

Earley

Martin Rau

February 6, 2026

Abstract

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

Contents

1	Slightly adjusted content from AFP/LocalLexing	2
2	Adjusted content from AFP/LocalLexing	6
3	Adjusted content from AFP/LocalLexing	7
4	Additional derivation lemmas	9
5	Slices	12
6	Earley recognizer	13
6.1	Earley items	13
6.2	Well-formedness	15
6.3	Soundness	15
6.4	Completeness	18
6.5	Correctness	21
6.6	Finiteness	21

7	Earley fixpoint	22
7.1	Definitions	22
7.2	Monotonicity and Absorption	23
7.3	Soundness	27
7.4	Completeness	28
7.5	Correctness	33
8	Earley recognizer	33
8.1	List auxiliaries	33
8.2	Definitions	35
8.3	Epsilon productions	37
8.4	Bin lemmas	38
8.5	Well-formed bins	49
8.6	Soundness	59
8.7	Completeness	65
8.8	Correctness	85
9	Earley parser	86
9.1	Pointer lemmas	86
9.2	Common Definitions	98
9.3	foldl lemmas	99
9.4	Parse tree	101
10	Examples	115
10.1	Common symbols	115
10.2	$O(n^3)$ ambiguous grammars	115
10.2.1	$S \rightarrow SS \mid a$	115
10.3	$O(n^2)$ unambiguous or bounded ambiguity	115
10.3.1	$S \rightarrow aS \mid a$	115
10.3.2	$S \rightarrow aSa \mid a$	116
10.4	$O(n)$ bounded state, non-right recursive LR(k) grammars	116
10.4.1	$S \rightarrow Sa \mid a$	116
10.5	$S \rightarrow SX, X \rightarrow Y \mid Z, Y \rightarrow a, Z \rightarrow a$	116
11	Input and Evaluation	117
theory <i>Limit</i>		
imports		
<i>Main</i>		
begin		

1 Slightly adjusted content from AFP/LocalLexing

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

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

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

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

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

lemma $\text{subset-setmonotone}: \text{setmonotone } f \Longrightarrow X \subseteq f \text{ } X$
by ($\text{simp add: setmonotone-def}$)

lemma[simp]: $\text{funpower } \text{id} \text{ } n = \text{id}$
by ($\text{rule ext, induct } n, \text{simp-all}$)

lemma[simp]: $\text{limit } \text{id} = \text{id}$
by ($\text{rule ext, auto simp add: limit-def natUnion-def}$)

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

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

lemma $\text{natUnion-upperbound}$:
 $(\bigwedge n. f \text{ } n \subseteq G) \Longrightarrow (\text{natUnion } f) \subseteq G$
by ($\text{auto simp add: natUnion-def}$)

lemma $\text{funpower-upperbound}$:
 $(\bigwedge I. I \subseteq G \Longrightarrow f \text{ } I \subseteq G) \Longrightarrow I \subseteq G \Longrightarrow \text{funpower } f \text{ } n \text{ } I \subseteq G$
proof ($\text{induct } n$)
 case 0 **thus** ?case **by** simp
 next
 case (Suc n) **thus** ?case **by** simp
qed

lemma limit-upperbound :
 $(\bigwedge I. I \subseteq G \Longrightarrow f \text{ } I \subseteq G) \Longrightarrow I \subseteq G \Longrightarrow \text{limit } f \text{ } I \subseteq G$
by ($\text{simp add: funpower-upperbound limit-def natUnion-upperbound}$)

lemma elem-limit-simp : $x \in \text{limit } f \text{ } X = (\exists n. x \in \text{funpower } f \text{ } n \text{ } X)$
by ($\text{auto simp add: limit-def natUnion-def}$)

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

$pointwise\ f = (\forall\ X.\ f\ X = \bigcup\ \{ f\ \{x\} \mid x.\ x \in X\})$

lemma *natUnion-elem*: $x \in f\ n \implies x \in natUnion\ f$
using *natUnion-def* **by** *fastforce*

lemma *limit-elem*: $x \in funpower\ f\ n\ X \implies x \in limit\ f\ X$
by (*simp add: limit-def natUnion-elem*)

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

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

lemma *chain-implies-mono*: $chain\ C \implies mono\ C$
by (*simp add: chain-def mono-iff-le-Suc*)

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

lemma *natUnion-subset*: $(\bigwedge\ n.\ \exists\ m.\ f\ n \subseteq g\ m) \implies natUnion\ f \subseteq natUnion\ g$
by (*meson natUnion-elem natUnion-upperbound subset-iff*)

lemma *natUnion-eq*[*case-names Subset Superset*]:
 $(\bigwedge\ n.\ \exists\ m.\ f\ n \subseteq g\ m) \implies (\bigwedge\ n.\ \exists\ m.\ g\ n \subseteq f\ m) \implies natUnion\ f = natUnion\ g$
by (*simp add: natUnion-subset subset-antisym*)

lemma *natUnion-shift*[*symmetric*]:
assumes *chain*: $chain\ C$
shows $natUnion\ C = natUnion\ (\lambda\ n.\ C\ (n + m))$
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* :: $('a\ set \Rightarrow 'a\ set) \Rightarrow bool$
where
 $regular\ f = (setmonotone\ f \wedge continuous\ f)$

lemma *regular-fixpoint*:
assumes *regular*: $regular\ f$
shows $f\ (limit\ f\ I) = limit\ f\ I$
proof –

```

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

let ?C =  $\lambda n. \text{funpower } f \ n \ I$ 
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 ( $\lambda n. f(\text{funpower } f \ n \ I)$ )
  by (meson comp-apply)
also have natUnion ( $\lambda n. f(\text{funpower } f \ n \ I)$ ) = natUnion ( $\lambda n. ?C (n + 1)$ )
  by simp
also have natUnion ( $\lambda n. ?C(n + 1)$ ) = 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:  $\bigwedge n. \text{funpower } f \ n \ I = I$ 
  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  $\implies$  limit f (limit f I) = limit f I
by (simp add: fix-is-fix-of-limit regular-fixpoint)

```

```

definition mk-regular1 :: ('b  $\implies$  'a  $\implies$  bool)  $\implies$  ('b  $\implies$  'a  $\implies$  'a)  $\implies$  'a set  $\implies$  'a set
where
  mk-regular1 P F I = I  $\cup$  { F q x | q x. x  $\in$  I  $\wedge$  P q x }

```

```

definition mk-regular2 :: ('b  $\implies$  'a  $\implies$  'a  $\implies$  bool)  $\implies$  ('b  $\implies$  'a  $\implies$  'a  $\implies$  'a)  $\implies$  'a set
 $\implies$  'a set where
  mk-regular2 P F I = I  $\cup$  { F q x y | q x y. x  $\in$  I  $\wedge$  y  $\in$  I  $\wedge$  P q x y }

```

```

end
theory CFG
  imports Main
begin

```

2 Adjusted content from AFP/LocalLexing

type-synonym 'a rule = 'a × 'a list

type-synonym 'a rules = 'a rule list

datatype 'a cfg = CFG (ℳ : 'a rules) (℄ : 'a)

definition nonterminals :: 'a cfg ⇒ 'a set **where**
 nonterminals G = set (map fst (ℳ G)) ∪ {℄ G}

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

definition derives1 :: 'a cfg ⇒ 'a list ⇒ 'a list ⇒ bool **where**
 derives1 G u v ≡ ∃ x y A α.
 u = x @ [A] @ y ∧
 v = x @ α @ y ∧
 (A, α) ∈ set (ℳ G)

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

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

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

by (auto simp add: derives-def derivations-def derivations1-def)

lemma *derives-trans*:

derives \mathcal{G} a b \implies *derives* \mathcal{G} b c \implies *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*: $\bigwedge y z. \text{derives } \mathcal{G} \ a \ y \implies \text{derives1 } \mathcal{G} \ y \ z \implies P \ y \implies P \ z$

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 \Rightarrow 'a list \Rightarrow nat \Rightarrow 'a rule \Rightarrow 'a list \Rightarrow bool **where**

Derives1 \mathcal{G} u i r v \equiv $\exists x y A \alpha.$

u = x @ [A] @ y \wedge

v = x @ α @ y \wedge

(A, α) ∈ set (\mathfrak{R} \mathcal{G}) \wedge r = (A, α) \wedge i = length x

lemma *Derives1-split*:

Derives1 \mathcal{G} u i r v \implies $\exists x y. u = x @ [\text{fst } r] @ y \wedge v = x @ (\text{snd } r) @ y \wedge$
length x = i

by (*metis Derives1-def fst-conv snd-conv*)

lemma *Derives1-implies-derives1*: *Derives1* \mathcal{G} u i r v \implies *derives1* \mathcal{G} u v

by (auto simp add: *Derives1-def derives1-def*)

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

by (auto simp add: *Derives1-def derives1-def*)

fun *Derivation* :: 'a cfg \Rightarrow 'a list \Rightarrow 'a derivation \Rightarrow 'a list \Rightarrow bool **where**

Derivation - a [] b = (a = b)

| *Derivation* $\mathcal{G} a (d\#D) b = (\exists x. \text{Derives1 } \mathcal{G} a (\text{fst } d) (\text{snd } d) x \wedge \text{Derivation } \mathcal{G} x D b)$

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

proof (*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** $\exists x. \text{Derives1 } \mathcal{G} a (\text{fst } d) (\text{snd } d) x \wedge \text{Derivation } \mathcal{G} x D b$ **by** *auto*

then obtain x **where** $\text{Derives1 } \mathcal{G} a (\text{fst } d) (\text{snd } d) x$ **and** $xb: \text{Derivation } \mathcal{G} x D b$ **by** *blast*

with *Derives1-implies-derives1* **have** $d1: \text{derives } \mathcal{G} a x$ **by** *fastforce*

from *ihyps* xb **have** $d2: \text{derives } \mathcal{G} x b$ **by** *simp*

show $\text{derives } \mathcal{G} a b$ **by** (*rule* *derives-trans*[*OF* $d1 d2$])

qed

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

proof (*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*: $\text{derives } \mathcal{G} a b \implies \exists D. \text{Derivation } \mathcal{G} a D b$

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** $ay: \text{Derivation } \mathcal{G} a D y$ **by** *blast*

from *ihyps* *derives1-implies-Derives1* **obtain** $i r$ **where** $yz: \text{Derives1 } \mathcal{G} y i r z$ **by** *blast*

from *Derivation-Derives1*[*OF* $ay yz$] **show** *?case* **by** *auto*

qed

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

using *Derives1-def* **by** *metis*

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

by (*induct* D *arbitrary*: $a c E$) *auto*

lemma *Derivation-implies-append*:

Derivation $\mathcal{G} a D b \implies \text{Derivation } \mathcal{G} b E c \implies \text{Derivation } \mathcal{G} a (D@E) c$
using *Derivation-append by blast*

4 Additional derivation lemmas

lemma *Derives1-prepend:*

assumes *Derives1* $\mathcal{G} u i r v$

shows *Derives1* $\mathcal{G} (w@u) (i + \text{length } w) r (w@v)$

proof –

obtain $x y A \alpha$ **where** *:

$u = x @ [A] @ y \ v = x @ \alpha @ y$

$(A, \alpha) \in \text{set } (\mathfrak{R} \ \mathcal{G}) \ r = (A, \alpha) \ i = \text{length } x$

using *assms Derives1-def* **by** (*smt (verit)*)

hence $w@u = w @ x @ [A] @ y \ w@v = w @ x @ \alpha @ y$

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 $\mathcal{G} b D b' \implies \text{Derivation } \mathcal{G} (a@b) (\text{map } (\lambda(i, r). (i + \text{length } a, r)) D)$
 $(a@b')$

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 $x y A \alpha$ **where** *:

$u = x @ [A] @ y \ v = x @ \alpha @ y$

$(A, \alpha) \in \text{set } (\mathfrak{R} \ \mathcal{G}) \ r = (A, \alpha) \ i = \text{length } x$

using *assms Derives1-def* **by** (*smt (verit)*)

hence $u@w = x @ [A] @ y @ w \ v@w = x @ \alpha @ y @ w$

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 $\mathcal{G} a D a' \implies \text{Derivation } \mathcal{G} (a@b) D (a'@b)$

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*:
 $(A, \alpha) \in \text{set } (\mathfrak{R} \mathcal{G}) \implies \text{derives1 } \mathcal{G} [A] \alpha$
unfolding *derives1-def*
apply (*rule-tac* *exI* [**where** $x = []$])
apply (*rule-tac* *exI* [**where** $x = []$])
by *simp*

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

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

lemma *Derivation-concat-split*:
Derivation \mathcal{G} $(a @ b)$ D $c \implies \exists E F a' b'$. *Derivation* \mathcal{G} a E $a' \wedge$ *Derivation* \mathcal{G} b F $b' \wedge$
 $c = a' @ b' \wedge \text{length } E \leq \text{length } D \wedge \text{length } F \leq \text{length } D$
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** $\#$:
 $a @ b = x @ [A] @ y$ $ab = x @ \alpha @ y$ $(A, \alpha) \in \text{set } (\mathfrak{R} \mathcal{G})$ $snd\ d = (A, \alpha)$ $fst\ d =$
 $\text{length } x$
using $*$ **unfolding** *Derives1-def* **by** *blast*
show *?case*
proof (*cases* $\text{length } a \leq \text{length } x$)
case *True*
hence *ab-def*:
 $a = \text{take } (\text{length } a) x$
 $b = \text{drop } (\text{length } a) x @ [A] @ y$
 $ab = \text{take } (\text{length } a) x @ \text{drop } (\text{length } a) x @ \alpha @ y$
using $\#(1,2)$ *True* **by** (*metis* *append-eq-append-conv-if*) $+$
then obtain E F a' b' **where** *IH*:
Derivation \mathcal{G} $(\text{take } (\text{length } a) x)$ E a'
Derivation \mathcal{G} $(\text{drop } (\text{length } a) x @ \alpha @ y)$ F b'
 $c = a' @ b'$

$length\ E \leq length\ D$
 $length\ F \leq length\ D$
using *Cons* *(2) **by** *blast*
have *Derives1* $\mathcal{G}\ b\ (fst\ d - length\ a)\ (snd\ d)\ (drop\ (length\ a)\ x\ @\ \alpha\ @\ y)$
unfolding *Derives1-def* **using** *(1) #(3-5) *ab-def*(2) **by** (*metis length-drop*)
hence *Derivation* $\mathcal{G}\ b\ ((fst\ d - length\ a,\ snd\ d)\ \#\ F)\ b'$
using *IH*(2) **by** *force*
moreover **have** *Derivation* $\mathcal{G}\ a\ E\ a'$
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 $ab = x\ @\ \alpha\ @\ take\ (length\ a - length\ x - 1)\ y\ @\ drop\ (length\ a - length\ x - 1)\ y$
using #(2) **by** *force*
then **obtain** $E\ F\ a'\ b'$ **where** *IH*:
 $Derivation\ \mathcal{G}\ (x\ @\ \alpha\ @\ take\ (length\ a - length\ x - 1)\ y)\ E\ a'$
 $Derivation\ \mathcal{G}\ (drop\ (length\ a - length\ x - 1)\ y)\ F\ b'$
 $c = a'\ @\ b'$
 $length\ E \leq length\ D$
 $length\ F \leq length\ D$
using *Cons.IH*[*of x @ alpha @ take (length a - length x - 1) y drop (length a - length x - 1) y*] *(2) **by** *auto*
have *Derives1* $\mathcal{G}\ a\ (fst\ d)\ (snd\ d)\ (x\ @\ \alpha\ @\ take\ (length\ a - length\ x - 1)\ y)$
unfolding *Derives1-def* **using** #(3-5) *a-def* **by** *blast*
hence *Derivation* $\mathcal{G}\ a\ ((fst\ d,\ snd\ d)\ \#\ E)\ a'$
using *IH*(1) **by** *fastforce*
moreover **have** *Derivation* $\mathcal{G}\ b\ F\ b'$
using *b-def IH*(2) **by** *blast*
ultimately *show* *?thesis*
using *IH*(3-5) **by** *fastforce*
qed
qed
lemma *Derivation-S1*:
assumes *Derivation* $\mathcal{G}\ [\mathfrak{S}\ \mathcal{G}]\ D\ \omega\ is\ word\ \mathcal{G}\ \omega$
shows $\exists\ \alpha\ E.\ Derivation\ \mathcal{G}\ \alpha\ E\ \omega \wedge (\mathfrak{S}\ \mathcal{G}, \alpha) \in set\ (\mathfrak{R}\ \mathcal{G})$
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** $Derives1 \mathcal{G} [\mathfrak{S} \mathcal{G}] (fst d) (snd d) \alpha$ *Derivation* $\mathcal{G} \alpha D \omega$
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 $\langle Derivation \mathcal{G} \alpha D \omega \rangle$ **by** *auto*
qed

end
theory *Earley*
imports
Derivations
begin

5 Slices

fun *slice* :: 'a list \Rightarrow nat \Rightarrow nat \Rightarrow '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 \Rightarrow nat \Rightarrow nat \Rightarrow 'a list $\langle _ _ _ \rangle [1000,0,0] 1000$

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

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 \leq c \Longrightarrow *slice* (x#xs) (Suc b) c = *slice* xs b (c-1)
using *Suc-le-D* **by** *fastforce*

lemma *slice-concat*:
a \leq b \Longrightarrow b \leq c \Longrightarrow *slice* xs a b @ *slice* xs b c = *slice* xs a c
proof (*induction* xs a b *arbitrary: c* *rule: slice.induct*)
case (\exists 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 \leq c \Longrightarrow *slice* xs a c = ys @ zs \Longrightarrow \exists b. ys = *slice* xs a b \wedge zs = *slice* xs b c \wedge
a \leq b \wedge b \leq c
proof (*induction* xs a c *arbitrary: ys zs* *rule: slice.induct*)

```

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

```

lemma slice-nth:

```

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

```

 $a \leq b \implies b < \text{length } xs \implies \text{slice } xs \ a \ b @ [xs!b] = \text{slice } xs \ a \ (b+1)$ 
by (metis le-add1 slice-concat slice-nth)

```

lemma slice-empty:

```

 $b \leq a \implies \text{slice } xs \ a \ b = []$ 
by (simp add: slice-drop-take)

```

lemma slice-id[simp]:

```

 $\text{slice } xs \ 0 \ (\text{length } xs) = xs$ 
by (simp add: slice-drop-take)

```

lemma slice-singleton:

```

 $b \leq \text{length } xs \implies [x] = \text{slice } xs \ a \ b \implies b = a + 1$ 
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 \Rightarrow 'a **where**
 lhs-rule \equiv fst

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

rhs-rule \equiv *snd*

datatype 'a item =

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

is-finished \mathcal{G} ω *x* \equiv
lhs-item *x* = \mathfrak{S} \mathcal{G} \wedge
start-item *x* = 0 \wedge
end-item *x* = *length* ω \wedge
is-complete *x*

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

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

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

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

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

Item r 0 0 0 \in *Earley* \mathcal{G} ω

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

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

Item r (*b* + 1) *i* (*j* + 1) \in *Earley* \mathcal{G} ω

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

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

Item r' 0 *j* *j* \in *Earley* \mathcal{G} ω

| *Complete*: $x = \text{Item } r_x$ *b*_{*x*} *i* *j* $\implies x \in$ *Earley* \mathcal{G} $\omega \implies y = \text{Item } r_y$ *b*_{*y*} *j* *k* \implies

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

6.2 Well-formedness

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

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

lemma wf-Init :

assumes $r \in \text{set } (\mathfrak{R} \ \mathcal{G}) \ \text{fst } r = \mathfrak{S} \ \mathcal{G}$
shows $\text{wf-item } \mathcal{G} \ \omega \ (\text{Item } r \ 0 \ 0 \ 0)$
using *assms* **unfolding** wf-item-def **by** *simp*

lemma wf-Scan :

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

lemma wf-Predict :

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

lemma wf-Complete :

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

lemma wf-Earley :

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

6.3 Soundness

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

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

lemma *sound-Init*:

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

proof –

let $?x = \text{Item } r$ 0 0 0
have $(\text{lhs-item } ?x, \beta\text{-item } ?x) \in \text{set } (\mathfrak{R} \mathcal{G})$
using *assms*(1) **by** (*simp add: item-defs*)
hence *derives* $\mathcal{G} [\text{lhs-item } ?x]$ ($\beta\text{-item } ?x$)
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 $x = \text{Item } r$ b i j *wf-item* $\mathcal{G} \ \omega$ x *sound-item* $\mathcal{G} \ \omega$ x
assumes $\omega!j = a$ $j < \text{length } \omega$ *next-symbol* $x = \text{Some } a$
shows *sound-item* $\mathcal{G} \ \omega$ (*Item* r $(b+1)$ i $(j+1)$)

proof –

define x' **where** [*simp*]: $x' = \text{Item } r$ $(b+1)$ i $(j+1)$
obtain $\beta\text{-item}'$ **where** $*$: $\beta\text{-item } x = a \# \beta\text{-item}'$ $\beta\text{-item } x' = \beta\text{-item}'$
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 @ $\beta\text{-item } x = \text{slice } \omega$ i $(j+1)$ @ $\beta\text{-item}'$
using $*$ *assms*(1,2,4,5) **by** (*auto simp: slice-append-nth wf-item-def*)
moreover **have** *derives* $\mathcal{G} [\text{lhs-item } x]$ (*slice* ω i j @ $\beta\text{-item } x$)
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 $x = \text{Item } r$ b i j *wf-item* $\mathcal{G} \ \omega$ x *sound-item* $\mathcal{G} \ \omega$ x
assumes $r' \in \text{set } (\mathfrak{R} \mathcal{G})$ *next-symbol* $x = \text{Some } (\text{lhs-rule } r')$
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 $x = \text{Item } r_x$ b_x i j *wf-item* $\mathcal{G} \ \omega$ x *sound-item* $\mathcal{G} \ \omega$ x
assumes $y = \text{Item } r_y$ b_y j k *wf-item* $\mathcal{G} \ \omega$ y *sound-item* $\mathcal{G} \ \omega$ y
assumes *is-complete* y *next-symbol* $x = \text{Some } (\text{lhs-item } y)$
shows *sound-item* $\mathcal{G} \ \omega$ (*Item* r_x $(b_x + 1)$ i k)

proof –

have *derives* $\mathcal{G} [\text{lhs-item } y]$ (*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* $\mathcal{G} [\text{lhs-item } y]$ E (*slice* ω j k)
using *derives-implies-Derivation* **by** *blast*
have *derives* $\mathcal{G} [\text{lhs-item } x]$ (*slice* ω i j @ $\beta\text{-item } x$)
using *assms*(1,3,4) **by** (*auto simp: sound-item-def*)

moreover have $0: \beta\text{-item } x = (\text{lhs-item } y) \# \text{tl } (\beta\text{-item } x)$
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* ($\beta\text{-item } x$)))
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* ($\beta\text{-item } x$))
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* ($\beta\text{-item } x$))
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 $x \in \text{Earley } \mathcal{G} \omega$ *wf-item* $\mathcal{G} \omega$ x
shows *sound-item* $\mathcal{G} \omega$ 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 r_y b_y k)
thus *?case*
using *wf-Earley sound-Complete* **by** *metis*
qed

