Marpa Correctness Proof

Jeffrey Kegler

June 28, 2010

Abstract

This is a "correctness" proof for the Marpa algorithm. "Correctness" is shown by proving equivalence to the Aycock & Horspool's modification of Earley's algorithm in their 2002 paper.

Contents

1 THIS IS A DRAFT

This paper is in late stages, but not yet finished. It will contain many errors of the kind that the last drafts should remove.

2 Introduction

The paper proves the equivalence of Marpa to Aycock and Horspool's 2002 revision of Earley's algorithm (AH2002). The intent is to establish confidence that the Marpa algorithm is correct. The reader is assumed to be familiar with standard grammar notation, LR(0) automata, Earley's algorithm and Aycock and Horspool's 2002 paper[1].

3 The Two Algorithms

In this paper, the logic of both the Marpa and the AH2002 algorithm is divided among five sections

- `Initializer`
- `Recognizer`
- `Earley Set Processing`

[1] Aycock and Horspool 2002

- `Scanner`
- `Completer`

The `Initializer`, `Recognizer`, `Scanner` and `Completer` are the same for both algorithms. `Marpa` and `AH2002` differ only in their `Earley Set Processing`.

Pseudocode is provided for `Recognizer`, `Scanner`, `Completer` and each algorithm's `Earley Set Processing`. All processing is deterministic. Variables are either global to the algorithm or local to their section.

Sections "call" other sections. When called, some sections take arguments, which become input variables in the called section. Input variables are local to the section, and their semantics is call-by-value. Despite the similarity to the semantics of subroutines in certain languages, no implementation is implied. Sections do not return a value.

4 Earley Sets

Each Earley set is an ordered list of Earley items. In AH2002 and Marpa, every Earley item is a `(state, parent)` duple, where `state` is an AHFA state. AHFA states are states of the split ϵ-DFA automatons first described by Aycock and Horspool. In this document a split ϵ-DFA automaton is called an AHFA (Aycock-Horspool Finite Automaton) and its states are called AHFA states. Details of AHFA's and their states are not relevant to this proof and may be found in Aycock and Horspool 2002.

In the `(state, parent)` Earley item duple, `parent` is a parent pointer. A parent pointer is a pointer to an Earley set, possibly the same one of which the Earley item is a member.

4.1 Worklists

In both the AH2002 and Marpa algorithms, the Earley sets need to be arrays that obey the semantics of worklists. Worklists are arrays which contain work items. Work items in a work list never disappear, unless the entire work list goes out of scope. Worklists only change in one way – by the addition of new work items at the end of the work list.

A dynamic array can be implemented so that is never changed except to grow it at the end. Such an array would provide worklist semantics.

4.2 AHFA States Versus LR(0) Items

The most visible difference between AH2002 and the standard Earley algorithm is in the form of the Earley items. In both AH2002 and the standard Earley algorithm, Earley items are duples. Also, in both AH2002 and the standard Earley algorithm, the second element of the duple is a parent pointer.

But the first element of the duple in AH2002 is not the same as in the standard Earley algorithm. In the standard Earley algorithm, the first element of the duple is an LR(0) **item**. In the AH2002 algorithm the first element of the Earley item is an AHFA **state**. An AHFA state is a **set** of one or more LR(0) items.

Working with sets of LR(0) items instead of single items makes the AH2002 algorithm more complex, but it also makes it much more efficient in practice. Earley items in the Marpa algorithm are the same as in the AH2002 algorithm – duples of AHFA state and parent pointer.

5 Initializer

In both Marpa and AH2002, the Initializer accepts a token stream and a grammar as input. The token stream is an array of `n` input tokens, `x[1], x[2], ..., x[n]`.

In each algorithm, the Recognizer Main Processing initializes `S[0]` with Earley items of the form `(start_state, 0)` where `start_state` is one of the AHFA start states and the parent pointer points to Earley set 0. There will be one or two start states, depending on the grammar.

5.1 Pure Functions of the Grammar

The Initializer creates the functions `COMPLETED`, `LHS` and `GOTO`. These are pure functions which depend only on the grammar. These are partial functions with a finite and, in fact, very limited domain. Since these functions depend only on the grammar, they may be precomputed. Since they are only defined for a fixed list of arguments, these functions may be implemented as table lookups.

`LHS` is a function from a rule to its left hand side symbol. `LHS` is defined for every rule of the grammar.

`COMPLETED` is a function from an AHFA state to a list of the rules which have completed LR(0) items in that AHFA state. A completed LR(0) item is one with the dot at the end. The list of rules returned by `COMPLETED` may be empty.

