IH-r*: *wf-yield* ω (*fst* (*bs*!*k*!*red*)) (*Branch N-red ts-red*)
using *bounds*(*3*) *entry prems*(*3*,*5*,*6*) *prered Nts-red* **by** *blast*
have *wf1*:
start-item (*fst* (*bs*!*k'*!*pre*)) $\leq$ *end-item* (*fst* (*bs*!*k'*!*pre*))
start-item (*fst* (*bs*!*k*!*red*)) $\leq$ *end-item* (*fst* (*bs*!*k*!*red*))
using *prems*(*5*) *bounds bounds'* **unfolding** *wf-bins-def wf-bin-def wf-bin-items-def items-def wf-item-def*
by (*metis length-map nth-map nth-mem order.strict-trans*)+
have *wf2*:
end-item (*fst* (*bs*!*k*!*red*)) = *k*
end-item (*fst* (*bs*!*k*!*i*)) = *k*
using *prems*(*5*) *bounds bounds'* **unfolding** *wf-bins-def wf-bin-def wf-bin-items-def items-def* **by** *simp-all*
have *yield* (*Branch N* (*ts* @ [*Branch N-red ts-red*])) = *concat* (*map yield* (*ts* @ [*Branch N-red ts-red*]))
by (*simp add*: *Nts-red*)
also have ... = *concat* (*map yield ts*) @ *yield* (*Branch N-red ts-red*)
by *simp*
also have ... = *slice* ω (*start-item* (*fst* (*bs*!*k'*!*pre*))) (*end-item* (*fst* (*bs*!*k'*!*pre*))) @
slice ω (*start-item* (*fst* (*bs*!*k*!*red*))) (*end-item* (*fst* (*bs*!*k*!*red*)))
using *IH-pre IH-r* **by** (*simp add*: *wf-yield-def*)
also have ... = *slice* ω (*start-item* (*fst* (*bs*!*k'*!*pre*))) (*end-item* (*fst* (*bs*!*k*!*red*)))
using *slice-concat wf1 completes-def completes* **by** (*metis* (*no-types*, *lifting*))
also have ... = *slice* ω (*start-item* (*fst e*)) (*end-item* (*fst* (*bs*!*k*!*red*)))
using *completes* **unfolding** *completes-def inc-item-def* **by** *simp*
also have ... = *slice* ω (*start-item* (*fst e*)) (*end-item* (*fst e*))
using *wf2 entry* **by** *presburger*
finally show *?thesis*
using *wf-yield-def entry prems*(*6*) *simp* **by** *force*
qed
qed
done
thus *?thesis*
using *assms*(*2*) **by** *blast*
qed

theorem *wf-rule-root-yield-build-tree*:
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 t* = $\mathfrak{S}$ $\mathcal{G}$ $\wedge$ *yield t* = ω
proof −
let *?k* = *length bs* − *1*