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

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

lemma *partially-completed-upto*:

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

proof (*induction* $\beta\text{-item } x$ *arbitrary*: $d \ i \ j \ k \ N \ \alpha \ x \ D$)

case *Nil*

have $\alpha\text{-item } x = \alpha$

using $\text{Nil}(1,4)$ **unfolding** $\alpha\text{-item-def } \beta\text{-item-def } \text{rhs-item-def } \text{rhs-rule-def}$ **by** *simp*

hence $x = \text{Item } (N, \alpha) \ (\text{length } \alpha) \ i \ j$

using $\text{Nil.hyps } \text{Nil.prem}(3-5)$ **unfolding** *wf-item-def item-defs* **by** *auto*

have $\text{Derivation } \mathcal{G} \ [] \ D \ (\text{slice } \omega \ j \ k)$

using $\text{Nil.hyps } \text{Nil.prem}(6)$ **by** *auto*

hence $\text{slice } \omega \ j \ k = []$

using *Derivation-from-empty* **by** *blast*

hence $j = k$

unfolding *slice-drop-take* **using** $\text{Nil.prem}(1,2)$ **by** *simp*

thus *?case*

using $\langle x = \text{Item } (N, \alpha) \ (\text{length } \alpha) \ i \ j \rangle \ \text{Nil.prem}(4)$ **by** *blast*

next

case (*Cons* $b \ bs$)

obtain $j' \ E \ F$ **where** $*$:

$\text{Derivation } \mathcal{G} \ [b] \ E \ (\text{slice } \omega \ j \ j')$

$\text{Derivation } \mathcal{G} \ bs \ F \ (\text{slice } \omega \ j' \ k)$

$j \leq j' \ j' \leq k \ \text{length } E \leq \text{length } D \ \text{length } F \leq \text{length } D$

using *Derivation-concat-split*[*of* $\mathcal{G} \ [b] \ bs \ D \ \text{slice } \omega \ j \ k$] *slice-concat-Ex*

using $\text{Cons.hyps}(2) \ \text{Cons.prem}(1,6)$

by (*smt* (*verit*, *ccfv-threshold*) *Cons-eq-appendI append-self-conv2*)

have $\text{next-symbol } x = \text{Some } b$

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'* $\in I$
using *Cons.prem*s(7) **unfolding** *partially-completed-def*
using *Cons.prem*s(2,3,4) *(1,3-5) **by** *blast*
moreover have *partially-completed* *k G* ωI ($\lambda D'$. *length* *D'* \leq *length* *F*)
using *Cons.prem*s(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.prem*s(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.prem*s(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

$i \leq j \wedge j \leq k \wedge k \leq \text{length } \omega \wedge$

$x = \text{Item } r b i' i \wedge x \in \text{Earley } \mathcal{G} \omega \wedge$

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

Derivation G [*a*] *D* (*slice* ω *i j*) \wedge *True*

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

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 $j \leq \text{length } \omega$

using *le-trans* 1.*prems* **by** *blast*

ultimately have $j = i + 1$

using *slice-singleton* **by** *metis*

hence $i < \text{length } \omega$

using $\langle j \leq \text{length } \omega \rangle$ **by** *simp*

hence $\omega ! i = a$

using *slice-nth* $\langle [a] = \text{slice } \omega \ i \ j \rangle \langle j = i + 1 \rangle$ **by** *fastforce*

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

using *Earley.Scan* 1.*prems* $\langle i < \text{length } \omega \rangle \langle j = i + 1 \rangle$ **by** *metis*

thus ?*thesis*

by (*simp add:* $\langle j = i + 1 \rangle$)

next

assume $\neg D = []$

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.prem*s by *auto*
hence *rule*: $(a, \alpha) \in \text{set } (\mathfrak{R} \mathcal{G})$ *fst* $d = 0$ *snd* $d = (a, \alpha)$
using **(1)* **unfolding** *Derives1-def* by *(simp add: Cons-eq-append-conv)+*
define y **where** *y-def*: $y = \text{Item } (a, \alpha) \ 0 \ i \ i$
have $\text{length } D' < \text{length } D$
using $\langle D = d \ \# \ D' \rangle$ by *fastforce*
hence *partially-completed* $k \ \mathcal{G} \ \omega$ (*Earley* $\mathcal{G} \ \omega$) $(\lambda E. \text{length } E \leq \text{length } D')$
unfolding *partially-completed-def* **using** *1.hyps order-le-less-trans* by *(smt (verit, best))*
hence *partially-completed* $j \ \mathcal{G} \ \omega$ (*Earley* $\mathcal{G} \ \omega$) $(\lambda E. \text{length } E \leq \text{length } D')$
unfolding *partially-completed-def* **using** *1.prem*s by *force*
moreover **have** *Derivation* $\mathcal{G} \ (\beta\text{-item } y) \ D'$ (*slice* $\omega \ i \ j$)
using **(2)* by *(auto simp: item-defs y-def)*
moreover **have** $y \in \text{Earley } \mathcal{G} \ \omega$
using *y-def 1.prem*s *rule* by *(auto simp: item-defs Earley.Predict)*
moreover **have** $j \leq \text{length } \omega$
using *1.prem*s by *simp*
ultimately **have** *Item* $(a, \alpha) \ (\text{length } \alpha) \ i \ j \in \text{Earley } \mathcal{G} \ \omega$
using *partially-completed-upto 1.prem*s *wf-Earley y-def* by *metis*
moreover **have** $x = \text{Item } r \ b \ i' \ i \ x \in \text{Earley } \mathcal{G} \ \omega$
using *1.prem*s by *blast+*
moreover **have** *next-symbol* $x = \text{Some } a$
using *1.prem*s 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 $\mathcal{G} \vdash [\mathfrak{S} \ \mathcal{G}] \Rightarrow^* \omega \text{ is-word } \mathcal{G} \ \omega$
shows *recognizing* $(\text{Earley } \mathcal{G} \ \omega) \ \mathcal{G} \ \omega$
proof –
obtain $\alpha \ D$ **where** $*$: $(\mathfrak{S} \ \mathcal{G}, \alpha) \in \text{set } (\mathfrak{R} \ \mathcal{G})$ *Derivation* $\mathcal{G} \ \alpha \ D \ \omega$
using *Derivation- $\mathfrak{S}1$ assms derives-implies-Derivation* by *metis*
define x **where** *x-def*: $x = \text{Item } (\mathfrak{S} \ \mathcal{G}, \alpha) \ 0 \ 0 \ 0$
have *partially-completed* $(\text{length } \omega) \ \mathcal{G} \ \omega$ (*Earley* $\mathcal{G} \ \omega$) $(\lambda -. \text{True})$
using *assms(2) partially-completed-Earley* by *blast*
hence 0 : *partially-completed* $(\text{length } \omega) \ \mathcal{G} \ \omega$ (*Earley* $\mathcal{G} \ \omega$) $(\lambda D'. \text{length } D' \leq \text{length } D)$
unfolding *partially-completed-def* by *blast*
have 1 : $x \in \text{Earley } \mathcal{G} \ \omega$
using *x-def Earley.Init *(1)* by *fastforce*
have 2 : *Derivation* $\mathcal{G} \ (\beta\text{-item } x) \ D$ (*slice* $\omega \ 0 \ (\text{length } \omega)$)
using **(2) x-def* by *(simp add: item-defs)*
have *Item* $(\mathfrak{S} \ \mathcal{G}, \alpha) \ (\text{length } \alpha) \ 0 \ (\text{length } \omega) \in \text{Earley } \mathcal{G} \ \omega$
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* $\mathcal{G} \ \omega$
shows *recognizing* (*Earley* $\mathcal{G} \ \omega$) $\mathcal{G} \ \omega \longleftrightarrow \mathcal{G} \vdash [\mathfrak{E} \ \mathcal{G}] \Rightarrow^* \omega$
using *assms soundness-Earley completeness-Earley* **by** *blast*

6.6 Finiteness

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

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

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

proof –

define *M* **where** $M = Max \ \{ length \ (rhs\text{-rule} \ r) \mid r. r \in set \ (\mathfrak{R} \ \mathcal{G}) \}$
define *Top* **where** $Top = (set \ (\mathfrak{R} \ \mathcal{G}) \times \{0..M\} \times \{0..length \ \omega\} \times \{0..length \ \omega\})$

hence *finite Top*
using *finiteness-product* **by** *blast*

have *inj-on item-intro Top*
unfolding *Top-def inj-on-def* **by** *simp*

hence *finite (item-intro ' Top)*
using *finiteness-image-iff* *'finite Top* **by** *auto*

have $\{ x \mid x. wf\text{-item} \ \mathcal{G} \ \omega \ x \} \subseteq item\text{-intro} \ ' \ Top$

proof *standard*

fix *x*

assume $x \in \{ x \mid x. wf\text{-item} \ \mathcal{G} \ \omega \ x \}$

then obtain *rule dot origin endp* **where** $*$: $x = Item \ rule \ dot \ origin \ endp$
 $rule \in set \ (\mathfrak{R} \ \mathcal{G}) \ dot \leq length \ (rhs\text{-item} \ x) \ origin \leq length \ \omega \ endp \leq length \ \omega$

unfolding *wf-item-def* **using** *item.exhaust-sel le-trans* **by** *blast*

hence $length \ (rhs\text{-rule} \ rule) \in \{ length \ (rhs\text{-rule} \ r) \mid r. r \in set \ (\mathfrak{R} \ \mathcal{G}) \}$

using $*(1,2)$ *rhs-item-def* **by** *blast*

moreover have $finite \ \{ length \ (rhs\text{-rule} \ r) \mid r. r \in set \ (\mathfrak{R} \ \mathcal{G}) \}$

using *finiteness-image-set*[*of* $\lambda x. x \in set \ (\mathfrak{R} \ \mathcal{G})$] **by** *fastforce*

ultimately have $M \geq length \ (rhs\text{-rule} \ rule)$

unfolding *M-def* **by** *simp*

hence $dot \leq M$

using $*(1,3)$ *rhs-item-def* **by** (*metis item.sel(1) le-trans*)

hence (*rule, dot, origin, endp*) $\in Top$

using $*(2,4,5)$ *Top-def* **by** *simp*

```

    thus  $x \in \text{item-intro } \text{'Top}$ 
      using  $*(1)$  by force
  qed
  thus  $?thesis$ 
    using  $\langle \text{finite } (\text{item-intro } \text{'Top}) \rangle \text{ rev-finite-subset}$  by auto
  qed

```

```

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

```

```

theorem finiteness-Earley:
  finite (Earley  $\mathcal{G} \ \omega$ )
  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 ::  $'a \text{ rule} \Rightarrow \text{nat} \Rightarrow 'a \text{ item}$  where
  init-item  $r \ k \equiv \text{Item } r \ 0 \ k \ k$ 

```

```

definition inc-item ::  $'a \text{ item} \Rightarrow \text{nat} \Rightarrow 'a \text{ item}$  where
  inc-item  $x \ k \equiv \text{Item } (\text{rule-item } x) \ (\text{dot-item } x + 1) \ (\text{start-item } x) \ k$ 

```

```

definition bin ::  $'a \text{ item set} \Rightarrow \text{nat} \Rightarrow 'a \text{ item set}$  where
  bin  $I \ k \equiv \{ x . x \in I \wedge \text{end-item } x = k \}$ 

```

```

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

```

```

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

```

```

definition InitF ::  $'a \text{ cfg} \Rightarrow 'a \text{ item set}$  where
  InitF  $\mathcal{G} \equiv \{ \text{init-item } r \ 0 \mid r. r \in \text{set } (\mathfrak{R} \ \mathcal{G}) \wedge \text{fst } r = (\mathfrak{S} \ \mathcal{G}) \}$ 

```

```

definition ScanF ::  $\text{nat} \Rightarrow 'a \text{ list} \Rightarrow 'a \text{ item set} \Rightarrow 'a \text{ item set}$  where
  ScanF  $k \ \omega \ I \equiv \{ \text{inc-item } x \ (k+1) \mid x \ a. \}$ 

```

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

definition $\text{Predict}_F :: \text{nat} \Rightarrow 'a \text{ cfg} \Rightarrow 'a \text{ item set} \Rightarrow 'a \text{ item set}$ **where**

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

definition $\text{Complete}_F :: \text{nat} \Rightarrow 'a \text{ item set} \Rightarrow 'a \text{ item set}$ **where**

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

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

$\text{Earley}_F\text{-bin-step} \ k \ \mathcal{G} \ \omega \ I = I \cup \text{Scan}_F \ k \ \omega \ I \cup \text{Complete}_F \ k \ I \cup \text{Predict}_F \ k \ \mathcal{G} \ I$

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

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

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

$\text{Earley}_F\text{-bins} \ 0 \ \mathcal{G} \ \omega = \text{Earley}_F\text{-bin} \ 0 \ \mathcal{G} \ \omega \ (\text{Init}_F \ \mathcal{G})$
 $\mid \text{Earley}_F\text{-bins} \ (\text{Suc } n) \ \mathcal{G} \ \omega = \text{Earley}_F\text{-bin} \ (\text{Suc } n) \ \mathcal{G} \ \omega \ (\text{Earley}_F\text{-bins } n \ \mathcal{G} \ \omega)$

definition $\text{Earley}_F :: 'a \text{ cfg} \Rightarrow 'a \text{ list} \Rightarrow 'a \text{ item set}$ **where**

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

7.2 Monotonicity and Absorption

lemma $\text{Earley}_F\text{-bin-step-empty}$:

$\text{Earley}_F\text{-bin-step} \ k \ \mathcal{G} \ \omega \ \{\} = \{\}$

unfolding $\text{Earley}_F\text{-bin-step-def}$ $\text{Scan}_F\text{-def}$ $\text{Complete}_F\text{-def}$ $\text{Predict}_F\text{-def}$ bin-def **by** *blast*

lemma $\text{Earley}_F\text{-bin-step-setmonotone}$:

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

by (*simp add: Un-assoc Earley_F-bin-step-def setmonotone-def*)

lemma $\text{Earley}_F\text{-bin-step-continuous}$:

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

unfolding *continuous-def*

proof (*standard, standard, standard*)

fix $C :: \text{nat} \Rightarrow 'a \text{ item set}$

```

assume chain C
thus chain (EarleyF-bin-step k  $\mathcal{G}$   $\omega \circ C$ )
  unfolding chain-def EarleyF-bin-step-def by (auto simp: ScanF-def PredictF-def
  CompleteF-def bin-def subset-eq)
next
  fix C :: nat  $\Rightarrow$  'a item set
  assume *: chain C
  show EarleyF-bin-step k  $\mathcal{G}$   $\omega$  (natUnion C) = natUnion (EarleyF-bin-step k  $\mathcal{G}$ 
   $\omega \circ C$ )
    unfolding natUnion-def
    proof standard
      show EarleyF-bin-step k  $\mathcal{G}$   $\omega$  ( $\bigcup \{C\ n \mid n. \text{True}\}$ )  $\subseteq$   $\bigcup \{(EarleyF-bin-step\ k\ \mathcal{G}\ \omega \circ C)\ n \mid n. \text{True}\}$ 
      proof standard
        fix x
        assume #:  $x \in EarleyF-bin-step\ k\ \mathcal{G}\ \omega\ (\bigcup \{C\ n \mid n. \text{True}\})$ 
        show  $x \in \bigcup \{(EarleyF-bin-step\ k\ \mathcal{G}\ \omega \circ C)\ n \mid n. \text{True}\}$ 
        proof (cases  $x \in Complete_F\ k\ (\bigcup \{C\ n \mid n. \text{True}\})$ )
          case True
          then show ?thesis
            using * unfolding chain-def EarleyF-bin-step-def CompleteF-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  $\exists I.$ 
              ( $\exists n. I = C\ n \cup$ 
                ScanF (end-item z)  $\omega$  (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))})  $\cup$ 
                PredictF (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: EarleyF-bin-step-def ScanF-def PredictF-def
bin-def; blast)
  qed
  qed
  next
    show  $\bigcup \{(Earley_F\text{-bin-step } k \mathcal{G} \omega \circ C) \ n \mid n. \text{ True}\} \subseteq Earley_F\text{-bin-step } k \mathcal{G} \omega$ 
    ( $\bigcup \{C \ n \mid n. \text{ True}\}$ )
    unfolding EarleyF-bin-step-def
    using * by (auto simp: ScanF-def PredictF-def CompleteF-def chain-def
bin-def, metis+)
  qed
  qed

```

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

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

lemma *Scan_F-bin-absorb:*
 $Scan_F \ k \ \omega (bin \ I \ k) = Scan_F \ k \ \omega \ I$
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:*
 $Scan_F \ k \ \omega (I \cup J) = Scan_F \ k \ \omega \ I \cup Scan_F \ k \ \omega \ J$
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:*
 $I \subseteq J \implies Scan_F \ k \ \omega \ I \subseteq Scan_F \ k \ \omega \ J$
unfolding *Scan_F-def bin-def by blast*

lemma *Predict_F-sub-mono:*
 $I \subseteq J \implies Predict_F \ k \ \mathcal{G} \ I \subseteq Predict_F \ k \ \mathcal{G} \ J$

unfolding *Predict_F-def bin-def* **by** *blast*

lemma *Complete_F-sub-mono*:

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

unfolding *Complete_F-def bin-def* **by** *blast*

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

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

unfolding *Earley_F-bin-step-def* **using** *Scan_F-sub-mono Predict_F-sub-mono Complete_F-sub-mono* **by** (*metis sup.mono*)

lemma *funpower-sub-mono*:

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

by (*induction n*) (*auto simp: Earley_F-bin-step-sub-mono*)

lemma *Earley_F-bin-sub-mono*:

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

proof *standard*

fix x

assume $I \subseteq J$ $x \in \text{Earley}_F\text{-bin } k \mathcal{G} \omega I$

then obtain n **where** $x \in \text{funpower } (\text{Earley}_F\text{-bin-step } k \mathcal{G} \omega) n I$

unfolding *Earley_F-bin-def limit-def natUnion-def* **by** *blast*

hence $x \in \text{funpower } (\text{Earley}_F\text{-bin-step } k \mathcal{G} \omega) n J$

using $\langle I \subseteq J \rangle$ *funpower-sub-mono* **by** *blast*

thus $x \in \text{Earley}_F\text{-bin } k \mathcal{G} \omega J$

unfolding *Earley_F-bin-def limit-def natUnion-def* **by** *blast*

qed

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

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

using *Earley_F-bin-step-def* **by** *blast*

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

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

using *Earley_F-bin-step-def* **by** *blast*

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

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

using *Earley_F-bin-step-def* **by** *blast*

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

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

proof –

have $\text{Earley}_F\text{-bin-step } k \mathcal{G} \omega I \subseteq \text{funpower } (\text{Earley}_F\text{-bin-step } k \mathcal{G} \omega) 1 I$

by *simp*

thus *?thesis*

by (*metis Earley_F-bin-def limit-lem 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 $I \subseteq \text{Earley } \mathcal{G} \ \omega$
shows *Earley_F-bin-step* $k \ \mathcal{G} \ \omega \ I \subseteq \text{Earley } \mathcal{G} \ \omega$
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 $I \subseteq \text{Earley } \mathcal{G} \ \omega$
shows *Earley_F-bin* $k \ \mathcal{G} \ \omega \ I \subseteq \text{Earley } \mathcal{G} \ \omega$
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* $n \ \mathcal{G} \ \omega \subseteq \text{Earley } \mathcal{G} \ \omega$
by (*induction n*) (*auto simp: Earley_F-bin-sub-Earley Init_F-sub-Earley*)