`GOTO` is a transition function from an AHFA state and a symbol (which may be undefined) to another AHFA state. Most AHFA states only have transitions on a few of the grammar's symbols, so for many combinations of `state` and `symbol`, the result of `GOTO(state, symbol)` will be undefined.

6 Recognizer

The Recognizer builds an array of `n+1` Earley sets, `S[0], S[1], ..., S[SIZE(x)]`, where `SIZE(x)` is the size of the token stream array, `x`. In both algorithms, the Earley sets need to be implemented so they can be treated as worklists.

In both Marpa and AH2002, the Recognizer Main Processing then does the Earley Set Processing. The Earley Set Processing is performed for each of the Earley sets from 0 to `SIZE(x)`, in order.

Algorithm 1 Recognizer

```
Recognizer Loop: for every (i) in (0 .. SIZE(x)) {
    do Earley Set i Processing
}
```

7 The AH2002 Algorithm

In the proofs, it is convenient to refer to a `Scanner Loop` and a `Completer Loop` for both `AH2002` and `Marpa`. `Marpa` has separate Scanner and Completer Loops defined in its Earley Set Processing pseudocode. For `AH2002`, both the `Scanner Loop` and the `Completer Loop` is defined as equivalent to the `Combined Loop`.

DEFINITION: `Scanner Loop @ AH2002 @ Earley Set i Processing` is defined as `Combined Loop @ AH2002 @ Earley Set i Processing`.

DEFINITION: `Completer Loop @ AH2002 @ Earley Set i Processing` is defined as `Combined Loop @ AH2002 @ Earley Set i Processing`.

Algorithm 2 AH2002 Earley Set Processing

```
Input variable: i

Combined Loop: for every (state, parent) in S[i] {
    do Scanner( state, parent )
    if parent != i {
       do Completer( state, parent )
    }
}
```

8 The Marpa Algorithm

The `Marpa` algorithm is the `AH2002` algorithm "upside-down and inside-out". It pulls the scanner and the completer out of the main loop and gives each its own loop. And `Marpa` reverses the order – the completer is called first to process all items in an Earley set, including those items added by the completer itself. After the completer is finished, the scanner is called to process all items in the Earley set. The scanner never adds items to the current Earley set, so the current Earley set, `S[i]` remains constant in `Scanner Loop`.

Algorithm 3 Marpa Earley Set Processing

```
Input variable: i

Completer Loop: for every (state, parent) in S[i] {
    if parent != i {
       do Completer(state, parent)
    }
} # End of Completer Loop

Scanner Loop: for every (state, parent) in S[i] {
    do Scanner(state, parent)
} # End of Scanner Loop
```

9 Scanner

The `Scanner` takes three input variables. The first two are the elements of an Earley item: `state` is an LR(0) state, and `parent` is the index of an Earley set. The third input variable, i, is the index of the current Earley set.

`k` and `nk` are temporaries, local to the subroutine, which hold AHFA state values returned by `GOTO`. `S` is a global array containing the Earley sets. `x` is global array containing the tokens.

Algorithm 4 Scanner

```
Input variables: state, parent, i

set k = GOTO(state, x[i+1])
if state k is not defined {
    return
}
add (k, parent) to S[i+1] at end, if it is not already present
set nk = GOTO(k, undefined)
if state nk is not defined {
    return
}
add (nk, i+1) to S[i+1] at end, if it is not already present
return
```

10 Completer

The `Completer` takes three input variables. The first two are the elements of an Earley item: `state` is an LR(0) state, and `parent` is the index of an Earley set. The third input variable, i, is the index of the current Earley set.

`k` and `nk` are temporaries, local to the subroutine, which hold LR(0) state values returned by `GOTO`. `rule` is a rule, returned by `COMPLETED`. `S` is a global array containing the Earley sets.

Algorithm 5 Completer

```
Input variables: state, parent, i

foreach rule in COMPLETED(state) {
    foreach ( pstate, pparent ) in S[parent] {
    set k = GOTO( pstate, LHS(rule) )
    if k is defined {
        add ( k, pparent ) to S[i] at end, if it is not already present
        set nk = GOTO( k, undefined )
        if nk is defined {
            add ( nk, i ) to S[i] at end, if it is not already present
        }
     } # if k is defined
}  # foreach rule
```

11 The Proof: Short Version

After, Initialization, the only changes to globals are to the Earley sets, and these occur in the Earley Set Processing. The Processing of Earley set i can change S[i] (called the current Earley set) and S[i+1] (the next Earley set). The proof is organized around these facts.