define *finished* **where** *finished-def*: *finished* = *filter-with-index* (*is-finished* $\mathcal{G}$ ω) (*items* (*bs*!*?k*))
then obtain *x i* **where** ∗: (*x*,*i*) ∈ *set finished Some t* = *build-tree′ bs* ω *?k i*
using *assms*(*5*) **unfolding** *finished-def build-tree-def* **by** (*auto simp*: *Let-def split*: *list.splits*, *presburger*)
have *k*: *?k* < *length bs ?k* ≤ *length* ω
using *assms*(*4*) **by** *simp-all*
have *i*: *i* < *length* (*bs*!*?k*)
using *index-filter-with-index-lt-length* ∗ *items-def finished-def* **by** (*metis length-map*)
have *x*: *x* = *fst* (*bs*!*?k*!*i*)
using ∗ *i filter-with-index-nth items-def nth-map finished-def* **by** *metis*
have *finished*: *is-finished* $\mathcal{G}$ ω *x*
using ∗ *filter-with-index-P finished-def* **by** *metis*
have *wf-trees-input*: (*bs*, ω, *?k*, *i*) ∈ *wf-tree-input*
unfolding *wf-tree-input-def* **using** *assms*(*2*,*3*) *i k* **by** *blast*
hence *wf-item-tree*: *wf-item-tree* $\mathcal{G}$ *x t*
using *wf-item-tree-build-tree′ assms*(*1*,*2*) *i k*(*1*) *x* ∗(*2*) **by** *metis*
have *wf-item*: *wf-item* $\mathcal{G}$ ω (*fst* (*bs*!*?k*!*i*))
using *k*(*1*) *i assms*(*1*) **unfolding** *wf-bins-def wf-bin-def wf-bin-items-def* **by** (*simp add*: *items-def*)
obtain *N ts* **where** *t*: *t* = *Branch N ts*
by (*metis* ∗(*2*) *build-tree′-termination option.inject wf-trees-input*)
hence *N* = *lhs-item x*
map root ts = *rhs-item x*
using *finished wf-item-tree* **by** (*auto simp*: *is-finished-def is-complete-def*)
hence ∃ *r* ∈ *set* ($\mathfrak{R}$ $\mathcal{G}$). *N* = *lhs-rule r* ∧ *map root ts* = *rhs-rule r*
using *wf-item x* **unfolding** *wf-item-def rhs-item-def lhs-item-def* **by** *blast*
hence *wf-rule*: *wf-rule-tree* $\mathcal{G}$ *t*
using *wf-item-tree t* **by** *simp*
have *root*: *root t* = $\mathfrak{S}$ $\mathcal{G}$
using *finished t* ‹*N* = *lhs-item x*› **by** (*auto simp*: *is-finished-def*)
have *yield t* = *slice* ω (*start-item* (*fst* (*bs*!*?k*!*i*))) (*end-item* (*fst* (*bs*!*?k*!*i*)))
using *k i assms*(*1*) *wf-trees-input wf-yield-build-tree′ wf-yield-def* ∗(*2*) **by** (*metis* (*no-types*, *lifting*))
hence *yield*: *yield t* = ω
using *finished x* **unfolding** *is-finished-def* **by** *simp*
show *?thesis*
using ∗ *wf-rule root yield assms*(*4*) **unfolding** *build-tree-def* **by** *simp*
qed

corollary *wf-rule-root-yield-build-tree-Earley*$_L$:
assumes ε*-free* $\mathcal{G}$
assumes *build-tree* $\mathcal{G}$ ω (*Earley*$_L$ $\mathcal{G}$ ω) = *Some t*
shows *wf-rule-tree* $\mathcal{G}$ *t* ∧ *root t* = $\mathfrak{S}$ $\mathcal{G}$ ∧ *yield t* = ω
using *assms wf-rule-root-yield-build-tree wf-bins-Earley*$_L$ *sound-mono-ptrs-Earley*$_L$ *Earley*$_L$*-def*
length-Earley$_L$*-bins length-bins-Init*$_L$ **by** (*metis* ε*-free-impl-nonempty-derives le-refl*)