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

theorem *soundness-Earley_F*:
assumes *recognizing* (*Earley_F* $\mathcal{G} \ \omega$) $\mathcal{G} \ \omega$
shows $\mathcal{G} \vdash [\mathcal{G}] \Rightarrow^* \omega$
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 $\forall k' < k. \text{bin} (\text{Earley } \mathcal{G} \ \omega) \ k' \subseteq I$
assumes *base* $\omega (\text{Earley } \mathcal{G} \ \omega) \ k \subseteq I$
shows *bin* (*Earley* $\mathcal{G} \ \omega$) $k \subseteq \text{bin} (\text{Earley_F-bin } k \ \mathcal{G} \ \omega \ I) \ k$

proof *standard*

fix x

assume $*$: $x \in \text{bin} (\text{Earley } \mathcal{G} \ \omega) \ k$

hence $x \in \text{Earley } \mathcal{G} \ \omega$

using *bin-def* **by** *blast*

thus $x \in \text{bin} (\text{Earley_F-bin } k \ \mathcal{G} \ \omega \ I) \ k$

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

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

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

by (metis (mono-tags, lifting) CollectI item.sel(4) subsetD)
 hence $\text{Item } r \ (b+1) \ i \ (j+1) \in \text{Scan}_F \ k \ \omega \ (\text{Earley}_F\text{-bin } k \ \mathcal{G} \ \omega \ I)$
 unfolding Scan_F-def inc-item-def using Scan.hyps ⟨j = k⟩ by force
 hence $\text{Item } r \ (b+1) \ i \ (j+1) \in \text{Earley}_F\text{-bin-step } k \ \mathcal{G} \ \omega \ (\text{Earley}_F\text{-bin } k \ \mathcal{G} \ \omega \ I)$
 using Scan_F-Earley_F-bin-step-mono by blast
 hence $\text{Item } r \ (b+1) \ i \ (j+1) \in \text{Earley}_F\text{-bin } k \ \mathcal{G} \ \omega \ I$
 using Earley_F-bin-idem Earley_F-bin-step-Earley_F-bin-mono by blast
 thus ?case
 using ⟨j = k⟩ bin-def by fastforce
 next
 case (Predict x r b i j r')
 have False
 using Predict.premis(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 $l-1 < \text{length } \omega$
 using Complete.premis(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 $\omega!(l-1) \notin \text{nonterminals } \mathcal{G}$
 using Complete.premis(4) is-word-def by force
 moreover have $\text{lhs-item } y \in \text{nonterminals } \mathcal{G}$
 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 $\text{prev-symbol } (\text{Item } r_x \ (b_x+1) \ i \ l) = \text{next-symbol } x$
 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 $\text{prev-symbol } (\text{Item } r_x \ (b_x+1) \ i \ l) = \text{Some } (\omega!(l-1))$
 using Complete.premis(5) base-def by (metis (mono-tags, lifting) CollectD
 item.sel(4))
 ultimately have False
 using Complete.hyps(6) Complete.premis(4) by simp
 thus ?case
 by blast
 qed
 qed
 lemma Earley_F-bin-k-sub-Earley_F-bins:
 assumes is-word $\mathcal{G} \ \omega \ k \leq n$
 shows $\text{bin } (\text{Earley } \mathcal{G} \ \omega) \ k \subseteq \text{Earley}_F\text{-bins } n \ \mathcal{G} \ \omega$
 using assms
 proof (induction n arbitrary: k)
 case 0
 have $\text{bin } (\text{Earley } \mathcal{G} \ \omega) \ 0 \subseteq \text{bin } (\text{Earley}_F\text{-bin } 0 \ \mathcal{G} \ \omega \ (\text{Init}_F \ \mathcal{G})) \ 0$
 using Earley_F-bin-sub-Earley_F-bin base-def by fastforce

```

thus ?case
  unfolding bin-def using 0.prem(2) by auto
next
case (Suc n)
show ?case
proof (cases k ≤ n)
  case True
  thus ?thesis
  using Suc EarleyF-bin-mono by force
next
case False
hence k = n+1
  using Suc.prem(2) by force
have 0: ∀ k' < k. bin (Earley  $\mathcal{G}$   $\omega$ ) k' ⊆ EarleyF-bins n  $\mathcal{G}$   $\omega$ 
  using Suc by simp
moreover have base  $\omega$  (Earley  $\mathcal{G}$   $\omega$ ) k ⊆ EarleyF-bins n  $\mathcal{G}$   $\omega$ 
proof -
  have ∀ k' < k-1. bin (Earley  $\mathcal{G}$   $\omega$ ) k' ⊆ EarleyF-bins n  $\mathcal{G}$   $\omega$ 
  using Suc <k = n + 1> by auto
  moreover have base  $\omega$  (Earley  $\mathcal{G}$   $\omega$ ) (k-1) ⊆ EarleyF-bins n  $\mathcal{G}$   $\omega$ 
  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  $\omega$  (Earley  $\mathcal{G}$   $\omega$ ) k ⊆ bin (EarleyF-bin n  $\mathcal{G}$   $\omega$  (EarleyF-bins
n  $\mathcal{G}$   $\omega$ )) k
  using Suc.prem(1,2) Earley-base-sub-EarleyF-bin <k = n + 1> InitF-sub-EarleyF-bins
by (metis add-diff-cancel-right')
  hence base  $\omega$  (Earley  $\mathcal{G}$   $\omega$ ) k ⊆ bin (EarleyF-bins n  $\mathcal{G}$   $\omega$ ) k
  by (metis EarleyF-bins.elims EarleyF-bin-idem)
  thus ?thesis
  using bin-def by blast
qed
ultimately have bin (Earley  $\mathcal{G}$   $\omega$ ) k ⊆ bin (EarleyF-bin k  $\mathcal{G}$   $\omega$  (EarleyF-bins
n  $\mathcal{G}$   $\omega$ )) k
  using EarleyF-bin-sub-EarleyF-bin InitF-sub-EarleyF-bins by metis
  thus ?thesis
  using EarleyF-bins.simps(2) <k = n + 1> bin-def by auto
qed
qed

lemma Earley-sub-EarleyF:
  assumes is-word  $\mathcal{G}$   $\omega$ 
  shows Earley  $\mathcal{G}$   $\omega$  ⊆ EarleyF  $\mathcal{G}$   $\omega$ 
proof -
  have ∀ k ≤ length  $\omega$ . bin (Earley  $\mathcal{G}$   $\omega$ ) k ⊆ EarleyF  $\mathcal{G}$   $\omega$ 
  by (simp add: EarleyF-bin-k-sub-EarleyF-bins EarleyF-def assms)
  thus ?thesis
  using wf-Earley wf-item-def bin-def by blast
qed

```

theorem *completeness-Earley_F*:
assumes $\mathcal{G} \vdash [\mathfrak{G} \mathcal{G}] \Rightarrow^* \omega$ *is-word* $\mathcal{G} \ \omega$
shows *recognizing* (*Earley_F* $\mathcal{G} \ \omega$) $\mathcal{G} \ \omega$
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* $\mathcal{G} \ \omega$
shows *Earley* $\mathcal{G} \ \omega =$ *Earley_F* $\mathcal{G} \ \omega$
using *Earley-sub-Earley_F* *Earley_F-sub-Earley* *assms* **by** *blast*

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

end

theory *Earley-Recognizer*

imports

Earley-Fixpoint

begin

8 Earley recognizer

8.1 List auxiliaries

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

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

lemma *filter-with-index'-P*:
 $(x, n) \in \text{set} \ (\text{filter-with-index}' \ i \ P \ xs) \Longrightarrow P \ x$
by (*induction xs arbitrary: i*) (*auto split: if-splits*)

lemma *filter-with-index-P*:
 $(x, n) \in \text{set} \ (\text{filter-with-index} \ P \ xs) \Longrightarrow P \ x$
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'*:

$(x, n) \in \text{set } (\text{filter-with-index}' i P xs) \implies n \geq i$

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

lemma *index-filter-with-index'-lt-length*:

$(x, n) \in \text{set } (\text{filter-with-index}' i P xs) \implies n - i < \text{length } xs$

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*:

$(x, n) \in \text{set } (\text{filter-with-index } P xs) \implies n < \text{length } xs$

by (*metis filter-with-index-def index-filter-with-index'-lt-length minus-nat.diff-0*)

lemma *filter-with-index'-nth*:

$(x, n) \in \text{set } (\text{filter-with-index}' i P xs) \implies xs ! (n - i) = x$

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 $(x, n) \in \text{set } (\text{filter-with-index}' (i+1) P xs)$

using *Cons.prem*s **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*:

$(x, n) \in \text{set } (\text{filter-with-index } P xs) \implies xs ! n = x$

by (*metis diff-zero filter-with-index'-nth filter-with-index-def*)

lemma *filter-with-index-nonempty*:

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

by (*metis filter-empty-conv filter-with-index-cong-filter list.map(1)*)

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

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

by (*induction xs arbitrary: n*) *auto*

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

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

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

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

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

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

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

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

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

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

wf-bin G ω k b ≡ *distinct (items b) ∧ wf-bin-items G ω k (items b)*

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

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

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

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

definition *nonempty-derives* :: *'a cfg* ⇒ *bool* **where**

nonempty-derives G ≡ $\forall s. \neg G \vdash [s] \Rightarrow^* []$

definition *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 $upd\text{-}bin :: 'a\ item \times pointer \Rightarrow 'a\ bin \Rightarrow 'a\ bin\ where$

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

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 => (
      if a ∉ nonterminals G then
        if k < length ω then upd-bins bs (k+1) (Scan_L k ω a x i)
      else bs
    else upd-bins bs k (Predict_L k G a))
  | None => upd-bins bs k (Complete_L k x bs i)
  in Earley_L-bin' k G ω bs' (i+1))

```

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

definition *Earley_L-bin* :: nat => 'a cfg => 'a list => 'a bins => 'a bins **where**
Earley_L-bin k G ω bs = *Earley_L-bin'* k G ω bs 0

fun *Earley_L-bins* :: nat => 'a cfg => 'a list => 'a bins **where**
Earley_L-bins 0 G ω = *Earley_L-bin* 0 G ω (*Init_L* G ω)
| *Earley_L-bins* (Suc n) G ω = *Earley_L-bin* (Suc n) G ω (*Earley_L-bins* n G ω)

definition *Earley_L* :: 'a cfg => 'a list => 'a bins **where**
Earley_L G ω ≡ *Earley_L-bins* (length ω) G ω

definition *recognizer* :: 'a cfg => 'a list => bool **where**
recognizer G ω = (∃ x ∈ set (items (Earley_L G ω ! length ω)). *is-finished* G ω x)

8.3 Epsilon productions

lemma *ε-free-impl-non-empty-word-deriv*:

ε-free G ⇒ a ≠ [] ⇒ ¬ *Derivation* G a D []

proof (*induction* length D arbitrary: a D rule: nat-less-induct)

case 1

show ?case

proof (*rule* ccontr)

assume *assm*: ¬ ¬ *Derivation* G a D []

show False

proof (*cases* D = [])

case True

then show ?thesis

using 1.prem(2) *assm* by auto

next

case False

then obtain d D' α **where** *:

D = d # D' *Derives1* G a (fst d) (snd d) α *Derivation* G α D' [] snd d ∈ set (ℳ 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  $\varepsilon$ -free-def rhs-rule-def 1.premis(1) by (metis
append-is-Nil-conv)
  next
    assume  $\neg \alpha = []$ 
    thus ?thesis
      using *(1,3) 1.hyps 1.premis(1) by auto
  qed
qed
qed
qed

```

lemma *ε -free-impl-nonempty-derives:*
 ε -free $\mathcal{G} \implies$ 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 $\neg \varepsilon$ -free \mathcal{G}
then obtain $N \alpha$ **where** *: $(N, \alpha) \in \text{set } (\mathfrak{R} \mathcal{G})$ 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 $\mathcal{G} \iff \varepsilon$ -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]:*
 $\text{length } (\text{upd-bins } bs \ k \ es) = \text{length } bs$
unfolding upd-bins-def **by** simp

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

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

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*:
 $k \neq n \implies upd\ bins\ bs\ k\ es\ !\ n = bs\ !\ n$
unfolding *upd-bins-def* **by** *simp*

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

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

lemma *items-nth-idem-upd-bins*:
 $n < length\ (bs\ !\ k) \implies items\ (upd\ bins\ bs\ k\ es\ !\ k)\ !\ n = items\ (bs\ !\ k)\ !\ n$
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*:
 $i \geq length\ b \implies bin\ upto\ b\ i = set\ (items\ b)$
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 *:  $\neg$  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.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)  $\cup$  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  $\cup$  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  $\dots = \bigcup$  {set (items (bs ! l)) | l. l < length bs  $\wedge$  l  $\neq$  k}  $\cup$  set (items
    (?bs ! k))
    unfolding upd-bins-def using assms by (auto, metis nth-list-update)
  also have  $\dots = \bigcup$  {set (items (bs ! l)) | l. l < length bs  $\wedge$  l  $\neq$  k}  $\cup$  set (items
    (bs ! k))  $\cup$  set (items es)
    using set-items-upds-bin[of es bs!k] by (simp add: assms upd-bins-def sup-assoc)
  also have  $\dots = \bigcup$  {set (items (bs ! k)) | k. k < length bs}  $\cup$  set (items es)
    using assms by blast
  also have  $\dots = bins bs \cup set (items es)$ 
    unfolding bins-def by blast
  finally show ?thesis .
qed

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

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

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

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

lemma *distinct-upd-bin*:
 $\text{distinct } (\text{items } b) \implies \text{distinct } (\text{items } (\text{upd-bin } e \ b))$
proof (*induction b arbitrary: e*)
case (*Cons b bs*)
show *?case*
proof (*cases* $\exists x \ xp \ xs \ y \ yp \ ys. e = (x, \text{PreRed } xp \ xs) \wedge b = (y, \text{PreRed } yp \ ys)$)
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  $upd-bin e (b \# bs) = b \# bs$ 
using False by (auto split: pointer.splits prod.split)
thus ?thesis
using * Cons.premis 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)
moreover have  $distinct (items (upd-bin e bs))$ 
using Cons by (auto simp: items-def)
ultimately show ?thesis
using * Cons.premis 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*:
 $distinct (items b) \implies distinct (items (upds-bin es b))$
by (*induction es arbitrary: b (auto simp add: distinct-upd-bin)*)

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

lemma *wf-bin-upd-bin*:
assumes $wf-bin \mathcal{G} \omega k b wf-item \mathcal{G} \omega (fst e) \wedge end-item (fst e) = k$
shows $wf-bin \mathcal{G} \omega k (upd-bin e b)$
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  $e = (x, \text{PreRed } xp \ xs)$   $b = (y, \text{PreRed } yp \ ys)$ 
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  $*$ :  $\text{fst } e = \text{fst } b$ 
hence  $\text{upd-bin } e \ (b \# \text{bs}) = b \# \text{bs}$ 
using False by (auto split: pointer.splits prod.split)
thus ?thesis
using  $*$  Cons.prems by (auto simp: items-def)
next
assume  $*$ :  $\neg \text{fst } e = \text{fst } b$ 
hence  $\text{upd-bin } e \ (b \# \text{bs}) = b \# \text{upd-bin } e \ \text{bs}$ 
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  $\text{wf-bin } \mathcal{G} \ \omega \ k \ b$ 
assumes  $\forall x \in \text{set } (\text{items } es). \ \text{wf-item } \mathcal{G} \ \omega \ x \wedge \text{end-item } x = k$ 
shows  $\text{wf-bin } \mathcal{G} \ \omega \ k \ (\text{upds-bin } es \ b)$ 
using assms by (induction es arbitrary: b) (auto simp: wf-bin-upd-bin items-def)

```

```

lemma wf-bins-upd-bins:
assumes  $\text{wf-bins } \mathcal{G} \ \omega \ bs$ 
assumes  $\forall x \in \text{set } (\text{items } es). \ \text{wf-item } \mathcal{G} \ \omega \ x \wedge \text{end-item } x = k$ 
shows  $\text{wf-bins } \mathcal{G} \ \omega \ (\text{upd-bins } bs \ k \ es)$ 
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:
 $\text{wf-bins } \mathcal{G} \ \omega \ bs \implies \forall x \in (\text{bins } bs). \ \text{wf-item } \mathcal{G} \ \omega \ x$ 
unfolding wf-bins-def wf-bin-def wf-bin-items-def bins-def by auto

```

```

lemma upds-bin-eq-items:
 $\text{set } (\text{items } es) \subseteq \text{set } (\text{items } b) \implies \text{set } (\text{items } (\text{upds-bin } es \ b)) = \text{set } (\text{items } b)$ 
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*:
 $\text{fst } e \in \text{set } (\text{items } b) \implies \text{items } (\text{upd-bin } e \ b) = \text{items } b$
proof (*induction b arbitrary: e*)
 case (*Cons b bs*)
 show *?case*
 proof (*cases $\exists x \ xp \ xs \ y \ yp \ ys. e = (x, \text{PreRed } xp \ xs) \wedge b = (y, \text{PreRed } yp \ ys)$*)
 case *True*
 then obtain $x \ xp \ xs \ y \ yp \ ys$ **where** $e = (x, \text{PreRed } xp \ xs) \ b = (y, \text{PreRed } yp \ ys)$
 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 $*$: $\text{fst } e = \text{fst } b$
 hence $\text{upd-bin } e \ (b \# \text{bs}) = b \# \text{bs}$
 using *False* **by** (*auto split: pointer.splits prod.split*)
 thus *?thesis*
 using $*$ *Cons.prem*s **by** (*auto simp: items-def*)
 next
 assume $*$: $\neg \text{fst } e = \text{fst } b$
 hence $\text{upd-bin } e \ (b \# \text{bs}) = b \# \text{upd-bin } e \ \text{bs}$
 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 $\text{set } (\text{items } es) \subseteq \text{set } (\text{items } b)$
shows $\text{items } (\text{upds-bin } es \ b) = \text{items } b$
using *assms*
proof (*induction es arbitrary: b*)
 case (*Cons e es*)
 have $\text{items } (\text{upds-bin } es \ (\text{upd-bin } e \ b)) = \text{items } (\text{upd-bin } e \ b)$
 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 $\text{items } (\text{upd-bin } e \ b) = \text{items } b$
 by (*metis Cons.prem*s *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 $bins\text{-}eq\text{-}items\ (upd\text{-}bins\ bs\ k\ es)\ bs$
unfolding $upd\text{-}bins\text{-}def$ **using** $assms\ bin\text{-}eq\text{-}items\text{-}upds\text{-}bin\ bin\text{-}eq\text{-}items\text{-}def$
by $(metis\ list\text{-}update\text{-}id\ map\text{-}update)$

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

lemma $bin\text{-}eq\text{-}items\text{-}dist\text{-}upd\text{-}bin\text{-}bin$:
assumes $items\ a = items\ b$
shows $items\ (upd\text{-}bin\ e\ a) = items\ (upd\text{-}bin\ e\ b)$
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 $Cons.prem\ by\ (auto\ simp:\ items\text{-}def)$
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 $items\ (upd\text{-}bin\ e\ (a\ \# as)) = x\ \# items\ as$
using $\#$ **by** $(auto\ simp:\ items\text{-}def)$
moreover have $items\ (upd\text{-}bin\ e\ (b'\ \# bs)) = x\ \# items\ bs$
using $bs\ \# *$ **by** $(auto\ simp:\ items\text{-}def\ split:\ pointer.splits\ prod.splits)$
ultimately show $?thesis$
using bs **by** $simp$
next
assume $*:\ \neg x = y$
hence $items\ (upd\text{-}bin\ e\ (a\ \# as)) = y\ \# items\ (upd\text{-}bin\ e\ as)$
using $\#$ **by** $(auto\ simp:\ items\text{-}def)$
moreover have $items\ (upd\text{-}bin\ e\ (b'\ \# bs)) = y\ \# items\ (upd\text{-}bin\ e\ bs)$
using $bs\ \# *$ **by** $(auto\ simp:\ items\text{-}def\ split:\ pointer.splits\ prod.splits)$
ultimately show $?thesis$
using $bs\ Cons.IH$ **by** $simp$
qed
next
case $False$
then show $?thesis$
proof $cases$
assume $*:\ fst\ e = fst\ a$
hence $items\ (upd\text{-}bin\ e\ (a\ \# as)) = fst\ a\ \# items\ as$
using $False$ **by** $(auto\ simp:\ items\text{-}def\ split:\ pointer.splits\ prod.splits)$

```

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.prem by (auto simp: items-def split: pointer.splits prod.splits)
  next
    assume *: ¬ x = y
    thus ?thesis
    using # Cons.prem