First, two lemmas are proved showing that if, before each Earley Set Processing step, the Earley sets in both algorithms are identical, then, after the Earley Set Processing Step, the current Earley sets and the next Earley sets remain identical .

11.1 The Current Earley Set Lemma

Statement of the Lemma: Assume that, before each Earley set is processed, the Earley sets in AH2002 are identical to the Earley sets in Marpa. Then, after Earley Set i Processing, S[i] in AH2002 is identical to S[i] in Marpa.

Proof: In the Earley Set Processing, S[i] will only be changed by the Completer. The Completer can be treated as a pure deterministic functions whose inputs are its input variables (state, parent and i), the current Earley set itself (S[i]) and the pure functions of the grammar (GOTO, LHS and COMPLETED). GOTO, LHS, COMPLETED do not change after Initialization and the input variable i will not change throughout the Earley Set Processing, so they may treated as constants for this purpose.

The input variables (state, parent) and the current Earley set (S[i]) do change but, in both AH2002 and Marpa, the calls to the Completer will be with same values for these and in the same sequence. This can be shown by induction on the calls to the Completer.

There may be zero such calls, and if this is the case, S[i] after Earley Set i Processing will clearly be unchanged from S[i] before Earley Set i Processing. Assume there have been n calls to the completer. By the induction hypothesis, after these n calls, S[i] is identical in both algorithms. The input variables state and parent depend only on S[i], and on i. i was already shown to be constant during the Earley Set i Processing. S[i] is identical in both algorithms by the induction hypothesis. The Completer is the same in both algorithms. Therefore, since the Completer and its inputs are identical, and the Completer is deterministic, the results of the Completer will be the same in both algorithms.

The induction above assumed that there is a n+1'th call to the Completer in both algorithms. This actually depends on the domain for the Completer Loop. But this domain is also the current Earley set, S[i]. So by the same induction as above, it is not possible there will be a n+1'th call to the Completer in one algorithm, without there being an n+1'th call to the Completer in the other algorithm. In other words, either there will be a n+1'st call to the Completer in both algorithms, or the n'th call to the Completer in both algorithms will have been the final one.

This completes the induction and shows that, after the Earley Set i Processing, the current Earley sets (S[i]) in both algorithms must be identical to each other. QED.

11.2 The Next Earley Set Lemma

Statement of the Lemma: Assume that, before each Earley set is processed, the Earley sets in AH2002 are identical to the Earley sets in Marpa. Then, after Earley Set i Processing, S[i+1] in AH2002 is identical to S[i+1] in Marpa.

Proof: First, recall the Current Earley Set Lemma. By it, we know that **after** Earley Set i Processing, the **current** Earley set (S[i]) is identical in both AH2002 and Marpa. We will make use

of this fact below.

In the Earley Set Processing, S[i+1] will only be changed by the Scanner. The Scanner can be treated as a pure deterministic function whose inputs are its input variables (state, parent and i), the next Earley set itself (S[i+1]), the token stream (x) and a pure function of the grammar (GOTO). GOTO and x do not change after Initialization and the input variable i will not change throughout the Earley Set Processing, so they may treated as constants for this purpose.

The input variables (state, parent) and the next Earley set (S[i+1]) do change but, in both AH2002 and Marpa, the calls to the Scanner will be with same values for these, will be made in the same sequence, and will have the same results. This can be shown by induction on the calls to the Scanner.

There may be zero such calls, and if this is the case, S[i+1] after Earley Set i Processing will clearly be unchanged from S[i+1] before Earley Set i Processing. Assume there have been n calls to the Scanner. By the induction hypothesis, after these n calls, S[i+1] is identical in both algorithms.

The input variables state and parent are the n+1'st work item in the current Earley set S[i]. Call this work item S[i][n+1]. As a work item, once S[i][n+1] is added to a work list it is never deleted and never changes. Since by the Current Earley Set Lemma, S[i] in AH2002 will be identical to S[i] in Marpa after Earley Set i Processing, if S[i][n+1] exists in both algorithms it must be identical.

S[i+1] is identical in both algorithms by the induction hypothesis. The Scanner is the same in both algorithms. Therefore, since the Scanner and its inputs are identical, and the Scanner is deterministic, the results of the Scanner will be the same in both algorithms.

The induction above assumed that there is a n+1'th call to the Scanner in both algorithms. This actually depends on the domain for the Scanner Loop. This domain is the current Earley set, S[i].