theorem *correctness-build-tree-Earley*$_L$:
assumes *is-word* $\mathcal{G}$ ω ε*-free* $\mathcal{G}$
shows ($\exists$ *t*. *build-tree* $\mathcal{G}$ ω (*Earley*$_L$ $\mathcal{G}$ ω) = *Some t*) $\longleftrightarrow$ $\mathcal{G} \vdash [\mathfrak{S}\ \mathcal{G}] \Rightarrow^* \omega$ (**is** *?L* $\longleftrightarrow$ *?R*)
proof *standard*
assume $*$: *?L*
let *?k* = *length* (*Earley*$_L$ $\mathcal{G}$ ω) − *1*
define *finished* **where** *finished-def*: *finished* = *filter-with-index* (*is-finished* $\mathcal{G}$ ω) (*items* ((*Earley*$_L$ $\mathcal{G}$ ω)!*?k*))
then obtain *t x i* **where** $*$: (*x*,*i*) $\in$ *set finished Some t* = *build-tree'* (*Earley*$_L$ $\mathcal{G}$ ω) ω *?k i*
using $*$ **unfolding** *finished-def build-tree-def* **by** (*auto simp*: *Let-def split*: *list.splits*, *presburger*)
have *k*: *?k* < *length* (*Earley*$_L$ $\mathcal{G}$ ω) *?k* $\leq$ *length* ω
by (*simp-all add*: *Earley*$_L$*-def assms*(*1*))
have *i*: *i* < *length* ((*Earley*$_L$ $\mathcal{G}$ ω) ! *?k*)
using *index-filter-with-index-lt-length* $*$ *items-def finished-def* **by** (*metis length-map*)
have *x*: *x* = *fst* ((*Earley*$_L$ $\mathcal{G}$ ω)!*?k*!*i*)
using $*$ *i filter-with-index-nth items-def nth-map finished-def* **by** *metis*
have *finished*: *is-finished* $\mathcal{G}$ ω *x*
using $*$ *filter-with-index-P finished-def* **by** *metis*
moreover have *x* $\in$ *set* (*items* ((*Earley*$_L$ $\mathcal{G}$ ω) ! *?k*))
using *x* **by** (*auto simp*: *items-def*; *metis One-nat-def i imageI nth-mem*)
ultimately have *recognizing* (*bins* (*Earley*$_L$ $\mathcal{G}$ ω)) $\mathcal{G}$ ω
by (*meson k*(*1*) *kth-bin-sub-bins recognizing-def subsetD*)
thus *?R*
using *soundness-Earley*$_L$ **by** *blast*
next
assume $*$: *?R*
let *?k* = *length* (*Earley*$_L$ $\mathcal{G}$ ω) − *1*
define *finished* **where** *finished-def*: *finished* = *filter-with-index* (*is-finished* $\mathcal{G}$ ω) (*items* ((*Earley*$_L$ $\mathcal{G}$ ω)!*?k*))
have *recognizing* (*bins* (*Earley*$_L$ $\mathcal{G}$ ω)) $\mathcal{G}$ ω
using *assms* $*$ *completeness-Earley*$_L$ **by** *blast*
moreover have *?k* = *length* ω
by (*simp add*: *Earley*$_L$*-def assms*(*1*))
ultimately have $\exists$ *x* $\in$ *set* (*items* ((*Earley*$_L$ $\mathcal{G}$ ω)!*?k*)). *is-finished* $\mathcal{G}$ ω *x*
unfolding *recognizing-def* **using** *assms*(*1*) *is-finished-def wf-bins-Earley*$_L$ *wf-item-in-kth-bin* **by** *metis*
then obtain *x i xs* **where** *xis*: *finished* = (*x*,*i*)#*xs*
using *filter-with-index-Ex-first* **by** (*metis finished-def*)
hence *simp*: *build-tree* $\mathcal{G}$ ω (*Earley*$_L$ $\mathcal{G}$ ω) = *build-tree'* (*Earley*$_L$ $\mathcal{G}$ ω) ω *?k i*
unfolding *build-tree-def finished-def* **by** *auto*
have (*x*,*i*) $\in$ *set finished*
using *xis* **by** *simp*
hence *i* < *length* ((*Earley*$_L$ $\mathcal{G}$ ω)!*?k*)
using *index-filter-with-index-lt-length* **by** (*metis finished-def items-def length-map*)
moreover have *?k* < *length* (*Earley*$_L$ $\mathcal{G}$ ω)
by (*simp add*: *Earley*$_L$*-def assms*(*1*))

ultimately have $(Earley_L\ \mathcal{G}\ \omega,\ \omega,\ ?k,\ i) \in wf\text{-}tree\text{-}input$
unfolding *wf-tree-input-def* **using** *sound-mono-ptrs-Earley*$_L$ *assms* ε*-free-impl-nonempty-derives*
using ‹$length\ (Earley_L\ \mathcal{G}\ \omega) - 1 = length\ \omega$› **by** *auto*
then obtain *N ts* **where** $build\text{-}tree'\ (Earley_L\ \mathcal{G}\ \omega)\ \omega\ ?k\ i = Some\ (Branch\ N$
$ts)$
using *build-tree′-termination* **by** *blast*
thus *?L*
using *simp* **by** *simp*
qed

end
theory *Examples*
imports
Earley-Parser
HOL−Library.Code-Target-Nat
begin

10 Examples

10.1 Common symbols

datatype *symbol* = *a* | *S* | *X* | *Y* | *Z*

10.2 $O(n^3)$ ambiguous grammars

10.2.1 S -> SS | a

definition *rules1* :: *symbol rule list* **where**
rules1 = [
(*S*, [*S*, *S*]),
(*S*, [*a*])
]

definition *cfg1* :: *symbol cfg* **where**
cfg1 = *CFG rules1 S*

lemma ε*-free1*:
ε*-free cfg1*
by (*auto simp*: ε*-free-def cfg1-def rules1-def rhs-rule-def*)