```

```

      by (auto simp: items-def split!: pointer.splits prod.splits, metis Cons.IH
Cons.prem items-def)+
    qed
  next
    case False
    then show ?thesis
    proof cases
      assume *: fst e = fst a
      thus ?thesis
      using Cons.prem by (auto simp: items-def split!: pointer.splits prod.splits)
    next
      assume *: ¬ fst e = fst a
      thus ?thesis
      using Cons.prem
      by (auto simp: items-def split!: pointer.splits prod.splits, metis Cons.IH
Cons.prem items-def)+
    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* $k < \text{length } \text{bs}$ $x \in \text{set } (\text{items } (\text{bs } ! \ k))$
assumes $k < \text{length } \omega$ *next-symbol* $x \neq \text{None}$ $y = \text{inc-item } x \ (k+1)$
shows *wf-item* \mathcal{G} ω $y \wedge \text{end-item } y = k+1$
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* $k < \text{length } \text{bs}$ $x \in \text{set } (\text{items } (\text{bs } ! \ k))$ $k < \text{length } \omega$
next-symbol $x \neq \text{None}$
shows $\forall y \in \text{set } (\text{items } (\text{Scan}_L \ k \ \omega \ a \ x \ \text{pre}))$. *wf-item* \mathcal{G} ω $y \wedge \text{end-item } y = (k+1)$
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* $k < \text{length } \text{bs}$ $k \leq \text{length } \omega$
shows $\forall y \in \text{set } (\text{items } (\text{Predict}_L \ k \ \mathcal{G} \ X))$. *wf-item* \mathcal{G} ω $y \wedge \text{end-item } y = k$
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* $x = \text{Some } a$ *start-item* $x \leq k$ $k \leq \text{length } \omega$
shows *wf-item* \mathcal{G} ω (*inc-item* $x \ k$) $\wedge \text{end-item } (\text{inc-item } x \ k) = k$
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* $k < \text{length } \text{bs}$ $y \in \text{set } (\text{items } (\text{bs } ! \ k))$
shows $\forall x \in \text{set } (\text{items } (\text{Complete}_L \ k \ y \ \text{bs} \ \text{red}))$. *wf-item* \mathcal{G} ω $x \wedge \text{end-item } x = k$

proof –

let *?orig* = *bs* ! (*start-item* y)
let *?is* = *filter-with-index* (λx . *next-symbol* $x = \text{Some } (\text{lhs-item } y)$) (*items* *?orig*)
let *?is'* = *map* ($\lambda(x, \text{pre})$. (*inc-item* $x \ k$, *PreRed* (*start-item* y , *pre*, *red*) [])) *?is*
{
 fix x
 assume $*$: $x \in \text{set } (\text{map } \text{fst } ?is)$
 have *end-item* $x = \text{start-item } y$
 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* $x = \text{Some } (\text{lhs-item } y)$
 using $*$ **by** (*simp add: filter-set filter-with-index-cong-filter*)
 moreover **have** *start-item* $x \leq k$
 using $\langle \text{end-item } x = \text{start-item } y \rangle \langle \text{wf-item } \mathcal{G} \ \omega \ x \rangle$ *assms wf-bins-kth-bin*

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

lemma *Ex-wf-bins*:
 $\exists n \ bs \ \omega \ \mathcal{G}. \ n \leq \text{length } \omega \wedge \ \text{length } bs = \text{Suc} \ (\text{length } \omega) \wedge \ \text{wf-bins } \mathcal{G} \ \omega \ bs$
 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* :: $(\text{nat} \times 'a \ \text{cfg} \times 'a \ \text{list} \times 'a \ \text{bins}) \ \text{set}$ **where**
 $\text{wf-earley-input} = \{$
 $(k, \mathcal{G}, \omega, bs) \mid k \ \mathcal{G} \ \omega \ bs.$
 $k \leq \text{length } \omega \wedge$
 $\text{length } bs = \text{length } \omega + 1 \wedge$
 $\text{wf-bins } \mathcal{G} \ \omega \ bs$
 $\}$

typedef $'a \ \text{wf-bins} = \text{wf-earley-input}::(\text{nat} \times 'a \ \text{cfg} \times 'a \ \text{list} \times 'a \ \text{bins}) \ \text{set}$
morphisms *from-wf-bins to-wf-bins*
 using *Ex-wf-bins* by (*auto simp: wf-earley-input-def*)

lemma *wf-earley-input-elim*:
 assumes $(k, \mathcal{G}, \omega, bs) \in \text{wf-earley-input}$
 shows $k \leq \text{length } \omega \wedge k < \text{length } bs \wedge \text{length } bs = \text{length } \omega + 1 \wedge \text{wf-bins } \mathcal{G} \ \omega \ bs$
 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 $k \leq \text{length } \omega \ \text{length } bs = \text{length } \omega + 1 \ \text{wf-bins } \mathcal{G} \ \omega \ bs$
 shows $(k, \mathcal{G}, \omega, bs) \in \text{wf-earley-input}$
 by (*simp add: assms wf-earley-input-def*)

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

have *: $k \leq \text{length } \omega \text{ length } bs = \text{length } \omega + 1 \text{ wf-bins } \mathcal{G} \ \omega \ bs$
using *wf-earley-input-elim* *assms(1)* **by** *metis+*
have $x: x \in \text{set } (\text{items } (bs \ ! \ k))$
using *assms(2,3)* **by** *simp*
have *start-item* $x < \text{length } bs$
using $x \text{ wf-bins-kth-bin } * \text{ wf-item-def}$
by (*metis One-nat-def add.right-neutral add-Suc-right dual-order.trans le-imp-less-Suc*)
hence *wf-bins* $\mathcal{G} \ \omega \ (\text{upd-bins } bs \ k \ (\text{Complete}_L \ k \ x \ bs \ \text{red}))$
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 $(k, \mathcal{G}, \omega, bs) \in \text{wf-earley-input} \neg \text{length } (\text{items } (bs \ ! \ k)) \leq i$
assumes $x = \text{items } (bs \ ! \ k) \ ! \ i \ \text{next-symbol } x = \text{Some } a$
assumes $k < \text{length } \omega$
shows $(k, \mathcal{G}, \omega, \text{upd-bins } bs \ (k+1) \ (\text{Scan}_L \ k \ \omega \ a \ x \ \text{pre})) \in \text{wf-earley-input}$
proof –
have *: $k \leq \text{length } \omega \ \text{length } bs = \text{length } \omega + 1 \ \text{wf-bins } \mathcal{G} \ \omega \ bs$
using *wf-earley-input-elim* *assms(1)* **by** *metis+*
have $x: x \in \text{set } (\text{items } (bs \ ! \ k))$
using *assms(2,3)* **by** *simp*
have *wf-bins* $\mathcal{G} \ \omega \ (\text{upd-bins } bs \ (k+1) \ (\text{Scan}_L \ k \ \omega \ a \ x \ \text{pre}))$
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 $(k, \mathcal{G}, \omega, bs) \in \text{wf-earley-input} \neg \text{length } (\text{items } (bs \ ! \ k)) \leq i$
assumes $x = \text{items } (bs \ ! \ k) \ ! \ i \ \text{next-symbol } x = \text{Some } a$
shows $(k, \mathcal{G}, \omega, \text{upd-bins } bs \ k \ (\text{Predict}_L \ k \ \mathcal{G} \ a)) \in \text{wf-earley-input}$
proof –
have *: $k \leq \text{length } \omega \ \text{length } bs = \text{length } \omega + 1 \ \text{wf-bins } \mathcal{G} \ \omega \ bs$
using *wf-earley-input-elim* *assms(1)* **by** *metis+*
have $x: x \in \text{set } (\text{items } (bs \ ! \ k))$
using *assms(2,3)* **by** *simp*
hence *wf-bins* $\mathcal{G} \ \omega \ (\text{upd-bins } bs \ k \ (\text{Predict}_L \ k \ \mathcal{G} \ a))$
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* :: $\text{nat} \times 'a \ \text{cfg} \times 'a \ \text{list} \times 'a \ \text{bins} \Rightarrow \text{nat} \Rightarrow \text{nat}$ **where**
earley-measure $(k, \mathcal{G}, \omega, bs) \ i = \text{card } \{ x \mid x. \text{wf-item } \mathcal{G} \ \omega \ x \wedge \text{end-item } x = k \}$

– i

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

$i \geq \text{length } (\text{items } (bs ! k)) \implies \text{Earley}_L\text{-bin}' k \mathcal{G} \omega bs i = bs$
 $\neg i \geq \text{length } (\text{items } (bs ! k)) \implies x = \text{items } (bs!k) ! i \implies \text{next-symbol } x = \text{None}$
 \implies
 $\text{Earley}_L\text{-bin}' k \mathcal{G} \omega bs i = \text{Earley}_L\text{-bin}' k \mathcal{G} \omega (\text{upd-bins } bs k (\text{Complete}_L k x bs i)) (i+1)$
 $\neg i \geq \text{length } (\text{items } (bs ! k)) \implies x = \text{items } (bs!k) ! i \implies \text{next-symbol } x = \text{Some } a \implies$
 $a \notin \text{nonterminals } \mathcal{G} \implies k < \text{length } \omega \implies \text{Earley}_L\text{-bin}' k \mathcal{G} \omega bs i = \text{Earley}_L\text{-bin}' k \mathcal{G} \omega (\text{upd-bins } bs (k+1) (\text{Scan}_L k \omega a x i)) (i+1)$
 $\neg i \geq \text{length } (\text{items } (bs ! k)) \implies x = \text{items } (bs!k) ! i \implies \text{next-symbol } x = \text{Some } a \implies$
 $a \notin \text{nonterminals } \mathcal{G} \implies \neg k < \text{length } \omega \implies \text{Earley}_L\text{-bin}' k \mathcal{G} \omega bs i = \text{Earley}_L\text{-bin}' k \mathcal{G} \omega bs (i+1)$
 $\neg i \geq \text{length } (\text{items } (bs ! k)) \implies x = \text{items } (bs!k) ! i \implies \text{next-symbol } x = \text{Some } a \implies$
 $a \in \text{nonterminals } \mathcal{G} \implies \text{Earley}_L\text{-bin}' k \mathcal{G} \omega bs i = \text{Earley}_L\text{-bin}' k \mathcal{G} \omega (\text{upd-bins } bs k (\text{Predict}_L k \mathcal{G} a)) (i+1)$
by (*subst Earley_L-bin'.simps, auto*) $+$

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

assumes $(k, \mathcal{G}, \omega, bs) \in \text{wf-earley-input}$
assumes *base*: $\bigwedge k \mathcal{G} \omega bs i. i \geq \text{length } (\text{items } (bs ! k)) \implies P k \mathcal{G} \omega bs i$
assumes *complete*: $\bigwedge k \mathcal{G} \omega bs i x. \neg i \geq \text{length } (\text{items } (bs ! k)) \implies x = \text{items } (bs ! k) ! i \implies$
 $\text{next-symbol } x = \text{None} \implies P k \mathcal{G} \omega (\text{upd-bins } bs k (\text{Complete}_L k x bs i)) (i+1) \implies P k \mathcal{G} \omega bs i$
assumes *scan*: $\bigwedge k \mathcal{G} \omega bs i x a. \neg i \geq \text{length } (\text{items } (bs ! k)) \implies x = \text{items } (bs ! k) ! i \implies$
 $\text{next-symbol } x = \text{Some } a \implies a \notin \text{nonterminals } \mathcal{G} \implies k < \text{length } \omega \implies$
 $P k \mathcal{G} \omega (\text{upd-bins } bs (k+1) (\text{Scan}_L k \omega a x i)) (i+1) \implies P k \mathcal{G} \omega bs i$
assumes *pass*: $\bigwedge k \mathcal{G} \omega bs i x a. \neg i \geq \text{length } (\text{items } (bs ! k)) \implies x = \text{items } (bs ! k) ! i \implies$
 $\text{next-symbol } x = \text{Some } a \implies a \notin \text{nonterminals } \mathcal{G} \implies \neg k < \text{length } \omega$
 \implies
 $P k \mathcal{G} \omega bs (i+1) \implies P k \mathcal{G} \omega bs i$
assumes *predict*: $\bigwedge k \mathcal{G} \omega bs i x a. \neg i \geq \text{length } (\text{items } (bs ! k)) \implies x = \text{items } (bs ! k) ! i \implies$
 $\text{next-symbol } x = \text{Some } a \implies a \in \text{nonterminals } \mathcal{G} \implies$
 $P k \mathcal{G} \omega (\text{upd-bins } bs k (\text{Predict}_L k \mathcal{G} a)) (i+1) \implies P k \mathcal{G} \omega bs i$
shows $P k \mathcal{G} \omega bs i$
using *assms(1)*
proof (*induction n \equiv earley-measure (k, G, ω, bs) i arbitrary: bs i rule: nat-less-induct*)
case 1
have *wf*: $k \leq \text{length } \omega \text{ length } bs = \text{length } \omega + 1 \text{ wf-bins } \mathcal{G} \omega bs$
using *1.premis wf-earley-input-elim by metis+*
hence *k*: $k < \text{length } bs$

```

  by simp
have fin: finite { x | x. wf-item  $\mathcal{G}$   $\omega$  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 (CompleteL 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  $\mathcal{G}$   $\omega$  ?bs'
    using wf-bins-CompleteL wf(3) wf-bins-upd-bins k x by metis
  hence wf': (k,  $\mathcal{G}$ ,  $\omega$ , ?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  $\mathcal{G}$   $\omega$  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  $\mathcal{G}$   $\omega$  x  $\wedge$  end-item x = k}
    using card-mono fin sub by blast
  finally have card {x | x. wf-item  $\mathcal{G}$   $\omega$  x  $\wedge$  end-item x = k} > i
    by blast
  hence earley-measure (k,  $\mathcal{G}$ ,  $\omega$ , ?bs') (Suc i) < earley-measure (k,  $\mathcal{G}$ ,  $\omega$ , 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  $\omega$ 

```

```

let ?bs' = upd-bins bs (k+1) (ScanL k ω a ?x i)
have wf-bins': wf-bins  $\mathcal{G}$  ω ?bs'
  using wf-bins-ScanL 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 (PredictL k  $\mathcal{G}$  a)
have wf-bins': wf-bins  $\mathcal{G}$  ω ?bs'
  using wf-bins-PredictL 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  $\mathcal{G}$   $\omega$  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  $\mathcal{G}$   $\omega$  x  $\wedge$  end-item x = k}
    using card-mono fin sub by blast
    finally have card {x | x. wf-item  $\mathcal{G}$   $\omega$  x  $\wedge$  end-item x = k} > i
    by blast
    hence earley-measure (k,  $\mathcal{G}$ ,  $\omega$ , ?bs') (Suc i) < earley-measure (k,  $\mathcal{G}$ ,  $\omega$ , 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-EarleyL-bin':
  assumes (k,  $\mathcal{G}$ ,  $\omega$ , bs)  $\in$  wf-earley-input
  shows (k,  $\mathcal{G}$ ,  $\omega$ , EarleyL-bin' k  $\mathcal{G}$   $\omega$  bs i)  $\in$  wf-earley-input
  using assms
proof (induction i rule: EarleyL-bin'-induct[OF assms(1), case-names Base CompleteF ScanF Pass PredictF])
  case (CompleteF k  $\mathcal{G}$   $\omega$  bs i x)
  let ?bs' = upd-bins bs k (CompleteL k x bs i)
  have (k,  $\mathcal{G}$ ,  $\omega$ , ?bs')  $\in$  wf-earley-input
  using CompleteF.hyps CompleteF.prems wf-earley-input-CompleteL by blast
  thus ?case
  using CompleteF.IH CompleteF.hyps by simp
next
  case (ScanF k  $\mathcal{G}$   $\omega$  bs i x a)
  let ?bs' = upd-bins bs (k+1) (ScanL k  $\omega$  a x i)
  have (k,  $\mathcal{G}$ ,  $\omega$ , ?bs')  $\in$  wf-earley-input
  using ScanF.hyps ScanF.prems wf-earley-input-ScanL by metis
  thus ?case
  using ScanF.IH ScanF.hyps by simp
next
  case (PredictF k  $\mathcal{G}$   $\omega$  bs i x a)
  let ?bs' = upd-bins bs k (PredictL k  $\mathcal{G}$  a)
  have (k,  $\mathcal{G}$ ,  $\omega$ , ?bs')  $\in$  wf-earley-input
  using PredictF.hyps PredictF.prems wf-earley-input-PredictL by metis
  thus ?case

```

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

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

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

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

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

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

lemma *kth-Earley_L-bin'-bins*:
assumes $(k, \mathcal{G}, \omega, bs) \in \text{wf-earley-input}$
assumes $j < \text{length } (\text{items } (bs ! l))$
shows $\text{items } (\text{Earley}_L\text{-bin}' k \mathcal{G} \omega bs i ! l) ! j = \text{items } (bs ! l) ! j$
using *assms(2)*

proof $(\text{induction } i \text{ rule: Earley}_L\text{-bin}'\text{-induct}[OF \text{ assms}(1), \text{case-names Base Complete}_F \text{ Scan}_F \text{ Pass Predict}_F])$
case $(\text{Complete}_F k \mathcal{G} \omega bs i x)$
let $?bs' = \text{upd-bins } bs k (\text{Complete}_L k x bs i)$
have $\text{items } (\text{Earley}_L\text{-bin}' k \mathcal{G} \omega ?bs' (i + 1) ! l) ! j = \text{items } (?bs' ! l) ! j$
using *Complete_F.IH Complete_F.prems length-nth-upd-bin-bins items-def order.strict-trans2* **by** $(\text{metis length-map})$
also have $\dots = \text{items } (bs ! l) ! j$
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 $(\text{Scan}_F k \mathcal{G} \omega bs i x a)$

```

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

```

```

lemma nth-bin-sub-EarleyL-bin':
  assumes (k,  $\mathcal{G}$ , ω, bs) ∈ wf-earley-input
  shows set (items (bs ! l)) ⊆ set (items (EarleyL-bin' k  $\mathcal{G}$  ω bs i ! l))
proof standard
  fix x
  assume x ∈ 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 (EarleyL-bin' k  $\mathcal{G}$  ω bs i ! l) ! j
    using kth-EarleyL-bin'-bins assms * by metis
  moreover have j < length (items (EarleyL-bin' k  $\mathcal{G}$  ω bs i ! l))
    using assms *(1) length-nth-bin-EarleyL-bin' less-le-trans by blast
  ultimately show x ∈ set (items (EarleyL-bin' k  $\mathcal{G}$  ω bs i ! l))
    by simp
qed

```

```

lemma nth-EarleyL-bin'-eq:
  assumes (k,  $\mathcal{G}$ , ω, bs) ∈ wf-earley-input
  shows l < k ⇒ EarleyL-bin' k  $\mathcal{G}$  ω bs i ! l = bs ! l
  by (induction i rule: EarleyL-bin'-induct[OF assms]) (auto simp: upd-bins-def)

```

```

lemma set-items-EarleyL-bin'-eq:
  assumes (k,  $\mathcal{G}$ , ω, bs) ∈ wf-earley-input
  shows l < k ⇒ set (items (EarleyL-bin' k  $\mathcal{G}$  ω bs i ! l)) = set (items (bs ! l))
  by (simp add: assms nth-EarleyL-bin'-eq)

```

```

lemma bins-upto-k0-EarleyL-bin'-eq:

```

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

lemma $\text{wf-earley-input-Init}_L$:

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

proof –

let $?rs = \text{filter } (\lambda r. \text{lhs-rule } r = \mathfrak{S} \mathcal{G}) (\text{remdups } (\mathfrak{R} \mathcal{G}))$

let $?b0 = \text{map } (\lambda r. (\text{init-item } r 0, \text{Null})) ?rs$

let $?bs = \text{replicate } (\text{length } \omega + 1) ([])$

have $\text{distinct } (\text{items } ?b0)$

using $\text{assms unfolding wf-bin-def wf-item-def items-def}$

by $(\text{auto simp: init-item-def distinct-map inj-on-def})$

moreover have $\forall x \in \text{set } (\text{items } ?b0). \text{wf-item } \mathcal{G} \omega x \wedge \text{end-item } x = 0$

using $\text{assms unfolding wf-bin-def wf-item-def}$ **by** $(\text{auto simp: init-item-def}$

$\text{items-def})$

moreover have $\text{wf-bins } \mathcal{G} \omega ?bs$

unfolding $\text{wf-bins-def wf-bin-def wf-bin-items-def items-def}$ **using** $\text{less-Suc-eq-0-disj}$

by force

ultimately show $?thesis$

using $\text{assms length-replicate wf-earley-input-intro}$

unfolding $\text{wf-bin-def Init}_L\text{-def wf-bin-def wf-bin-items-def wf-bins-def}$

by $(\text{metis (no-types, lifting) length-list-update nth-list-update-eq nth-list-update-neq})$

qed

lemma $\text{length-bins-Init}_L[\text{simp}]$:

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

by $(\text{simp add: Init}_L\text{-def})$

lemma $\text{wf-earley-input-Earley}_L\text{-bins}[\text{simp}]$:

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

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

using assms

proof $(\text{induction } k)$

case 0

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

using $\text{assms wf-earley-input-Init}_L$ **by** blast

thus $?case$

by $(\text{simp add: assms wf-earley-input-Init}_L \text{wf-earley-input-Earley}_L\text{-bin})$

next

case $(\text{Suc } k)$

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

using $\text{Suc.IH Suc.prem1(1) Suc-leD assms wf-earley-input-elim wf-earley-input-intro}$

by metis

thus $?case$

by $(\text{simp add: wf-earley-input-Earley}_L\text{-bin})$

qed

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

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

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

8.6 Soundness

lemma *Init_L-eq-Init_F*:
 $\text{bins } (\text{Init}_L \mathcal{G} \omega) = \text{Init}_F \mathcal{G}$
proof –
let $?rs = \text{filter } (\lambda r. \text{lhs-rule } r = \mathfrak{S} \mathcal{G}) (\text{remdups } (\mathfrak{R} \mathcal{G}))$
let $?b0 = \text{map } (\lambda r. (\text{init-item } r \ 0, \text{Null})) ?rs$
let $?bs = \text{replicate } (\text{length } \omega + 1) (\[])$
have $\text{bins } (?bs[0 := ?b0]) = \text{set } (\text{items } ?b0)$
proof –
have $\text{bins } (?bs[0 := ?b0]) = \bigcup \{ \text{set } (\text{items } ((?bs[0 := ?b0]) ! k)) \mid k. k < \text{length } (?bs[0 := ?b0]) \}$
unfolding *bins-def* **by** *blast*
also have $\dots = \text{set } (\text{items } ((?bs[0 := ?b0]) ! 0)) \cup \bigcup \{ \text{set } (\text{items } ((?bs[0 := ?b0]) ! k)) \mid k. k < \text{length } (?bs[0 := ?b0]) \wedge k \neq 0 \}$
by *fastforce*
also have $\dots = \text{set } (\text{items } (?b0))$
by (*auto simp: items-def*)
finally show *?thesis* .
qed
also have $\dots = \text{Init}_F \mathcal{G}$
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 $\text{wf-bins } \mathcal{G} \omega \text{ bs bins } \text{bs} \subseteq I \ x \in \text{set } (\text{items } (\text{bs} ! k)) \ k < \text{length } \text{bs} \ k < \text{length } \omega$
assumes $\text{next-symbol } x = \text{Some } a$
shows $\text{set } (\text{items } (\text{Scan}_L \ k \ \omega \ a \ x \ \text{pre})) \subseteq \text{Scan}_F \ k \ \omega \ I$
proof *standard*
fix y
assume $*$: $y \in \text{set } (\text{items } (\text{Scan}_L \ k \ \omega \ a \ x \ \text{pre}))$

have $x \in \text{bin } I \ k$
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 #: $k < \text{length } \omega \ \omega!k = a$
 hence $y = \text{inc-item } x \ (k+1)$
 using * **unfolding** *Scan_L-def* **by** (*simp add: items-def*)
 hence $y \in \text{Scan}_F \ k \ \omega \ I$
 using $\langle x \in \text{bin } I \ k \rangle$ # *assms(6)* **unfolding** *Scan_F-def* **by** *blast*
}
thus $y \in \text{Scan}_F \ k \ \omega \ I$
using * *assms(5)* **unfolding** *Scan_L-def* **by** (*auto simp: items-def*)
qed

lemma *Predict_L-sub-Predict_F*:

assumes *wf-bins* $\mathcal{G} \ \omega \ bs \ \text{bins} \ bs \subseteq I \ x \in \text{set } (\text{items } (bs \ ! \ k)) \ k < \text{length } bs$
assumes *next-symbol* $x = \text{Some } X$
shows $\text{set } (\text{items } (\text{Predict}_L \ k \ \mathcal{G} \ X)) \subseteq \text{Predict}_F \ k \ \mathcal{G} \ I$
proof *standard*
 fix y
 assume *: $y \in \text{set } (\text{items } (\text{Predict}_L \ k \ \mathcal{G} \ X))$
 have $x \in \text{bin } I \ k$
 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 $?rs = \text{filter } (\lambda r. \text{lhs-rule } r = X) \ (\mathfrak{R} \ \mathcal{G})$
 let $?xs = \text{map } (\lambda r. \text{init-item } r \ k) \ ?rs$
 have $y \in \text{set } ?xs$
 using * **unfolding** *Predict_L-def* *items-def* **by** *simp*
 then obtain r **where** $y = \text{init-item } r \ k \ \text{lhs-rule } r = X \ r \in \text{set } (\mathfrak{R} \ \mathcal{G}) \ \text{next-symbol}$
 $x = \text{Some } (\text{lhs-rule } r)$
 using *assms(5)* **by** *auto*
 thus $y \in \text{Predict}_F \ k \ \mathcal{G} \ I$
 unfolding *Predict_F-def* **using** $\langle x \in \text{bin } I \ k \rangle$ **by** *blast*
qed

lemma *Complete_L-sub-Complete_F*:

assumes *wf-bins* $\mathcal{G} \ \omega \ bs \ \text{bins} \ bs \subseteq I \ y \in \text{set } (\text{items } (bs \ ! \ k)) \ k < \text{length } bs$
assumes *next-symbol* $y = \text{None}$
shows $\text{set } (\text{items } (\text{Complete}_L \ k \ y \ bs \ \text{red})) \subseteq \text{Complete}_F \ k \ I$
proof *standard*
 fix x
 assume *: $x \in \text{set } (\text{items } (\text{Complete}_L \ k \ y \ bs \ \text{red}))$
 have $y \in \text{bin } I \ k$
 using *kth-bin-sub-bins* *assms* *items-def* *wf-bin-def* *wf-bins-def* *bin-def* *wf-bin-items-def*
by *fast*
 let $?orig = bs \ ! \ \text{start-item } y$
 let $?xs = \text{filter-with-index } (\lambda x. \text{next-symbol } x = \text{Some } (\text{lhs-item } y)) \ (\text{items } ?orig)$
 let $?xs' = \text{map } (\lambda(x, \text{pre}). (\text{inc-item } x \ k, \text{PreRed } (\text{start-item } y, \text{pre}, \text{red}) \ [])) \ ?xs$
 have $0: \text{start-item } y < \text{length } bs$