But what if there was an S[i][n+1] in AH2002 but no S[i][n+1] in Marpa, or vice versa? We can show this won't happen, by a reduction to absurdity. Suppose, just after the n'th call to the Scanner, S[i][n+1] exists in AH2002, but not in Marpa. By the Current Earley Set Lemma S[i] in both algorithms is identical after the Earley Set i Processing. But if S[i][n+1] does not exist for the Scanner Loop in Marpa, examination of the pseudocode will show that S[i][n+1] will not exist in Marpa at the End of Earley Set i Processing either. On the other hand, if S[i][n+1] exists at the End of Earley Set i Processing in AH2002, it will still exist at the End of Earley Set i Processing in AH2002. This means that S[i] at the End of Earley Set i Processing in AH2002 will not be the same as S[i] at the End of Earley Set i Processing in Marpa.

But, by the Current Earley Set Lemma, at the End of Earley Set i Processing, S[i] in each of the two algorithms must be identical to S[i] in the other. This means that the assumption for the reduction to absurdity in the paragraph above must be false. For the reduction to absurdity, we assumed that, just after the n'th call to the Scanner, there was an S[i][n+1] in AH2002, but not in Marpa. Therefore if, just after the n'th call to the Scanner, there is an S[i][n+1] in the AH2002, there must also be one in Marpa.

The same reduction to absurdity works in reverse, and shows the if, just after the n'th call to the Scanner, there is an S[i][n+1] in Marpa, there must also be one in AH2002. Putting the two together, just after the n'th call to the Scanner, there is an S[i][n+1] in AH2002 if and only if there is an S[i][n+1] in Marpa.

This means that it is not possible there will be a n+1'th call to the Scanner in one algorithm, without there being an n+1'st call to the Scanner in the other algorithm. In other words, either there will be a n+1'st call to the Scanner in both algorithms, or the n'th call to the Scanner in both algorithms will have been the final one.

This completes the induction and shows that, after the Earley Set i Processing, the next Earley

sets (S[i+1]) in both algorithms must be identical to each other. QED.

11.3 Showing Equivalence

We say that AH2002 and Marpa are equivalent if, when they are run on the same token streams and grammars, they produce the same Earley sets. Given the Current Earley Set Lemma and the Next Earley Set Lemma, the proof follows quickly. The proof is by induction on the Earley Set Processing.

Initialization is the same in AH2002 and Marpa, and the input to the algorithms (their tokens streams and grammars) are assumed to be identical. Initialization is deterministic, so before the Processing of Earley Set 0, all globals are identical.

Suppose before the Earley Set n Processing, all globals are identical. We know from examination of the pseudocode that, during Earley Set n Processing, the only globals which change are S[n] and S[n+1].

We have from the induction hypothesis that, before Earley Set n Processing, S[n] in AH2002 is identical is S[n] in Marpa. From this and the Current Earley Set Lemma, we can conclude that, after Earley Set n Processing, S[n] in each of the algorithms will be identical to S[n] in the other algorithm.

We have from the induction hypothesis that, before Earley Set n Processing, S[n+1] in AH2002 is identical is S[n+1] in Marpa. From this, the Current Earley Set Lemma and the Next Earley Set Lemma, we can conclude that, after Earley Set n Processing, S[n+1] in each of the algorithms will be identical to S[n+1] in the other algorithm.

Above we have that:

- All globals remains constant during Earley Set Processing except S[n] and S[n+1].
- After Earley Set n Processing, S[n] in each of the algorithms will be identical to S[n] in the other algorithm.
- After Earley Set n Processing, S[n+1] in each of the algorithms will be identical to S[n+1] in the other algorithm.

Combining these three, we have that if, after Earley Set n Processing, all globals in AH2002 will be identical to the corresponding global in Marpa. The point after Earley Set n Processing is the same as the point before Earley Set n+1 Processing. This proves the induction.

The Recognizer will process l+1 Earley sets, where l is the length of the token stream. From the induction, we know that, after the processing of the l+1 Earley sets, the Earley sets in Marpa will be identical to those in AH2002. This is the equivalence we were after.

QED.

12 Why a Detailed Proof?

The rest of this paper repeats the proof already given, more carefully and in more detail. It describes some new notations, useful for detailed proofs of this kind. Readers, if satisfied with what they have seen so far, will not necessarily want to read any further.

The form of the short proof above is similar to that of the correctness proofs as they usually appear in journals. Proofs in this form are problematic. English-language descriptions of program

locations, even when carefully written, are verbose. Even so, these descriptions are always in danger of being ambiguous.

More importantly, the reader can not easily confirm that all crucial details are being established. For example, in this proof, the existence or non-existence of loop passes and worklist items at certain points is as crucial as any issue in the proof, but very easy to overlook.