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

10.3.1 S -> aS | a

definition *rules2* :: *symbol rule list* **where**
rules2 = [
(*S*, [*a*, *S*]),
(*S*, [*a*])
]

definition *cfg2* :: *symbol cfg* **where**
cfg2 = *CFG rules2 S*

lemma *ε-free2*:
ε-free cfg2
by (*auto simp*: *ε-free-def cfg2-def rules2-def rhs-rule-def*)

10.3.2 S -> aSa | a

definition *rules3* :: *symbol rule list* **where**
rules3 = [
(*S*, [*a*, *S*, *a*]),
(*S*, [*a*])
]

definition *cfg3* :: *symbol cfg* **where**
cfg3 = *CFG rules3 S*

lemma *ε-free3*:
ε-free cfg3
by (*auto simp*: *ε-free-def cfg3-def rules3-def rhs-rule-def*)

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

10.4.1 S -> Sa | a

definition *rules4* :: *symbol rule list* **where**
rules4 = [
(*S*, [*S*, *a*]),
(*S*, [*a*])
]

definition *cfg4* :: *symbol cfg* **where**
cfg4 = *CFG rules4 S*

lemma *ε-free4*:
ε-free cfg4
by (*auto simp*: *ε-free-def cfg4-def rules4-def rhs-rule-def*)

10.5 S -> SX, X -> Y | Z, Y -> a, Z -> a

definition *rules5* :: *symbol rule list* **where**
rules5 = [
(*S*, [*S*, *X*]),
(*S*, [*a*]),
(*X*, [*Y*]),
(*X*, [*Z*]),
(*Y*, [*a*]),
(*Z*, [*a*])

]

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

lemma *ε-free5*:
ε-free cfg5
by (*auto simp*: *ε-free-def cfg5-def rules5-def rhs-rule-def*)

11 Input and Evaluation

definition *inp* :: *symbol list* **where**
inp = [*a*,
a, *a*, *a*, *a*, *a*, *a*, *a*, *a*, *a*, *a*, *a*, *a*, *a*, *a*, *a*, *a*,
a, *a*, *a*, *a*, *a*, *a*, *a*, *a*, *a*, *a*, *a*, *a*, *a*, *a*, *a*, *a*,
a, *a*, *a*, *a*, *a*, *a*, *a*, *a*, *a*, *a*, *a*, *a*, *a*, *a*, *a*, *a*,
a, *a*, *a*, *a*, *a*, *a*, *a*, *a*, *a*, *a*, *a*, *a*, *a*, *a*, *a*, *a*
]

lemma *is-word-inp1*:
is-word cfg1 inp
by (*auto simp*: *is-word-def cfg1-def rules1-def nonterminals-def inp-def*)

lemma *is-word-inp2*:
is-word cfg2 inp
by (*auto simp*: *is-word-def cfg2-def rules2-def nonterminals-def inp-def*)

lemma *is-word-inp3*:
is-word cfg3 inp
by (*auto simp*: *is-word-def cfg3-def rules3-def nonterminals-def inp-def*)

lemma *is-word-inp4*:
is-word cfg4 inp
by (*auto simp*: *is-word-def cfg4-def rules4-def nonterminals-def inp-def*)

lemma *is-word-inp5*:
is-word cfg5 inp
by (*auto simp*: *is-word-def cfg5-def rules5-def nonterminals-def inp-def*)

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

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

definition *size-pointers* :: *'a bins* ⇒ *nat* **where**
size-pointers bs = *fold* (+) (*map* (λ*b*. *fold* (+) (*map* (λ*e*. *size-pointer e*) *b*) *0*) *bs*)
0

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

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

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

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

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

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

end

References

[1] J. Earley. An efficient context-free parsing algorithm. *Commun. ACM*, 13(2):94102, 1970.

[2] C. B. Jones. Formal development of correct algorithms: An example based on earley's recogniser. In *Proceedings of ACM Conference on Proving Assertions about Programs*, page 150169, New York, NY, USA, 1972. Association for Computing Machinery.

[3] S. Obua. Local lexing. *Archive of Formal Proofs*, 2017. https://isa-afp.org/entries/LocalLexing.html, Formal proof development.

[4] S. Obua, P. Scott, and J. Fleuriot. Local lexing, 2017.

[5] E. Scott. Sppf-style parsing from earley recognisers. *Electronic Notes in Theoretical Computer Science*, 203(2):53–67, 2008. Proceedings of the Seventh Workshop on Language Descriptions, Tools, and Applications (LDTA 2007).