```

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 ∈ set (map fst ?xs)
  have next-symbol z = Some (lhs-item y)
    using * by (simp add: filter-with-index-cong-filter)
  moreover have z ∈ 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 ∈ bin I (start-item y)
    by simp-all
}
hence 1: ∀ z ∈ set (map fst ?xs). next-symbol z = Some (lhs-item y) ∧ z ∈ bin
I (start-item y)
  by blast
obtain z where z: x = inc-item z k z ∈ set (map fst ?xs)
  using * unfolding CompleteL-def by (auto simp: rev-image-eqI items-def)
moreover have next-symbol z = Some (lhs-item y) z ∈ bin I (start-item y)
  using 1 z by blast+
ultimately show x ∈ CompleteF k I
  using ⟨y ∈ bin I k⟩ assms(5) unfolding CompleteF-def next-symbol-def by
(auto split: if-splits)
qed

```

lemma sound-Scan_L:

```

assumes wf-bins  $\mathcal{G}$   $\omega$  bs bins bs ⊆ I x ∈ set (items (bs!k)) k < length bs k <
length  $\omega$ 
assumes next-symbol x = Some a ∀ x ∈ I. wf-item  $\mathcal{G}$   $\omega$  x ∀ x ∈ I. sound-item  $\mathcal{G}$ 
 $\omega$  x
shows ∀ x ∈ set (items (ScanL k  $\omega$  a x i)). sound-item  $\mathcal{G}$   $\omega$  x
proof standard
  fix y
  assume y ∈ set (items (ScanL k  $\omega$  a x i))
  hence y ∈ ScanF k  $\omega$  I
    by (meson ScanL-sub-ScanF assms(1-6) in-mono)
  thus sound-item  $\mathcal{G}$   $\omega$  y
    using sound-Scan assms(7,8) unfolding ScanF-def inc-item-def bin-def
    by (smt (verit, best) item.exhaust-sel mem-Collect-eq)
qed

```

lemma sound-Predict_L:

```

assumes wf-bins  $\mathcal{G}$   $\omega$  bs bins bs ⊆ I x ∈ set (items (bs!k)) k < length bs
assumes next-symbol x = Some X ∀ x ∈ I. wf-item  $\mathcal{G}$   $\omega$  x ∀ x ∈ I. sound-item
 $\mathcal{G}$   $\omega$  x
shows ∀ x ∈ set (items (PredictL k  $\mathcal{G}$  X)). sound-item  $\mathcal{G}$   $\omega$  x
proof standard
  fix y
  assume y ∈ set (items (PredictL k  $\mathcal{G}$  X))

```

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

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

fix x
assume $x \in \text{set}(\text{items}(\text{Complete}_L k y bs i))$
hence $x \in \text{Complete}_F k I$
using *Complete_L-sub-Complete_F assms(1-5) by blast*
thus *sound-item* $\mathcal{G} \omega x$
using *sound-Complete assms(6,7) unfolding 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 $(k, \mathcal{G}, \omega, bs) \in \text{wf-earley-input}$
assumes $\forall x \in \text{bins } bs. \text{sound-item } \mathcal{G} \omega x$
shows $\forall x \in \text{bins}(\text{Earley}_L\text{-bin}' k \mathcal{G} \omega bs i). \text{sound-item } \mathcal{G} \omega x$
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 $?bs' = \text{upd-bins } bs k (\text{Complete}_L k x bs i)$
have $x \in \text{set}(\text{items}(bs!k))$
using *Complete_F.hyps(1,2) by force*
hence $\forall x \in \text{set}(\text{items}(\text{Complete}_L k x bs i)). \text{sound-item } \mathcal{G} \omega x$
using *sound-Complete_L Complete_F.hyps(3) Complete_F.prems wf-earley-input-elim*
wf-bins-impl-wf-items **by** *fastforce*
moreover $(k, \mathcal{G}, \omega, ?bs') \in \text{wf-earley-input}$
using *Complete_F.hyps Complete_F.prems(1) wf-earley-input-Complete_L by blast*
ultimately $\forall x \in \text{bins}(\text{Earley}_L\text{-bin}' k \mathcal{G} \omega ?bs' (i + 1)). \text{sound-item } \mathcal{G} \omega x$
using *Complete_F.IH 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 *Complete_F.hyps by simp*

next
case (*Scan_F k* $\mathcal{G} \omega bs i x a$)
let $?bs' = \text{upd-bins } bs (k+1) (\text{Scan}_L k \omega a x i)$
have $x \in \text{set}(\text{items}(bs!k))$
using *Scan_F.hyps(1,2) by force*

hence $\forall x \in \text{set } (\text{items } (\text{Scan}_L k \omega a x i)). \text{sound-item } \mathcal{G} \omega x$
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 $(k, \mathcal{G}, \omega, ?bs') \in \text{wf-earley-input}$
using *Scan_F.hyps Scan_F.prems(1) wf-earley-input-Scan_L* **by metis**
ultimately have $\forall x \in \text{bins } (\text{Earley}_L\text{-bin}' k \mathcal{G} \omega ?bs' (i + 1)). \text{sound-item } \mathcal{G} \omega x$
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 $(\text{Predict}_F k \mathcal{G} \omega bs i x a)$
let $?bs' = \text{upd-bins } bs k (\text{Predict}_L k \mathcal{G} a)$
have $x \in \text{set } (\text{items } (bs ! k))$
using *Predict_F.hyps(1,2)* **by force**
hence $\forall x \in \text{set } (\text{items}(\text{Predict}_L k \mathcal{G} a)). \text{sound-item } \mathcal{G} \omega x$
using *sound-Predict_L Predict_F.hyps(3) Predict_F.prems wf-earley-input-elim wf-bins-impl-wf-items* **by fast**
moreover have $(k, \mathcal{G}, \omega, ?bs') \in \text{wf-earley-input}$
using *Predict_F.hyps Predict_F.prems(1) wf-earley-input-Predict_L* **by metis**
ultimately have $\forall x \in \text{bins } (\text{Earley}_L\text{-bin}' k \mathcal{G} \omega ?bs' (i + 1)). \text{sound-item } \mathcal{G} \omega x$
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 $(k, \mathcal{G}, \omega, bs) \in \text{wf-earley-input}$
assumes $\forall x \in \text{bins } bs. \text{sound-item } \mathcal{G} \omega x$
shows $\forall x \in \text{bins } (\text{Earley}_L\text{-bin}' k \mathcal{G} \omega bs). \text{sound-item } \mathcal{G} \omega x$
using *sound-Earley_L-bin' assms Earley_L-bin-def* **by metis**

lemma *Earley_L-bin'-sub-Earley_F-bin:*
assumes $(k, \mathcal{G}, \omega, bs) \in \text{wf-earley-input}$
assumes $\text{bins } bs \subseteq I$
shows $\text{bins } (\text{Earley}_L\text{-bin}' k \mathcal{G} \omega bs i) \subseteq \text{Earley}_F\text{-bin } k \mathcal{G} \omega I$
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 $(\text{Base } k \mathcal{G} \omega bs i)$
thus *?case*
using *Earley_F-bin-mono* **by fastforce**
next
case $(\text{Complete}_F k \mathcal{G} \omega bs i x)$
let $?bs' = \text{upd-bins } bs k (\text{Complete}_L k x bs i)$

have $x \in \text{set } (\text{items } (bs \ ! \ k))$
using $\text{Complete}_F.\text{hyps}(1,2)$ **by force**
hence $\text{bins } ?bs' \subseteq I \cup \text{Complete}_F \ k \ I$
using $\text{Complete}_L\text{-sub-Complete}_F \ \text{Complete}_F.\text{hyps}(3) \ \text{Complete}_F.\text{prems}(1,2)$
bins-upd-bins wf-earley-input-elim **by blast**
moreover have $(k, \mathcal{G}, \omega, ?bs') \in \text{wf-earley-input}$
using $\text{Complete}_F.\text{hyps} \ \text{Complete}_F.\text{prems}(1) \ \text{wf-earley-input-Complete}_L$ **by blast**
ultimately have $\text{bins } (\text{Earley}_L\text{-bin}' \ k \ \mathcal{G} \ \omega \ bs \ i) \subseteq \text{Earley}_F\text{-bin } k \ \mathcal{G} \ \omega \ (I \cup \text{Complete}_F \ k \ I)$
using $\text{Complete}_F.IH \ \text{Complete}_F.\text{hyps}$ **by simp**
also have $\dots \subseteq \text{Earley}_F\text{-bin } k \ \mathcal{G} \ \omega \ (\text{Earley}_F\text{-bin } k \ \mathcal{G} \ \omega \ I)$
using $\text{Complete}_F\text{-Earley}_F\text{-bin-mono} \ \text{Earley}_F\text{-bin-mono} \ \text{Earley}_F\text{-bin-sub-mono}$
by (*metis Un-subset-iff*)
finally show *?case*
using $\text{Earley}_F\text{-bin-idem}$ **by blast**

next

case $(\text{Scan}_F \ k \ \mathcal{G} \ \omega \ bs \ i \ x \ a)$
let $?bs' = \text{upd-bins } bs \ (k+1) \ (\text{Scan}_L \ k \ \omega \ a \ x \ i)$
have $x \in \text{set } (\text{items } (bs \ ! \ k))$
using $\text{Scan}_F.\text{hyps}(1,2)$ **by force**
hence $\text{bins } ?bs' \subseteq I \cup \text{Scan}_F \ k \ \omega \ I$
using $\text{Scan}_L\text{-sub-Scan}_F \ \text{Scan}_F.\text{hyps}(3,5) \ \text{Scan}_F.\text{prems} \ \text{bins-upd-bins} \ \text{wf-earley-input-elim}$
by (*metis add-mono1 sup-mono*)
moreover have $(k, \mathcal{G}, \omega, ?bs') \in \text{wf-earley-input}$
using $\text{Scan}_F.\text{hyps} \ \text{Scan}_F.\text{prems}(1) \ \text{wf-earley-input-Scan}_L$ **by metis**
ultimately have $\text{bins } (\text{Earley}_L\text{-bin}' \ k \ \mathcal{G} \ \omega \ bs \ i) \subseteq \text{Earley}_F\text{-bin } k \ \mathcal{G} \ \omega \ (I \cup \text{Scan}_F \ k \ \omega \ I)$
using $\text{Scan}_F.IH \ \text{Scan}_F.\text{hyps}$ **by simp**
thus *?case*
using $\text{Scan}_F\text{-Earley}_F\text{-bin-mono} \ \text{Earley}_F\text{-bin-mono} \ \text{Earley}_F\text{-bin-sub-mono} \ \text{Earley}_F\text{-bin-idem}$
by (*metis Un-subset-iff subset-trans*)

next

case $(\text{Pass } k \ \mathcal{G} \ \omega \ bs \ i \ x \ a)$
thus *?case*
by *simp*

next

case $(\text{Predict}_F \ k \ \mathcal{G} \ \omega \ bs \ i \ x \ a)$
let $?bs' = \text{upd-bins } bs \ k \ (\text{Predict}_L \ k \ \mathcal{G} \ a)$
have $x \in \text{set } (\text{items } (bs \ ! \ k))$
using $\text{Predict}_F.\text{hyps}(1,2)$ **by force**
hence $\text{bins } ?bs' \subseteq I \cup \text{Predict}_F \ k \ \mathcal{G} \ I$
using $\text{Predict}_L\text{-sub-Predict}_F \ \text{Predict}_F.\text{hyps}(3) \ \text{Predict}_F.\text{prems} \ \text{bins-upd-bins}$
wf-earley-input-elim
by (*metis sup-mono*)
moreover have $(k, \mathcal{G}, \omega, ?bs') \in \text{wf-earley-input}$
using $\text{Predict}_F.\text{hyps} \ \text{Predict}_F.\text{prems}(1) \ \text{wf-earley-input-Predict}_L$ **by metis**
ultimately have $\text{bins } (\text{Earley}_L\text{-bin}' \ k \ \mathcal{G} \ \omega \ bs \ i) \subseteq \text{Earley}_F\text{-bin } k \ \mathcal{G} \ \omega \ (I \cup \text{Predict}_F \ k \ \mathcal{G} \ I)$

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

lemma *Earley_L-bins-sub-Earley_F-bins*:
assumes $k \leq \text{length}\ \omega$
shows $bins\ (\text{Earley}_L\text{-bins}\ k\ \mathcal{G}\ \omega) \subseteq \text{Earley}_F\text{-bins}\ k\ \mathcal{G}\ \omega$
using *assms*
proof (*induction k*)
case 0
have $(k, \mathcal{G}, \omega, \text{Init}_L\ \mathcal{G}\ \omega) \in \text{wf-earley-input}$
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 $(\text{Suc}\ k, \mathcal{G}, \omega, \text{Earley}_L\text{-bins}\ k\ \mathcal{G}\ \omega) \in \text{wf-earley-input}$
by (*simp add: Suc.prem(1) Suc-leD assms wf-earley-input-intro*)
thus *?case*
by (*simp add: Suc.IH Suc.prem(1) Suc-leD Earley_L-bin-sub-Earley_F-bin assms*)
qed

lemma *Earley_L-sub-Earley_F*:
 $bins\ (\text{Earley}_L\ \mathcal{G}\ \omega) \subseteq \text{Earley}_F\ \mathcal{G}\ \omega$
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* $(bins\ (\text{Earley}_L\ \mathcal{G}\ \omega))\ \mathcal{G}\ \omega$
shows $\mathcal{G} \vdash [\mathcal{G}] \Rightarrow^* \omega$
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 $\text{wf-bins}\ \mathcal{G}\ \omega\ bs\ k < \text{length}\ bs\ i \geq \text{length}\ (\text{items}\ (bs\ !\ k))\ l \leq k$
shows $\text{bin}\ (\text{bins-upto}\ bs\ k\ i)\ l = \text{bin}\ (\text{bins}\ bs)\ l$
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 $\mathcal{G} \ \omega \ x$ is-complete x start-item $x = k$ end-item $x = k$
nonempty-derives \mathcal{G}

shows False

proof –

have $\mathcal{G} \vdash [\text{lhs-item } x] \Rightarrow^* []$

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 $z = \text{Some } a$ $a \notin \text{nonterminals } \mathcal{G} \ \forall x \in I$. wf-item $\mathcal{G} \ \omega \ x$
wf-item $\mathcal{G} \ \omega \ z$

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

proof (rule ccontr)

assume Complete_F $k (I \cup \{z\}) \neq \text{Complete}_F k I$

hence Complete_F $k I \subset \text{Complete}_F k (I \cup \{z\})$

using Complete_F-sub-mono by blast

then obtain $w \ x \ y$ where *:

$w \in \text{Complete}_F k (I \cup \{z\})$ $w \notin \text{Complete}_F k I$ $w = \text{inc-item } x \ k$

$x \in \text{bin } (I \cup \{z\})$ (start-item y) $y \in \text{bin } (I \cup \{z\})$ k

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

unfolding Complete_F-def by fast

show False

proof (cases $x = z$)

case True

have lhs-item $y \in \text{nonterminals } \mathcal{G}$

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

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

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

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

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 $Complete_F\text{-sub-mono}$ **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-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** *fast*
consider $(A) x' = z \mid (B) y = z$
using $*(2-7)$ $Complete_F\text{-def}$ **by** (*auto simp: bin-def; blast*)
thus *False*
proof *cases*
case A
have $start\text{-item } y = k$
using $*(4)$ A $bin\text{-def}$ $assms(5)$ **by** (*metis (mono-tags, lifting) mem-Collect-eq*)
moreover **have** $end\text{-item } y = k$
using $*(5)$ $bin\text{-def}$ **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 $*(5)$ $assms(1,4)$ $wf\text{-item-def}$ **by** (*auto simp: bin-def*)
ultimately show *?thesis*
using $impossible\text{-complete-item } *(6)$ $assms(3)$ **by** *blast*
next
case B
thus *?thesis*
using $*(6)$ $assms(6)$ **by** (*auto simp: next-symbol-def*)
qed
qed

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

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

lemma $Complete_F\text{-sub-bins-Un-Complete}_L$:
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 $x \in bin\ I\ (start\text{-}item\ z)$
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 $wf\text{-}item\text{-}in\text{-}kth\text{-}bin\ assms(2,4)\ bin\text{-}def$ **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 $Complete_L\text{-}eq\text{-}start\text{-}item$:

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

by *(auto simp: Complete_L-def)*

lemma $kth\text{-}bin\text{-}bins\text{-}upto\text{-}empty$:

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 $bins\text{-}upto\text{-}def\ bin\text{-}upto\text{-}def$ **by** *blast*

hence $end\text{-}item\ x = l$

```

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

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

```

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

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

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

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

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

using *assms*

proof (*induction i rule: Earley_L-bin'-induct*[*OF assms(1)*], *case-names Base Complete_F Scan_F Pass Predict_F*)

case (*Base* $k \ \mathcal{G} \ \omega \ bs \ i$)

have $bin \ (bins \ bs) \ k = bin \ (bins\text{-upto } bs \ k \ i) \ k$

using *Base.hyps Base.prem(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.prem(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 $?bs' = \text{upd-bins } bs \ k \ (\text{Complete}_L \ k \ x \ bs \ i)$

have $x: x \in set \ (items \ (bs \ ! \ k))$

using *Complete_F.hyps(1,2)* **by** *auto*

have $\text{wf}: (k, \mathcal{G}, \omega, ?bs') \in \text{wf-earley-input}$

using *Complete_F.hyps Complete_F.prems(1) wf-earley-input-Complete_L* **by** *blast*

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

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 $\text{Scan}_F \ k \ \omega \ (bins\text{-upto } ?bs' \ k \ (i + 1)) \subseteq bins \ ?bs'$

proof –

have $\text{Scan}_F \ k \ \omega \ (bins\text{-upto } ?bs' \ k \ (i + 1)) = \text{Scan}_F \ k \ \omega \ (bins\text{-upto } ?bs' \ k \ i \cup \{items \ (?bs' \ ! \ k) \ ! \ i\})$

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 $\dots = \text{Scan}_F \ k \ \omega \ (bins\text{-upto } bs \ k \ i \cup \{x\})$

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 $\dots \subseteq bins \ bs \cup \text{Scan}_F \ k \ \omega \ \{x\}$

using *Complete_F.prems(2,3) Scan_F-Un Scan_F-Earley_F-bin-step-mono* **by** *fastforce*

also have $\dots = bins \ bs$

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* $k \ \mathcal{G} \ (bins\text{-upto } ?bs' \ k \ (i + 1)) \subseteq bins \ ?bs'$

proof –

have *Predict_F* $k \ \mathcal{G} \ (bins\text{-upto } ?bs' \ k \ (i + 1)) = \text{Predict}_F \ k \ \mathcal{G} \ (bins\text{-upto } ?bs' \ k$

$i \cup \{items (?bs' ! k) ! i\}$
using $Complete_F.hyps(1)$ $bins\text{-}upto\text{-}Suc\text{-}Un$ $length\text{-}nth\text{-}upd\text{-}bin\text{-}bins$
by $(metis\ dual\text{-}order.\text{strict}\text{-}trans1\ items\text{-}def\ length\text{-}map\ not\text{-}le\text{-}imp\text{-}less)$
also have $\dots = Predict_F\ k\ \mathcal{G}\ (bins\text{-}upto\ bs\ k\ i \cup \{x\})$
using $Complete_F.hyps(1,2)$ $Complete_F.prem(1)$ $items\text{-}nth\text{-}idem\text{-}upd\text{-}bins$
 $bins\text{-}upto\text{-}kth\text{-}nth\text{-}idem$ $wf\text{-}earley\text{-}input\text{-}elim$
by $(metis\ dual\text{-}order.\text{refl}\ items\text{-}def\ length\text{-}map\ not\text{-}le\text{-}imp\text{-}less)$
also have $\dots \subseteq bins\ bs \cup Predict_F\ k\ \mathcal{G}\ \{x\}$
using $Complete_F.prem(2,3)$ $Predict_F\text{-}Un$ $Predict_F\text{-}Earley_F\text{-}bin\text{-}step\text{-}mono$
by $blast$
also have $\dots = bins\ bs$
using $Complete_F.hyps(3)$ **by** $(auto\ simp:\ Predict_F\text{-}def\ bin\text{-}def)$
finally show $?thesis$
using $Complete_F.prem(1)$ $wf\text{-}earley\text{-}input\text{-}elim$ $bins\text{-}upd\text{-}bins$ **by** $blast$
qed
moreover have $Complete_F\ k\ (bins\text{-}upto\ ?bs'\ k\ (i + 1)) \subseteq bins\ ?bs'$
proof –
have $Complete_F\ k\ (bins\text{-}upto\ ?bs'\ k\ (i + 1)) = Complete_F\ k\ (bins\text{-}upto\ ?bs'\ k\ i \cup \{items (?bs' ! k) ! i\})$
using $bins\text{-}upto\text{-}Suc\text{-}Un$ $length\text{-}nth\text{-}upd\text{-}bin\text{-}bins$ $Complete_F.hyps(1)$
by $(metis\ (no\text{-}types,\ opaque\text{-}lifting)\ dual\text{-}order.\text{trans}\ items\text{-}def\ length\text{-}map\ not\text{-}le\text{-}imp\text{-}less)$
also have $\dots = Complete_F\ k\ (bins\text{-}upto\ bs\ k\ i \cup \{x\})$
using $items\text{-}nth\text{-}idem\text{-}upd\text{-}bins$ $Complete_F.hyps(1,2)$ $bins\text{-}upto\text{-}kth\text{-}nth\text{-}idem$
 $Complete_F.prem(1)$ $wf\text{-}earley\text{-}input\text{-}elim$
by $(metis\ dual\text{-}order.\text{refl}\ items\text{-}def\ length\text{-}map\ not\text{-}le\text{-}imp\text{-}less)$
also have $\dots \subseteq bins\ bs \cup set\ (items\ (Complete_L\ k\ x\ bs\ i))$
using $Complete_F\text{-}sub\text{-}bins\text{-}Un\ Complete_L$ $Complete_F.hyps(3)$ $Complete_F.prem(1,2,3)$
 $next\text{-}symbol\text{-}def$
 $bins\text{-}upto\text{-}sub\text{-}bins$ $wf\text{-}bins\text{-}kth\text{-}bin\ x$ $Complete_F\text{-}Earley_F\text{-}bin\text{-}step\text{-}mono$ $wf\text{-}earley\text{-}input\text{-}elim$
by $(smt\ (verit,\ best)\ option.\text{distinct}(1)\ subset\text{-}trans)$
finally show $?thesis$
using $Complete_F.prem(1)$ $wf\text{-}earley\text{-}input\text{-}elim$ $bins\text{-}upd\text{-}bins$ **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 $Complete_F.IH$ $Complete_F.prem\ sound\ wf$ $Earley_F\text{-}bin\text{-}step\text{-}def$ $bins\text{-}upto\text{-}sub\text{-}bins$
 $wf\text{-}earley\text{-}input\text{-}elim$ $bins\text{-}upd\text{-}bins$
by $(metis\ UnE\ sup.\text{bounded}I)$
thus $?case$
using $Complete_F.hyps$ $Complete_F.prem(1)$ $Earley_L\text{-}bin'\text{-}simps(2)$ $Earley_F\text{-}bin\text{-}step\text{-}sub\text{-}mono$
 $bins\text{-}upd\text{-}bins$ $wf\text{-}earley\text{-}input\text{-}elim$
by $(smt\ (verit,\ best)\ sup.\text{cobounded}I2\ sup.\text{order}E\ sup.\text{ge}1)$
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 $sound: \forall x \in set\ (items\ (Scan_L\ k\ \omega\ a\ x\ i)).\ sound\text{-}item\ \mathcal{G}\ \omega\ x$

using *sound-Scan_L Scan_F.hypos(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, G, ω, ?bs') ∈ wf-earley-input*
using *Scan_F.hypos Scan_F.prems(1) wf-earley-input-Scan_L by metis*
have *Scan_F k ω (bins-upto ?bs' k (i + 1)) ⊆ bins ?bs'*
proof –
have *Scan_F k ω (bins-upto ?bs' k (i + 1)) = Scan_F k ω (bins-upto ?bs' k i ∪ {items (?bs' ! k) ! i})*
using *bins-upto-Suc-Un Scan_F.hypos(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 ω (bins-upto bs k i ∪ {x})*
using *Scan_F.hypos(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 *... ⊆ bins bs ∪ Scan_F k ω {x}*
using *Scan_F.prems(2,3) Scan_F-Un Scan_F-Earley_F-bin-step-mono by fastforce*
finally have **: Scan_F k ω (bins-upto ?bs' k (i + 1)) ⊆ bins bs ∪ Scan_F k ω {x}*.
show *?thesis*
proof cases
assume *a1: ω!k = a*
hence *Scan_F k ω {x} = {inc-item x (k+1)}*
using *Scan_F.hypos(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 ω (bins-upto ?bs' k (i + 1)) ⊆ bins bs ∪ {inc-item x (k+1)}*
using ** by blast*
also have *... = bins bs ∪ set (items (Scan_L k ω a x i))*
using *a1 Scan_F.hypos(5) by (auto simp: Scan_L-def items-def)*
also have *... = bins ?bs'*
using *Scan_F.hypos(5) Scan_F.prems(1) wf-earley-input-elim bins-upd-bins by*
(*metis add-mono1*)
finally show *?thesis* .
next
assume *a1: ¬ ω!k = a*
hence *Scan_F k ω {x} = {}*
using *Scan_F.hypos(3) by (auto simp: Scan_F-def bin-def)*
hence *Scan_F k ω (bins-upto ?bs' k (i + 1)) ⊆ bins bs*
using ** by blast*
also have *... ⊆ bins ?bs'*
using *Scan_F.hypos(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 G (bins-upto ?bs' k (i + 1)) ⊆ bins ?bs'*
proof –

have $\text{Predict}_F k \mathcal{G} (\text{bins-upto } ?bs' k (i + 1)) = \text{Predict}_F k \mathcal{G} (\text{bins-upto } ?bs' k i \cup \{\text{items } (?bs' ! k) ! i\})$
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 $\dots = \text{Predict}_F k \mathcal{G} (\text{bins-upto } bs k i \cup \{x\})$
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-strict-right-mono items-def le-add1 length-map less-add-one linorder-not-le*)
also have $\dots \subseteq \text{bins } bs \cup \text{Predict}_F k \mathcal{G} \{x\}$
using *Scan_F.prems(2,3)* *Predict_F-Un* *Predict_F-Earley_F-bin-step-mono* **by** *fastforce*
also have $\dots = \text{bins } bs$
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 $\text{Complete}_F k (\text{bins-upto } ?bs' k (i + 1)) \subseteq \text{bins } ?bs'$
proof –
have $\text{Complete}_F k (\text{bins-upto } ?bs' k (i + 1)) = \text{Complete}_F k (\text{bins-upto } ?bs' k i \cup \{\text{items } (?bs' ! k) ! i\})$
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 $\dots = \text{Complete}_F k (\text{bins-upto } bs k i \cup \{x\})$
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-mono1 items-def length-map less-add-one linorder-not-le not-add-less1*)
also have $\dots = \text{Complete}_F k (\text{bins-upto } bs k i)$
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 $\text{Earley}_F\text{-bin-step } k \mathcal{G} \omega (\text{bins } ?bs') \subseteq \text{bins } (\text{Earley}_L\text{-bin}' k \mathcal{G} \omega ?bs' (i+1))$
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.cobounded1 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 ScanF k  $\omega$  (bins-upto bs k (i + 1))  $\subseteq$  bins bs
  using ScanF-def Pass.hyps(5) by auto
moreover have PredictF k  $\mathcal{G}$  (bins-upto bs k (i + 1))  $\subseteq$  bins bs
proof –
  have PredictF k  $\mathcal{G}$  (bins-upto bs k (i + 1)) = PredictF k  $\mathcal{G}$  (bins-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 ... = PredictF k  $\mathcal{G}$  (bins-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  $\cup$  PredictF k  $\mathcal{G}$  {x}
  using Pass.prems(2) PredictF-Un PredictF-EarleyF-bin-step-mono by blast
  also have ... = bins bs
  using Pass.hyps(3,4) Pass.prems(1) wf-earley-input-elim
  apply (auto simp: PredictF-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 CompleteF k (bins-upto bs k (i + 1))  $\subseteq$  bins bs
proof –
  have CompleteF k (bins-upto bs k (i + 1)) = CompleteF k (bins-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 ... = CompleteF k (bins-upto bs k i)
  using CompleteF-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) CompleteF-EarleyF-bin-step-mono wf-earley-input-elim
by blast
qed
ultimately have EarleyF-bin-step k  $\mathcal{G}$   $\omega$  (bins bs)  $\subseteq$  bins (EarleyL-bin' k  $\mathcal{G}$   $\omega$ 
bs (i+1))
  using Pass.IH Pass.prems EarleyF-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 (PredictF k  $\mathcal{G}$   $\omega$  bs i x a)
let ?bs' = upd-bins bs k (PredictL k  $\mathcal{G}$  a)
have k  $\geq$  length  $\omega$   $\vee$   $\neg$   $\omega$  ! k = a
  using PredictF.hyps(4) PredictF.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 $sound: \forall x \in set (items(Predict_L k \mathcal{G} a)). sound-item \mathcal{G} \omega x$
using $sound-Predict_L Predict_F.hyps(3) Predict_F.premis wf-earley-input-elim$
wf-bins-impl-wf-items **by** *fast*
have $wf: (k, \mathcal{G}, \omega, ?bs') \in wf-earley-input$
using $Predict_F.hyps Predict_F.premis(1) wf-earley-input-Predict_L$ **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 $Scan_F k \omega (bins-upto ?bs' k (i + 1)) \subseteq bins ?bs'$
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 $Predict_F.hyps(1) bins-upto-Suc-Un$ **by** (*metis items-def len length-map*)
also have $\dots = Scan_F k \omega (bins-upto bs k i \cup \{x\})$
using $Predict_F.hyps(1,2) Predict_F.premis(1) items-nth-idem-upd-bins bins-upto-kth-nth-idem$
wf-earley-input-elim
by (*metis dual-order.refl items-def length-map not-le-imp-less*)
also have $\dots \subseteq bins bs \cup Scan_F k \omega \{x\}$
using $Predict_F.premis(2,3) Scan_F-Un Scan_F-Earley_F-bin-step-mono$ **by** *fastforce*
also have $\dots = bins bs$
using $Predict_F.hyps(3) \langle length \omega \leq k \vee \omega ! k \neq a \rangle$ **by** (*auto simp: Scan_F-def bin-def*)
finally show *?thesis*
using $Predict_F.premis(1) wf-earley-input-elim bins-upd-bins$ **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 $\dots = Predict_F k \mathcal{G} (bins-upto bs k i \cup \{x\})$
using $Predict_F.hyps(1,2) Predict_F.premis(1) items-nth-idem-upd-bins bins-upto-kth-nth-idem$
wf-earley-input-elim
by (*metis dual-order.refl items-def length-map not-le-imp-less*)
also have $\dots \subseteq bins bs \cup Predict_F k \mathcal{G} \{x\}$
using $Predict_F.premis(2,3) Predict_F-Un Predict_F-Earley_F-bin-step-mono$ **by**
fastforce
also have $\dots = bins bs \cup set (items (Predict_L k \mathcal{G} a))$
using $Predict_F.hyps Predict_F.premis(1-3) wf-earley-input-elim$
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 $Predict_F.premis(1) wf-earley-input-elim bins-upd-bins$ **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 $(k, \mathcal{G}, \omega, bs) \in wf\text{-earley-input}$
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 $(k, \mathcal{G}, \omega, as) \in wf\text{-earley-input}$

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.prem(1,2) unfolding bins-eq-items-def*

by (*metis Earley_L-bin'-sims(1) length-map nth-map wf-earley-input-elim*)

ultimately show *?case*

using *Base.prem(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 *$k < \text{length } as$*

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 $(k, \mathcal{G}, \omega, ?as') \in wf\text{-earley-input}$

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'-sims(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 $(k, \mathcal{G}, \omega, ?as') \in wf\text{-earley-input}$

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* \leq *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: EarleyL-bin'-induct[OF assms(1), case-names
Base CompleteF ScanF Pass PredictF])
  case (CompleteF k G ω bs i x)
  let ?bs' = upd-bins bs k (CompleteL k x bs i)
  have x: x ∈ set (items (bs ! k))
    using CompleteF.hyps(1,2) by auto
  have wf: (k, G, ω, ?bs') ∈ wf-earley-input
    using CompleteF.hyps CompleteF.prems(1) wf-earley-input-CompleteL by blast
  hence ∀ x ∈ set (items (CompleteL k x bs i)). sound-item G ω x
    using sound-CompleteL CompleteF.hyps(3) CompleteF.prems wf-earley-input-elim
wf-bins-impl-wf-items x
    by (metis dual-order.refl)
  hence sound: ∀ x ∈ bins ?bs'. sound-item G ω x
    by (metis CompleteF.prems(1,3) UnE bins-upd-bins wf-earley-input-elim)
  show ?case
proof cases
  assume i+1 ≤ j
  thus ?thesis
    using wf sound CompleteF EarleyL-bin'-simps(2) by metis
next
  assume ¬ i+1 ≤ j
  hence i = j
    using CompleteF.prems(2) by simp
  have bins (EarleyL-bin' k G ω (EarleyL-bin' k G ω bs i) j) = bins (EarleyL-bin'
k G ω (EarleyL-bin' k G ω ?bs' (i+1)) j)
    using EarleyL-bin'-simps(2) CompleteF.hyps(1-3) by simp
  also have ... = bins (EarleyL-bin' k G ω (EarleyL-bin' k G ω ?bs' (i+1))
(j+1))
proof –
  let ?bs'' = EarleyL-bin' k G ω ?bs' (i+1)
  have length (items (?bs'' ! k)) ≥ length (items (bs ! k))
    using length-nth-bin-EarleyL-bin' length-nth-upd-bin-bins order-trans wf
CompleteF.hyps CompleteF.prems(1)
    by (smt (verit, ccfv-threshold) EarleyL-bin'-simps(2))
  hence 0: ¬ length (items (?bs'' ! k)) ≤ j
    using ⟨i = j⟩ CompleteF.hyps(1) by linarith
  have x = items (?bs' ! k) ! j
    using ⟨i = j⟩ items-nth-idem-upd-bins CompleteF.hyps(1,2)
    by (metis items-def length-map not-le-imp-less)
  hence 1: x = items (?bs'' ! k) ! j
    using ⟨i = j⟩ kth-EarleyL-bin'-bins CompleteF.hyps CompleteF.prems(1)
EarleyL-bin'-simps(2) leI by metis
  have bins (EarleyL-bin' k G ω ?bs'' j) = bins (EarleyL-bin' k G ω (upd-bins
?bs'' k (CompleteL k x ?bs'' i)) (j+1))
    using EarleyL-bin'-simps(2) 0 1 CompleteF.hyps(1,3) CompleteF.prems(2)
⟨i = j⟩ by auto
  moreover have bins-eq-items (upd-bins ?bs'' k (CompleteL k x ?bs'' i)) ?bs''
proof –

```

have $k < \text{length } bs$
using $\text{Complete}_F.\text{prems}(1)$ *wf-earley-input-elim* **by** *blast*
have $0: \text{set } (\text{Complete}_L k x bs i) = \text{set } (\text{Complete}_L k x ?bs'' i)$
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 $\text{Complete}_F.\text{prems}(1)$ *wf-bins-kth-bin wf-item-def nat-less-le* **by**
(*metis wf-earley-input-elim*)
hence $bs ! \text{start-item } x = ?bs'' ! \text{start-item } x$
using *False nth-idem-upd-bins nth-Earley_L-bin'-eq wf* **by** *metis*
thus *?thesis*
using *Complete_L-eq-start-item* **by** *metis*
qed
have $\text{set } (\text{items } (\text{Complete}_L k x bs i)) \subseteq \text{set } (\text{items } (?bs' ! k))$
by (*simp add: <k < length bs> upd-bins-def set-items-upds-bin*)
hence $\text{set } (\text{items } (\text{Complete}_L k x ?bs'' i)) \subseteq \text{set } (\text{items } (?bs' ! k))$
using 0 **by** (*simp add: items-def*)
also have $\dots \subseteq \text{set } (\text{items } (?bs'' ! k))$
by (*simp add: wf nth-bin-sub-Earley_L-bin'*)
finally show *?thesis*
using *bins-eq-items-upd-bins* **by** *blast*
qed
moreover have $(k, \mathcal{G}, \omega, \text{upd-bins } ?bs'' k (\text{Complete}_L k x ?bs'' i)) \in$
wf-earley-input
using *wf-earley-input-Earley_L-bin' wf-earley-input-Complete_L Complete_F.hyps*
Complete_F.prems(1)
 $\langle \text{length } (\text{items } (bs ! k)) \leq \text{length } (\text{items } (?bs'' ! k)) \rangle$ *kth-Earley_L-bin'-bins*
 $0 1$ **by** *blast*
ultimately show *?thesis*
using *Earley_L-bin'-bins-eq bins-eq-items-imp-eq-bins wf-earley-input-elim* **by**
blast
qed
also have $\dots = \text{bins } (\text{Earley}_L\text{-bin}' k \mathcal{G} \omega ?bs' (i + 1))$
using $\text{Complete}_F.\text{IH}[OF \text{ wf - sound } \text{Complete}_F.\text{prems}(4)] \langle i = j \rangle$ **by** *blast*
finally show *?thesis*
using $\text{Complete}_F.\text{hyps}$ **by** *simp*
qed
next
case $(\text{Scan}_F k \mathcal{G} \omega bs i x a)$
let $?bs' = \text{upd-bins } bs (k+1) (\text{Scan}_L k \omega a x i)$
have $x: x \in \text{set } (\text{items } (bs ! k))$
using $\text{Scan}_F.\text{hyps}(1,2)$ **by** *auto*
hence $\forall x \in \text{set } (\text{items } (\text{Scan}_L k \omega a x i)). \text{sound-item } \mathcal{G} \omega x$

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: $\forall x \in \text{bins } ?bs'. \text{sound-item } \mathcal{G} \omega x$*
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: $(k, \mathcal{G}, \omega, ?bs') \in \text{wf-earley-input}$*
using *Scan_F.hyps Scan_F.prems(1) wf-earley-input-Scan_L* **by** *metis*
show *?case*
proof *cases*
assume $i+1 \leq j$
thus *?thesis*
using *sound Scan_F* **by** (*metis Earley_L-bin'-simps(3) wf-earley-input-Scan_L*)
next
assume $\neg i+1 \leq j$
hence $i = j$
using *Scan_F.prems(2)* **by** *auto*
have *bins (Earley_L-bin' k $\mathcal{G} \omega$ (Earley_L-bin' k $\mathcal{G} \omega$ bs i) j) = bins (Earley_L-bin' k $\mathcal{G} \omega$ (Earley_L-bin' k $\mathcal{G} \omega$?bs' (i+1)) j)*
using *Scan_F.hyps* **by** *simp*
also have $\dots = \text{bins (Earley}_L\text{-bin' k } \mathcal{G} \omega \text{ (Earley}_L\text{-bin' k } \mathcal{G} \omega \text{ ?bs' (i+1)) (j+1))}$
proof $-$
let $?bs'' = \text{Earley}_L\text{-bin' k } \mathcal{G} \omega \text{ ?bs' (i+1)}$
have $\text{length (items (?bs'' ! k))} \geq \text{length (items (bs ! k))}$
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 $\mathcal{G} \omega$?bs'' j) = bins (Earley_L-bin' k $\mathcal{G} \omega$ (upd-bins ?bs'' (k+1) (Scan_L k ω a x i) (j+1))*
using $\langle i = j \rangle$ *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 $k+1 < \text{length } bs$
using *Scan_F.hyps(5) Scan_F.prems wf-earley-input-elim* **by** *fastforce+*
hence $\text{set (items (Scan}_L\text{ k } \omega \text{ a x i))} \subseteq \text{set (items (?bs' ! (k+1)))}$
by (*simp add: upd-bins-def set-items-upds-bin*)
also have $\dots \subseteq \text{set (items (?bs'' ! (k+1)))}$
using *wf nth-bin-sub-Earley_L-bin'* **by** *blast*
finally show *?thesis*
using *bins-eq-items-upd-bins* **by** *blast*
qed
moreover have $(k, \mathcal{G}, \omega, \text{upd-bins ?bs'' (k+1) (Scan}_L\text{ k } \omega \text{ a x i)}) \in$
wf-earley-input
using *wf-earley-input-Earley_L-bin' wf-earley-input-Scan_L Scan_F.hyps Scan_F.prems(1)*
 $\langle \text{length (items (bs ! k))} \leq \text{length (items (?bs'' ! k))} \rangle$ *kth-Earley_L-bin'-bins*
by (*smt (verit, ccfv-SIG) Earley_L-bin'-simps(3) linorder-not-le order.trans*)
ultimately show *?thesis*

```

    using EarleyL-bin'-bins-eq bins-eq-items-imp-eq-bins wf-earley-input-elim by
blast
  qed
  also have ... = bins (EarleyL-bin' k  $\mathcal{G}$   $\omega$  ?bs' (i + 1))
    using ⟨i = j⟩ ScanF.IH ScanF.prems ScanF.hyps sound wf-earley-input-ScanL
by fast
  finally show ?thesis
    using ScanF.hyps by simp
  qed
next
case (Pass k  $\mathcal{G}$   $\omega$  bs i x a)
show ?case
proof cases
  assume i+1 ≤ j
  thus ?thesis
    using Pass by (metis EarleyL-bin'-simps(4))
next
  assume ¬ i+1 ≤ j
  show ?thesis
    using Pass EarleyL-bin'-simps(1,4) kth-EarleyL-bin'-bins by (metis Suc-eq-plus1
Suc-leI antisym-conv2 not-le-imp-less)
  qed
next
case (PredictF k  $\mathcal{G}$   $\omega$  bs i x a)
let ?bs' = upd-bins bs k (PredictL k  $\mathcal{G}$  a)
have x: x ∈ set (items (bs ! k))
  using PredictF.hyps(1,2) by auto
hence ∀ x ∈ set (items (PredictL k  $\mathcal{G}$  a)). sound-item  $\mathcal{G}$   $\omega$  x
  using sound-PredictL PredictF.hyps(3) PredictF.prems wf-earley-input-elim
wf-bins-impl-wf-items by fast
hence sound: ∀ x ∈ bins ?bs'. sound-item  $\mathcal{G}$   $\omega$  x
  using PredictF.prems(1,3) UnE bins-upd-bins wf-earley-input-elim by metis
have wf: (k,  $\mathcal{G}$ ,  $\omega$ , ?bs') ∈ wf-earley-input
  using PredictF.hyps PredictF.prems(1) wf-earley-input-PredictL by metis
have len: i < length (items (?bs' ! k))
  using length-nth-upd-bin-bins PredictF.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 ≤ j
  thus ?thesis
    using sound wf PredictF by (metis EarleyL-bin'-simps(5))
next
  assume ¬ i+1 ≤ j
  hence i = j
    using PredictF.prems(2) by auto
  have bins (EarleyL-bin' k  $\mathcal{G}$   $\omega$  (EarleyL-bin' k  $\mathcal{G}$   $\omega$  bs i) j) = bins (EarleyL-bin'
k  $\mathcal{G}$   $\omega$  (EarleyL-bin' k  $\mathcal{G}$   $\omega$  ?bs' (i+1)) j)

```

using *Predict_F.hyps* **by** *simp*
also have ... = *bins* (*Earley_L-bin'* *k* *G* ω (*Earley_L-bin'* *k* *G* ω *?bs'* (*i+1*))
(*j+1*))
proof –
let *?bs''* = *Earley_L-bin'* *k* *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*
by (*metis* (*no-types*, *lifting*) *items-def* *length-map*)
hence *bins* (*Earley_L-bin'* *k* *G* ω *?bs''* *j*) = *bins* (*Earley_L-bin'* *k* *G* ω (*upd-bins*
?bs'' *k* (*Predict_L* *k* *G* *a*)) (*j+1*))
using $\langle i = j \rangle$ *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* *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* *G* *a*)) \subseteq *set* (*items* (*?bs'* ! *k*))
by (*simp* *add: upd-bins-def* *set-items-upds-bin*)
also have ... \subseteq *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*, *G*, ω , *upd-bins* *?bs''* *k* (*Predict_L* *k* *G* *a*)) \in *wf-earley-input*
using *wf-earley-input-Earley_L-bin'* *wf-earley-input-Predict_L* *Predict_F.hyps*
Predict_F.prems(1)
 \langle *length* (*items* (*bs* ! *k*)) \leq *length* (*items* (*?bs''* ! *k*)) \rangle *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* *G* ω *?bs'* (*i + 1*))
using $\langle i = j \rangle$ *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*, *G*, ω , *bs*) \in *wf-earley-input*
assumes $\forall x \in$ *bins* *bs*. *sound-item* *G* ω *x* *nonempty-derives* *G*
shows *bins* (*Earley_L-bin* *k* *G* ω (*Earley_L-bin* *k* *G* ω *bs*)) = *bins* (*Earley_L-bin* *k*
G ω *bs*)
using *assms* *Earley_L-bin'-idem* *Earley_L-bin-def* *le0* **by** *metis*

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

assumes $(k, \mathcal{G}, \omega, bs) \in wf\text{-earley-input}$
assumes $Earley_F\text{-bin-step } k \mathcal{G} \omega (bins\text{-upto } bs \ k \ 0) \subseteq bins \ bs \ \forall x \in bins \ bs.$
sound-item $\mathcal{G} \ \omega \ x$
assumes *is-word* $\mathcal{G} \ \omega \ nonempty\text{-derives } \mathcal{G}$
shows $funpower (Earley_F\text{-bin-step } k \ \mathcal{G} \ \omega) \ n (bins \ bs) \subseteq bins (Earley_L\text{-bin } k \ \mathcal{G} \ \omega \ bs)$
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 $0: Earley_F\text{-bin-step } k \ \mathcal{G} \ \omega (bins\text{-upto } (Earley_L\text{-bin } k \ \mathcal{G} \ \omega \ bs) \ k \ 0) \subseteq bins (Earley_L\text{-bin } k \ \mathcal{G} \ \omega \ bs)$
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 $funpower (Earley_F\text{-bin-step } k \ \mathcal{G} \ \omega) (Suc \ n) (bins \ bs) \subseteq Earley_F\text{-bin-step } k \ \mathcal{G} \ \omega (bins (Earley_L\text{-bin } k \ \mathcal{G} \ \omega \ bs))$
using *Earley_F-bin-step-sub-mono Suc* **by** (*metis funpower.simps(2)*)
also have $\dots \subseteq bins (Earley_L\text{-bin } k \ \mathcal{G} \ \omega (Earley_L\text{-bin } k \ \mathcal{G} \ \omega \ bs))$
using *Earley_F-bin-step-sub-Earley_L-bin Suc.premis wf-bins-Earley_L-bin sound-Earley_L-bin 0 wf-earley-input-Earley_L-bin* **by** *blast*
also have $\dots \subseteq bins (Earley_L\text{-bin } k \ \mathcal{G} \ \omega \ bs)$
using *Earley_L-bin-idem Suc.premis* **by** *blast*
finally show *?case* .
qed

lemma *Earley_F-bin-sub-Earley_L-bin:*
assumes $(k, \mathcal{G}, \omega, bs) \in wf\text{-earley-input}$
assumes $Earley_F\text{-bin-step } k \ \mathcal{G} \ \omega (bins\text{-upto } bs \ k \ 0) \subseteq bins \ bs \ \forall x \in bins \ bs.$
sound-item $\mathcal{G} \ \omega \ x$
assumes *is-word* $\mathcal{G} \ \omega \ nonempty\text{-derives } \mathcal{G}$
shows $Earley_F\text{-bin } k \ \mathcal{G} \ \omega (bins \ bs) \subseteq bins (Earley_L\text{-bin } k \ \mathcal{G} \ \omega \ bs)$
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 $k \leq length \ \omega$
assumes *is-word* $\mathcal{G} \ \omega \ nonempty\text{-derives } \mathcal{G}$
shows $Earley_F\text{-bins } k \ \mathcal{G} \ \omega \subseteq bins (Earley_L\text{-bins } k \ \mathcal{G} \ \omega)$
using *assms*
proof (*induction* k)
case 0
hence $Earley_F\text{-bin } 0 \ \mathcal{G} \ \omega (Init_F \ \mathcal{G}) \subseteq bins (Earley_L\text{-bin } 0 \ \mathcal{G} \ \omega (Init_L \ \mathcal{G} \ \omega))$
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, G, ω, Earley_L-bins k G ω) ∈ wf-earley-input*
by (*simp add: Suc.premis(1) Suc-leD assms(2) wf-earley-input-intro*)
have *sub: Earley_F-bin-step (Suc k) G ω (bins-upto (Earley_L-bins k G ω) (Suc k)*
0) ⊆ bins (Earley_L-bins k G ω)
proof –
have *bin (bins-upto (Earley_L-bins k G ω) (Suc k) 0) (Suc k) = {}*
using *kth-bin-bins-upto-empty wf Suc.premis wf-earley-input-elim* **by** *blast*
hence *Earley_F-bin-step (Suc k) G ω (bins-upto (Earley_L-bins k G ω) (Suc k)*
0) = bins-upto (Earley_L-bins k 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 *... ⊆ bins (Earley_L-bins k G ω)*
using *wf Suc.premis bins-upto-sub-bins wf-earley-input-elim* **by** *blast*
finally show *?thesis .*
qed
have *sound: ∀ x ∈ bins (Earley_L-bins k G ω). sound-item 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) G ω ⊆ Earley_F-bin (Suc k) G ω (bins (Earley_L-bins*
k G ω))
using *Suc Earley_F-bin-sub-mono* **by** *simp*
also have *... ⊆ bins (Earley_L-bin (Suc k) G ω (Earley_L-bins k G ω))*
using *Earley_F-bin-sub-Earley_L-bin wf sub sound Suc.premis* **by** *fastforce*
finally show *?case*
by *simp*
qed

lemma *Earley_F-sub-Earley_L:*
assumes *is-word G ω ε-free G*
shows *Earley_F G ω ⊆ bins (Earley_L 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 *G ⊢ [G] ⇒* ω is-word G ω ε-free G*
shows *recognizing (bins (Earley_L G ω)) 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* $\mathcal{G} \ \omega \ \varepsilon$ -free \mathcal{G}
shows *Earley* $\mathcal{G} \ \omega = \text{bins} (\text{Earley}_L \ \mathcal{G} \ \omega)$
using *assms* *Earley_F-sub-Earley_L* *Earley_L-sub-Earley_F* *Earley-eq-Earley_F* **by**
blast

lemma *correctness-recognizer*:

assumes *is-word* $\mathcal{G} \ \omega \ \varepsilon$ -free \mathcal{G}
shows *recognizer* $\mathcal{G} \ \omega \longleftrightarrow \mathcal{G} \vdash [\mathfrak{S} \ \mathcal{G}] \Rightarrow^* \omega$ (**is** $?L \longleftrightarrow ?R$)
proof *standard*
assume $?L$
then obtain x **where** $x \in \text{set} (\text{items} (\text{Earley}_L \ \mathcal{G} \ \omega \ ! \ \text{length} \ \omega))$ *is-finished* $\mathcal{G} \ \omega$
 x
using *assms*(1) **unfolding** *recognizer-def* **by** *blast*
moreover have $x \in \text{bins} (\text{Earley}_L \ \mathcal{G} \ \omega)$
using *assms*(2) *kth-bin-sub-bins* $\langle x \in \text{set} (\text{items} (\text{Earley}_L \ \mathcal{G} \ \omega \ ! \ \text{length} \ \omega)) \rangle$
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* :: $'a \ \text{item} \Rightarrow \text{bool}$ **where**

predicts $x \equiv \text{start-item} \ x = \text{end-item} \ x \wedge \text{dot-item} \ x = 0$

definition *scans* :: $'a \ \text{list} \Rightarrow \text{nat} \Rightarrow 'a \ \text{item} \Rightarrow 'a \ \text{item} \Rightarrow \text{bool}$ **where**

scans $\omega \ k \ x \ y \equiv y = \text{inc-item} \ x \ k \wedge (\exists a. \text{next-symbol} \ x = \text{Some} \ a \wedge \omega!(k-1) = a)$

definition *completes* :: $\text{nat} \Rightarrow 'a \ \text{item} \Rightarrow 'a \ \text{item} \Rightarrow 'a \ \text{item} \Rightarrow \text{bool}$ **where**

completes $k \ x \ y \ z \equiv y = \text{inc-item} \ x \ k \wedge \text{is-complete} \ z \wedge \text{start-item} \ z = \text{end-item} \ x \wedge$
 $(\exists N. \text{next-symbol} \ x = \text{Some} \ N \wedge N = \text{lhs-item} \ z)$

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

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

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

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

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

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

lemma *upd-bin-append*:
fst e ∉ set (items es) ⟹ 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 ∈ set (items es) ⟹ snd e = Null ∨ snd e = Pre pre ⟹ 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 $\overset{\#}{\neq}$ 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') ∧ (∀ j < length es'. i ≠ j ⟶ es!j = es!j) ∧ 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  $\exists x xp xs y yp ys. e = (x, \text{PreRed } xp \ xs) \wedge e' = (y, \text{PreRed } yp \ ys)$ )
      case True
      then obtain x xp xs y yp ys where ee':  $e = (x, \text{PreRed } xp \ xs) \ e' = (y, \text{PreRed } yp \ ys)$ 
      x = y
      using * by auto
      have simp:  $\text{upd-bin } e \ (e' \# \text{es}') = (x, \text{PreRed } yp \ (xp \# \text{xs} \ @ \ ys)) \# \text{es}'$ 
      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  $\text{upd-bin } e \ (e' \# \text{es}') = e' \# \text{es}'$ 
    using * by (auto split: pointer.splits prod.splits)
    thus ?thesis
    using False * Cons.prem(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:  $\text{upd-bin } e \ (e' \# \text{es}) = e' \# \text{upd-bin } e \ \text{es}$ 
  using * by (auto split: pointer.splits prod.splits)
  have 0: distinct (items es)
  using Cons.prem(1) unfolding items-def by simp
  have 1:  $i-1 < \text{length } \text{es}$ 
  using Cons.prem(2,3) * by (metis One-nat-def leI less-diff-conv2 less-one list.size(4) nth-Cons-0)
  have 2:  $\text{fst } e = \text{fst } (\text{es}!(i-1))$ 
  using Cons.prem(3) * by (metis nth-Cons')
  have 3:  $\text{snd } e = \text{PreRed } p \ rs$ 
  using Cons.prem(4) by simp
  have 4:  $\text{snd } (\text{es}!(i-1)) = \text{PreRed } p' \ rs'$ 
  using Cons.prem(3,5) * by (metis nth-Cons')
  have  $\text{snd } (\text{upd-bin } e \ \text{es}!(i-1)) = \text{PreRed } p' \ (p \# \text{rs} \ @ \ \text{rs}') \wedge$ 
   $(\forall j < \text{length } (\text{upd-bin } e \ \text{es}). i-1 \neq j \longrightarrow (\text{upd-bin } e \ \text{es}) ! j = \text{es} ! j)$ 
  using Cons.IH[OF 0 1 2 3 4] by blast
  hence  $\text{snd } ((e' \# \text{upd-bin } e \ \text{es}) ! i) = \text{PreRed } p' \ (p \# \text{rs} \ @ \ \text{rs}') \wedge$ 
   $(\forall j < \text{length } (e' \# \text{upd-bin } e \ \text{es}). i \neq j \longrightarrow (e' \# \text{upd-bin } e \ \text{es}) ! j = (e' \# \text{es}) ! j)$ 
  using * Cons.prem(2,3) less-Suc-eq-0-disj by auto
  moreover have  $e' \# \text{upd-bin } e \ \text{es} = \text{es}'$ 
  using Cons.prem(6) simp by auto

```

```

ultimately show ?thesis
  by (metis 0 1 2 3 4 Cons.IH Cons.prem6 length-Cons)
qed
qed simp

lemma sound-ptrs-upd-bin:
  assumes sound-ptrs  $\omega$  bs k < length bs es = bs!k distinct (items es)
  assumes sound-null-ptr e sound-pre-ptr  $\omega$  bs k e sound-prered-ptr bs k e
  shows sound-ptrs  $\omega$  (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  $\omega$  ?bs idx elem  $\wedge$  sound-prered-ptr ?bs
  idx elem
  proof cases
    assume a2: idx = k
    have elem  $\in$  set es  $\implies$  sound-pre-ptr  $\omega$  bs idx elem
      using a0 a2 assms(1-3) sound-ptrs-def by blast
    hence pre-es: elem  $\in$  set es  $\implies$  sound-pre-ptr  $\omega$  ?bs idx elem
      using a2 unfolding sound-pre-ptr-def by force
    have elem = e  $\implies$  sound-pre-ptr  $\omega$  bs idx elem
      using a2 assms(6) by auto
    hence pre-e: elem = e  $\implies$  sound-pre-ptr  $\omega$  ?bs idx elem
      using a2 unfolding sound-pre-ptr-def by force
    have elem  $\in$  set es  $\implies$  sound-prered-ptr bs idx elem
      using a0 a2 assms(1-3) sound-ptrs-def by blast
    hence prered-es: elem  $\in$  set es  $\implies$  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  $\implies$  sound-prered-ptr bs idx elem
      using a2 assms(7) by auto
    hence prered-e: elem = e  $\implies$  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  $\nexists p' ps'. \text{snd } (es!i) = \text{PreRed } p' ps'$ 
    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  $\neg (\exists p' ps'. \text{snd } (es!i) = \text{PreRed } p' ps')$ 
    then obtain p' ps' where D:  $\text{snd } (es!i) = \text{PreRed } p' ps'$ 
    by blast
    hence 0:  $\text{snd } (\text{upd-bin } e \text{ es!i}) = \text{PreRed } p' (p\#ps@ps') \wedge (\forall j < \text{length } (\text{upd-bin } e \text{ es}). i \neq j \longrightarrow \text{upd-bin } e \text{ es!j} = \text{es!j})$ 
    using C assms(4) upd-bin-prered-upd by blast
    obtain j where 1:  $j < \text{length } es \wedge \text{elem} = \text{upd-bin } e \text{ es!j}$ 
    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:  $\text{snd } \text{elem} = \text{PreRed } p' (p\#ps@ps')$ 
      using 0 1 by blast
      have sound-null-ptr elem
      using a3 unfolding sound-null-ptr-def by simp
      moreover have sound-pre-ptr  $\omega$  ?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)
      fix P PS k' pre red
      assume a4:  $\text{snd } \text{elem} = \text{PreRed } P \text{ PS} \wedge (k', \text{pre}, \text{red}) \in \text{set } (P\#PS)$ 
      show  $k' < \text{idx} \wedge \text{pre} < \text{length } (\text{bs}[k := \text{upd-bin } e \text{ es!}k']) \wedge \text{red} < \text{length } (\text{bs}[k := \text{upd-bin } e \text{ es!}idx]) \wedge$ 
      completes idx (fst (bs[k := upd-bin e es!k'pre])) (fst elem) (fst (bs[k :=
upd-bin e es!idxred]))
    proof cases
      assume a5:  $(k', \text{pre}, \text{red}) \in \text{set } (p\#ps)$ 
      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) ∉ set (p#ps)
      hence a5: (k', pre, red) ∈ set (p'#ps')
      using a3 a4 by auto
      have k' < idx ∧ pre < length (bs!k') ∧ red < length (bs!idx) ∧
      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 ∈ 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 ≠ 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* $bs\ k < \text{length}\ bs\ es = bs!k\ \text{distinct}\ (\text{items}\ es)$
assumes $\forall k'\ pre\ red\ ps.\ \text{snd}\ e = \text{PreRed}\ (k',\ pre,\ red)\ ps \longrightarrow \text{red} < \text{length}\ es$
shows *mono-red-ptr* $(bs[k := \text{upd-bin}\ e\ es])$
unfolding *mono-red-ptr-def*
proof (*standard, standard*)
fix idx
let $?bs = bs[k := \text{upd-bin}\ e\ es]$
assume $a0: idx < \text{length}\ ?bs$
show $\forall i < \text{length}\ (?bs!idx).\ \forall k'\ pre\ red\ ps.\ \text{snd}\ (?bs!idx!i) = \text{PreRed}\ (k',\ pre,\ red)\ ps \longrightarrow \text{red} < i$
proof *cases*
assume $a1: idx=k$
consider $(A)\ \text{fst}\ e \notin \text{set}\ (\text{items}\ es) \mid$
 $(B)\ \text{fst}\ e \in \text{set}\ (\text{items}\ es) \wedge (\exists pre.\ \text{snd}\ e = \text{Null} \vee \text{snd}\ e = \text{Pre}\ pre) \mid$
 $(C)\ \text{fst}\ e \in \text{set}\ (\text{items}\ es) \wedge \neg (\exists pre.\ \text{snd}\ e = \text{Null} \vee \text{snd}\ e = \text{Pre}\ pre)$
by *blast*
thus *?thesis*
proof *cases*
case A
hence $\text{upd-bin}\ e\ es = es\ @\ [e]$
using *upd-bin-append* **by** *blast*
thus *?thesis*
using $a1\ \text{assms}(1-3,5)\ \text{mono-red-ptr-def}$
by (*metis length-append-singleton less-antisym nth-append nth-append-length nth-list-update-eq*)
next
case B
hence $\text{upd-bin}\ e\ es = es$
using *upd-bin-null-pre* **by** *blast*
thus *?thesis*
using $a1\ \text{assms}(1-3)\ \text{mono-red-ptr-def}$ **by** *force*
next
case C
then obtain $i\ p\ ps$ **where** $C: i < \text{length}\ es\ \text{fst}\ e = \text{fst}\ (es!i)\ \text{snd}\ e = \text{PreRed}\ p\ ps$
by (*metis in-set-conv-nth items-def length-map nth-map pointer.exhaust*)
show *?thesis*
proof *cases*
assume $\nexists p'\ ps'.\ \text{snd}\ (es!i) = \text{PreRed}\ p'\ ps'$
hence $\text{upd-bin}\ e\ es = es$
using *upd-bin-prered-nop* $C\ \text{assms}(4)$ **by** *blast*
thus *?thesis*
using $a1\ \text{assms}(1-3)\ \text{mono-red-ptr-def}$ **by** (*metis nth-list-update-eq*)
next
assume $\neg (\exists p'\ ps'.\ \text{snd}\ (es!i) = \text{PreRed}\ p'\ ps')$
then obtain $p'\ ps'$ **where** $D: \text{snd}\ (es!i) = \text{PreRed}\ p'\ ps'$
by *blast*
have $0: \text{snd}\ (\text{upd-bin}\ e\ es!i) = \text{PreRed}\ p'\ (p\#ps@ps') \wedge$

```

    (∀ j < length (upd-bin e es). i ≠ j → upd-bin e es!j = es!j) ∧
    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 < length (?bs!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 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≠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: idx≠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 < length bs b = bs!k distinct (items
  b)
  assumes ∀ e ∈ set es. sound-null-ptr e ∧ sound-pre-ptr ω bs k e ∧ sound-prered-ptr
  bs k e
  assumes ∀ e ∈ set es. ∀ k' pre red ps. snd e = PreRed (k', pre, red) ps → red
  < length b
  shows sound-ptrs ω (bs[k := upds-bin es b]) ∧ mono-red-ptr (bs[k := upds-bin es
  b])
  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.prem(1,3-5,6) by (metis list.set-intros(1))
    have 1: mono-red-ptr ?bs
      using mono-red-ptr-upd-bin Cons.prem(2-5,7) by (metis list.set-intros(1))
  
```

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

lemma *sound-mono-ptrs-Earley_L-bin'*:
assumes $(k, \mathcal{G}, \omega, bs) \in \text{wf-earley-input}$
assumes *sound-ptrs* $\omega \ bs \ \forall x \in \text{bins } bs. \text{sound-item } \mathcal{G} \ \omega \ x$
assumes *mono-red-ptr* bs
assumes *nonempty-derives* \mathcal{G}
shows *sound-ptrs* $\omega \ (\text{Earley}_L\text{-bin}' \ k \ \mathcal{G} \ \omega \ bs \ i) \wedge \text{mono-red-ptr } (\text{Earley}_L\text{-bin}' \ k \ \mathcal{G} \ \omega \ bs \ i)$
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 $?bs' = \text{upd-bins } bs \ k \ (\text{Complete}_L \ k \ x \ bs \ i)$
have $x \in \text{set } (\text{items } (bs \ ! \ k))$
using *Complete_F.hyps*(1,2) **by** *force*
hence $\forall x \in \text{set } (\text{items } (\text{Complete}_L \ k \ x \ bs \ i)). \text{sound-item } \mathcal{G} \ \omega \ x$
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*)
hence *sound*: $\forall x \in \text{bins } ?bs'. \text{sound-item } \mathcal{G} \ \omega \ x$
by (*metis* *Complete_F.prems*(1,3) *UnE bins-upd-bins wf-earley-input-elim*)
have 0: $k < \text{length } bs$
using *Complete_F.prems*(1) *wf-earley-input-elim* **by** *auto*
have 1: $\forall e \in \text{set } (\text{Complete}_L \ k \ x \ bs \ i). \text{sound-null-ptr } e$
unfolding *Complete_L-def sound-null-ptr-def* **by** *auto*
have 2: $\forall e \in \text{set } (\text{Complete}_L \ k \ x \ bs \ i). \text{sound-pre-ptr } \omega \ bs \ k \ e$
unfolding *Complete_L-def sound-pre-ptr-def* **by** *auto*
{
fix e

```

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

have fst  $e = \text{inc-item } (\text{fst } (\text{bs!k!pre})) \ k$ 
  using a0 a1 0 2 CompleteF.hyps(1,2,3) CompleteF.prems(1)  $\langle k' = \text{start-item}$ 
 $x \rangle$  unfolding CompleteL-def
  by (auto simp: items-def, metis filter-with-index-nth nth-map)
moreover have is-complete  $(\text{fst } (\text{bs!k!red}))$ 
  using a0 a1 0 2 CompleteF.hyps(1,2,3) CompleteF.prems(1)  $\langle k' = \text{start-item}$ 
 $x \rangle$  unfolding CompleteL-def
  by (auto simp: next-symbol-def items-def split: if-splits)
moreover have start-item  $(\text{fst } (\text{bs!k!red})) = \text{end-item } (\text{fst } (\text{bs!k!pre}))$ 
  using a0 a1 0 2 CompleteF.hyps(1,2,3) CompleteF.prems(1)  $\langle k' = \text{start-item}$ 
 $x \rangle$  unfolding CompleteL-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  $(\exists N. \text{next-symbol } (\text{fst } (\text{bs!k!pre})) = \text{Some } N \wedge N =$ 
lhs-item  $(\text{fst } (\text{bs!k!red})))$ 
  using a0 a1 0 2 CompleteF.hyps(1,2,3) CompleteF.prems(1)  $\langle k' = \text{start-item}$ 
 $x \rangle$  unfolding CompleteL-def
  by (auto simp: items-def, metis (mono-tags, lifting) filter-with-index-P fil-
ter-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.hypos(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*, *cfv-SIG*) *list.set-intros(1)*)
moreover **have** $(k, \mathcal{G}, \omega, ?bs') \in$ *wf-earley-input*
using *Complete_F.hypos* *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.hypos* **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.hypos(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.hypos(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.hypos(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.hypos(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.hypos(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*

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

lemma *sound-mono-ptrs-Earley_L-bin:*
assumes $(k, \mathcal{G}, \omega, bs) \in \text{wf-earley-input}$
assumes *sound-ptrs* ω bs $\forall x \in \text{bins } bs.$ *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 $k \leq \text{length } \omega$ *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 $(0, \mathcal{G}, \omega, (\text{Init}_L \mathcal{G} \omega)) \in \text{wf-earley-input}$

using *assms(2) wf-earley-input-Init_L* **by** *blast*

moreover have $\forall x \in \text{bins } (\text{Init}_L \mathcal{G} \omega). \text{sound-item } \mathcal{G} \omega x$

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.prem*

by *fastforce*

next

case (*Suc* k)

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

by (*simp add: Suc.prem(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 $\forall x \in \text{bins } (\text{Earley}_L\text{-bins } k \mathcal{G} \omega). \text{sound-item } \mathcal{G} \omega x$

by (*meson Suc.prem(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.prem 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 (ℳ \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*:

$k < \text{length } xs \implies \text{foldl } (+) z (\text{map length } (\text{take } k \text{ } xs)) + \text{length } (xs!k) = \text{foldl } (+) z (\text{map length } (\text{take } (k+1) \text{ } xs))$

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*:

$a \leq b \implies \text{foldl } (+) a \text{ } xs \leq \text{foldl } (+) b \text{ } xs$ **for** $a :: \text{nat}$

by (induction *xs* arbitrary: *a* *b*) auto

lemma *foldl-ge-z-nth*:

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

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 ... ≤ 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 ... ≤ 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:

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

proof (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 ... ≤ foldl (+) z (map length (take (j+1) (x#xs)))
      using foldl-ge-z-nth Cons.prem(2) by blast
    finally show ?thesis
      by blast
  next
  case False
  have i-1 ≤ j-1
    by (simp add: Cons.prem(1) diff-le-mono)
  have j-1 < length xs
    using Cons.prem(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.prem(1) False by auto
  also have ... ≤ foldl (+) (z + length x) (map length (take (j-1+1) xs))
    using Cons.IH ⟨i - 1 ≤ j - 1⟩ ⟨j - 1 < length xs⟩ by blast
  also have ... = foldl (+) (z + length x) (map length (take j xs))
    using Cons.prem(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 ← build-tree' bs ω k' pre;
    case t of
      Branch N ts ⇒
        do {
          t ← build-tree' bs ω k red;
          Some (Branch N (ts @ [t]))
        }
      | - ⇒ undefined — impossible case
    })
  ))

```

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

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

```

build-tree G ω bs = (
  let k = length bs - 1 in (
    case filter-with-index (λx. is-finished G ω x) (items (bs!k)) of
      [] ⇒ None
    | (-, i)#- ⇒ build-tree' bs ω k i
  ))

```

lemma *build-tree'-simps[simp]*:

```

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

```

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

```

wf-tree-input = {

```

```

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

```

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

lemma *wf-tree-input-pre*:
assumes (bs, ω, k, i) ∈ *wf-tree-input*
assumes e = bs!k!i snd e = *Pre pre*
shows (bs, ω, (k-1), pre) ∈ *wf-tree-input*
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, ω, k, i) ∈ *wf-tree-input*
assumes e = bs!k!i snd e = *PreRed (k', pre, red) ps*
shows (bs, ω, k', pre) ∈ *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, ω, k, i) ∈ *wf-tree-input*
assumes e = bs!k!i snd e = *PreRed (k', pre, red) ps*
shows (bs, ω, k, red) ∈ *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, ω, k, i) ∈ *wf-tree-input*
assumes $\bigwedge bs \ \omega \ k \ i.$
 $(\bigwedge e \ pre. \ e = bs!k!i \implies \text{snd } e = \text{Pre } pre \implies P \ bs \ \omega \ (k-1) \ pre) \implies$
 $(\bigwedge e \ k' \ pre \ red \ ps. \ e = bs!k!i \implies \text{snd } e = \text{PreRed } (k', \ pre, \ red) \ ps \implies P \ bs \ \omega$
 $k' \ pre) \implies$
 $(\bigwedge e \ k' \ pre \ red \ ps. \ e = bs!k!i \implies \text{snd } e = \text{PreRed } (k', \ pre, \ red) \ ps \implies P \ bs \ \omega$
 $k \ red) \implies$
 $P \ bs \ \omega \ k \ i$

```

shows  $P\ bs\ \omega\ k\ i$ 
using assms(1)
proof (induction  $n \equiv \text{build-tree}'\text{-measure}(bs, \omega, k, i)$  arbitrary:  $k\ i$  rule: nat-less-induct)
  case 1
  obtain  $e$  where entry:  $e = bs!k!i$ 
    by simp
  consider (Null) snd  $e = \text{Null}$ 
    | (Pre)  $\exists\ pre.$  snd  $e = \text{Pre}\ pre$ 
    | (PreRed)  $\exists\ k'\ pre\ red\ reds.$  snd  $e = \text{PreRed}(k', pre, red)\ reds$ 
    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 snd  $e = \text{Pre}\ pre$ 
    by blast
  define  $n$  where  $n = \text{build-tree}'\text{-measure}(bs, \omega, (k-1), pre)$ 
  have  $0 < k\ pre < \text{length}(bs!(k-1))$ 
  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  $k < \text{length}\ bs$ 
    using  $1(2)$  unfolding wf-tree-input-def by blast+
  have  $\text{foldl}(+) 0 (\text{map}\ \text{length}\ (\text{take}\ k\ bs)) + i - (\text{foldl}(+) 0 (\text{map}\ \text{length}\ (\text{take}\ (k-1)\ bs)) + pre) =$ 
     $\text{foldl}(+) 0 (\text{map}\ \text{length}\ (\text{take}\ (k-1)\ bs)) + \text{length}(bs!(k-1)) + i - (\text{foldl}(+) 0 (\text{map}\ \text{length}\ (\text{take}\ (k-1)\ bs)) + pre)$ 
    using foldl-add-nth[of  $\langle k-1 \rangle\ bs\ 0$ ] by (simp add:  $\langle 0 < k \rangle\ \langle k < \text{length}\ bs \rangle$ )
    less-imp-diff-less)
  also have  $\dots = \text{length}(bs!(k-1)) + i - pre$ 
    by simp
  also have  $\dots > 0$ 
    using  $\langle pre < \text{length}(bs!(k-1)) \rangle$  by auto
  finally have  $\text{build-tree}'\text{-measure}(bs, \omega, k, i) - \text{build-tree}'\text{-measure}(bs, \omega, (k-1), pre) > 0$ 
    by simp
  hence  $P\ bs\ \omega\ (k-1)\ pre$ 
    using  $1\ n\ wf\text{-tree}\text{-input}\text{-pre}\ \text{entry}\ pre\ \text{zero-less-diff}$  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  $e = \text{PreRed}(k', pre, red)\ ps$ 
    by blast
  have  $k' < k\ pre < \text{length}(bs!k')$ 
  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.prem1 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.prem1 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  $\omega$  (k-1) pre = Some (Branch N ts)
    using IH(1) entry pre by blast
  have build-tree' bs  $\omega$  k i = Some (Branch N (ts @ [Leaf ( $\omega$ !(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 prerred: snd e = PreRed (k', pre, red) ps
    by blast
  then obtain N ts where Nts: build-tree' bs  $\omega$  k' pre = Some (Branch N ts)
    using IH(2) entry prerred by blast
  obtain t-red where t-red: build-tree' bs  $\omega$  k red = Some t-red
    using IH(3) entry prerred Nts by (metis option.exhaust)
  have build-tree' bs  $\omega$  k i = Some (Branch N (ts @ [t-red]))
    using build-tree'-simps(8) entry prerred Nts t-red by auto
  thus ?thesis
    by blast
qed
done
thus ?thesis
  by blast
qed

```

```

lemma wf-item-tree-build-tree':
  assumes (bs,  $\omega$ , k, i)  $\in$  wf-tree-input
  assumes wf-bins  $\mathcal{G}$   $\omega$  bs
  assumes build-tree' bs  $\omega$  k i = Some t
  shows wf-item-tree  $\mathcal{G}$  (fst (bs!k!i)) t
proof -
  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  $\omega$  k i t
  proof -
    define e where entry: e = bs!k!i
    have bounds: k < length bs k  $\leq$  length  $\omega$  i < length (bs!k)
      using prems(4) wf-tree-input-def by force+

```

```

consider (Null) snd e = Null
  | (Pre)  $\exists$  pre. snd e = Pre pre
  | (PreRed)  $\exists$  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  $\omega$  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  $\omega$  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  $\omega$  (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  $\omega$  k i = Some (Branch N (ts @ [Leaf ( $\omega$ !(k-1))]))
    using build-tree'-simps(3) entry pre Nts by simp
  have sound-ptrs  $\omega$  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  $\mathcal{G}$  (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  $\omega$  k (fst (bs!(k-1)!pre)) (fst e)
    using entry pre bounds  $\langle$ sound-ptrs  $\omega$  bs $\rangle$  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 ( $\omega$ !(k-1))
  unfolding scans-def inc-item-def by (simp-all add: lhs-item-def rhs-item-def)
  have map root (ts @ [Leaf ( $\omega$ !(k-1))]) = map root ts @ [ $\omega$ !(k-1)]
    by simp
  also have ... = take (dot-item (fst (bs!(k-1)!pre))) (rhs-item (fst (bs!(k-1)!pre)))

```

```

@ [ω!(k-1)]
  using IH by simp
  also have ... = take (dot-item (fst (bs!(k-1)!pre))) (rhs-item (fst e)) @
[ω!(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 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 prerred: 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) prerred wf-tree-input-prerred-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) prerred wf-tree-input-prerred-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 prerred 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 prerred 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 prerred 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 prerred] by blast
  hence IH-pre: wf-item-tree G (fst (bs!k!pre)) (Branch N ts)
  using prems(2)[OF entry prerred - prems(5)] Nts bounds(1,2) order-less-trans
prems(6) by blast

```

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

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

```

apply (induction arbitrary: t rule: build-tree'-induct[OF assms(1)])
subgoal premises prems for bs  $\omega$  k i t
proof –
  define e where entry:  $e = bs!k!i$ 
  have bounds:  $k < \text{length } bs \ k \leq \text{length } \omega \ i < \text{length } (bs!k)$ 
    using prems(4) wf-tree-input-def by force+
  consider (Null) snd  $e = \text{Null}$ 
    | (Pre)  $\exists pre. \text{snd } e = \text{Pre } pre$ 
    | (PreRed)  $\exists k' pre \ red \ reds. \text{snd } e = \text{PreRed } (k', pre, red) \ reds$ 
    by (metis pointer.exhaust surj-pair)
  thus ?thesis
proof cases
  case Null
    hence build-tree' bs  $\omega$  k i = Some (Branch (lhs-item (fst e))) []
      using entry by simp
    have simp:  $t = \text{Branch } (\text{lhs-item } (\text{fst } e))$  []
      using build-tree'-simps(1) Null prems(6) entry by simp
    have sound-ptrs  $\omega$  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  $e = \text{Pre } pre$ 
      by blast
    obtain N ts where Nts: build-tree' bs  $\omega$  ( $k-1$ ) pre = Some (Branch N ts)
      using build-tree'-termination entry pre prems(4) wf-tree-input-pre by blast
    have simp: build-tree' bs  $\omega$  k i = Some (Branch N (ts @ [Leaf ( $\omega!(k-1)$ )]))
      using build-tree'-simps(3) entry pre Nts by simp
    have sound-ptrs  $\omega$  bs
      using prems(4) unfolding wf-tree-input-def by blast
    hence bounds':  $k > 0 \ pre < \text{length } (bs!(k-1))$ 
      using entry pre bounds unfolding sound-ptrs-def sound-pre-ptr-def by
(metis nth-mem)+
    moreover have  $k-1 < \text{length } bs$ 
      by (simp add: bounds less-imp-diff-less)
    ultimately have IH: wf-yield  $\omega$  (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  $\omega$  k (fst ( $bs!(k-1)!pre$ )) (fst e)
      using entry pre bounds  $\langle \text{sound-ptrs } \omega \ bs \rangle$  unfolding sound-ptrs-def
sound-pre-ptr-def by simp
    have wf:
      start-item (fst ( $bs!(k-1)!pre$ ))  $\leq \text{end-item } (\text{fst } (bs!(k-1)!pre))$ 
      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  $\omega$  (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,  $\omega, k, red$ )  $\in$  wf-tree-input
  using wf-tree-input-prered-red[OF prems(4) entry prered] by blast
hence IH-r: wf-yield  $\omega$  (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  $\omega$  (start-item (fst (bs!k!pre))) (end-item (fst (bs!k!pre)))
@
  slice  $\omega$  (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  $\omega$  (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  $\omega$  (start-item (fst e)) (end-item (fst (bs!k!red)))
  using completes unfolding completes-def inc-item-def by simp
also have ... = slice  $\omega$  (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  $\mathcal{G}$   $\omega$  bs sound-ptrs  $\omega$  bs mono-red-ptr bs length bs = length  $\omega$  +
1
assumes build-tree  $\mathcal{G}$   $\omega$  bs = Some t
shows wf-rule-tree  $\mathcal{G}$  t  $\wedge$  root t =  $\mathfrak{S}$   $\mathcal{G}$   $\wedge$  yield t =  $\omega$ 
proof –
  let ?k = length bs – 1

```

```

define finished where finished-def: finished = filter-with-index (is-finished  $\mathcal{G}$   $\omega$ )
(items (bs!?k))
then obtain x i where *: (x, i)  $\in$  set finished Some t = build-tree' bs  $\omega$  ?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  $\leq$  length  $\omega$ 
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  $\mathcal{G}$   $\omega$  x
using * filter-with-index-P finished-def by metis
have wf-trees-input: (bs,  $\omega$ , ?k, i)  $\in$  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  $\mathcal{G}$   $\omega$  (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  $\exists r \in$  set ( $\mathfrak{R}$   $\mathcal{G}$ ). N = lhs-rule r  $\wedge$  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  $\langle N =$  lhs-item x  $\rangle$  by (auto simp: is-finished-def)
have yield t = slice  $\omega$  (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 =  $\omega$ 
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  $\varepsilon$ -free  $\mathcal{G}$ 
assumes build-tree  $\mathcal{G}$   $\omega$  (EarleyL  $\mathcal{G}$   $\omega$ ) = Some t
shows wf-rule-tree  $\mathcal{G}$  t  $\wedge$  root t =  $\mathfrak{S}$   $\mathcal{G}$   $\wedge$  yield t =  $\omega$ 
using assms wf-rule-root-yield-build-tree wf-bins-EarleyL sound-mono-ptrs-EarleyL
EarleyL-def
length-EarleyL-bins length-bins-InitL by (metis  $\varepsilon$ -free-impl-nonempty-derives
le-refl)

```

theorem *correctness-build-tree-Earley_L*:
assumes *is-word* \mathcal{G} ω ε -free \mathcal{G}
shows $(\exists t. \text{build-tree } \mathcal{G} \ \omega \ (\text{Earley}_L \ \mathcal{G} \ \omega) = \text{Some } t) \longleftrightarrow \mathcal{G} \vdash [\mathfrak{S} \ \mathcal{G}] \Rightarrow^* \omega \ (\text{is } ?L \longleftrightarrow ?R)$
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** *: $(x, i) \in \text{set finished}$ *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 ≤ 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* ∈ 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 $\exists x \in \text{set (items ((Earley}_L \ \mathcal{G} \ \omega)! ?k)). \text{is-finished } \mathcal{G} \ \omega \ x$
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 $(x, i) \in \text{set finished}$
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 (EarleyL  $\mathcal{G}$   $\omega$ ,  $\omega$ , ? $k$ ,  $i$ )  $\in$  wf-tree-input
unfolding wf-tree-input-def using sound-mono-ptrs-EarleyL assms  $\varepsilon$ -free-impl-nonempty-derives
using  $\langle$ length (EarleyL  $\mathcal{G}$   $\omega$ )  $- 1 =$  length  $\omega$  $\rangle$  by auto
then obtain  $N$   $ts$  where build-tree' (EarleyL  $\mathcal{G}$   $\omega$ )  $\omega$  ? $k$   $i =$  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 $S \rightarrow SS \mid a$

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

```

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

```

definition *cfg1* :: *symbol cfg* where

```
cfg1 = CFG rules1 S
```

lemma ε -free1:

```
 $\varepsilon$ -free cfg1
```

```
by (auto simp:  $\varepsilon$ -free-def cfg1-def rules1-def rhs-rule-def)
```

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

10.3.1 $S \rightarrow aS \mid a$

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

```

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

```

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

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

10.3.2 $S \rightarrow aSa \mid a$

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

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

lemma *ε-free3*:
ε-free cfg3
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 $S \rightarrow Sa \mid a$

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

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

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

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

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

]

definition *cfg5* :: *symbol cfg* **where**

cfg5 = *CFG rules5 S*

lemma *ε-free5*:

ε-free cfg5

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
]

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 (λb. fold (+) (map (λe. 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).