Short, open-form, English-language proofs of this kind, while they dominate the academic literature, are error-prone and hard to read. In the journals, perhaps reasons of space force their acceptance. Page-count in academic journals is a scarce resource, inelastic to the demands of academics. Detailed proofs would perhaps double the size of the corresponding articles. Doubling the page count means halving the number of articles published. Everyone in the field realizes that that one or more of the articles left unpublished might well be his own.

Web space is not nearly so expensive as space in printed academic journals. The rest of this paper contains the detailed working out of the correctness proof just given in short form. The short proof was written after, and based on, the detailed proof.

13 Notation for the Detailed Proof

13.1 Grafs

Most detailed proofs are in table form, with one equation per line. Equations are right aligned and their explanations left aligned. Program correctness proofs require a degree of explanation which easily overwhelms the one-line-per-equation format.

This proof is organized with one equation per paragraph, usually abbreviated "graf". The first sentence is an equation, and the others justify its derivation or provide explanations.

13.2 Time-Sets

This proof I introduce time-sets notation. Time-set notation can be looked at as an abbreviation of English, or as a semi-formalized, notation. I hope it is an improvement in precision. I would also claim it is more readable than its expansion into ordinary English would be.

A key to this notation is that the idea of sequence is an afterthought in time-set notation. Time-sets are unordered sets of point in the execution – time-points. Statements about their sequence – which time-point is before what other time-point, can obviously be important in analyzing programs. But I think premature introduction of the sequencing into the semantics is counterproductive.

The notation for locations in the code uses sets of time-points. The points in time, or time-points, are points in the processing – not locations in the pseudocode. Every operation of any kind separates time-points, so many time-points can correspond to the same pseudo-code location. Additionally, if a lexical location in the pseudocode is passed through several times, as it would be in a loop, each pass creates a new set of time-points for that location.

For time-points to be separate, some operation must happen between them. This means that sometimes the same time-point can be identified with two different lexical locations in the pseudocode, and have two different names. This occurs frequently in loops. More on this below.

13.3 Names of Sets

13.3.1 Algorithms

Each algorithm's name is used for the time-set of all the time-points in that algorithm. So AH2002 is the name of the time-set of all the time-points in the execution of the AH2002 algorithm. Marpa is the name of the time-set of all the time-points in the execution of the Marpa algorithm. Start of Marpa is the time point before any operations in the Marpa algorithm. End of Marpa is the time-point after any operations.

13.3.2 Pseudocode Sections

Each pseudocode section's name is used for the time-set of all time-points in that pseudo-code section, including all time-points in either algorithm. Recognizer is the name of the time-set of all time-points in the Recognizer, whether for AH2002 or Marpa. When it is useful to distinguish the @-notation can be used: `Recognizer @ AH2002` or `Recognizer @ Marpa`.

`Start of Recognizer` is the set of the time-points before any operation in the Recognizers, including both algorithms. `End of Recognizer @ Marpa` is the singleton set containing the time-point after all operations in the Marpa algorithm.

Pseudocode sections can be called multiple times. For example, `Scanner @ Marpa` refers to all time-points in any call of `Scanner` in the `Marpa` algorithm.

Where it is useful to specifically distinguish one call to a section, this can be done in the time-set's name. For example, the call the `Earley Set Processing` in the `AH2002` algorithm for Earley Set `s` is `Earley Set s Processing @ AH2002`.

13.4 Loops

Loops have not only a `Start` and `End`, but a `Top` and a `Bottom`. The `Start of Loop L` is before execution of any instructions in the loop. The `End of Loop L` is after execution of any instructions. The `Top of Loop L(c)` is before the execution of pass `c` through `Loop L`. The `Bottom of Loop L(c)` is after the execution of pass `c` through Loop `L`.

Passes through loops are numbered starting at 0. The final test which terminates a loop is considered a pass, but a pass which does not execute any instructions inside the loop. A loop over the numbers from 0 to 10 will have 12 passes, with pass 0 being the first pass, and pass 11 being the last pass. Pass 0 will have the loop variable with the value 0. In pass n, the value of the loop variable will be n, unless n is the last pass. In the last pass, only the test is performed – the fails, none of the instructions inside the loop are executed, and the loop terminates. Where z is the last pass, there is still considered to be a both `Top of Loop(z)` and `Bottom of Loop(z)`.

DEFINITION: Let L be the name of a loop. `DOMAIN(L)` is the array over which loop `L` iterates.

In this definition, it is important that L is not a timeset – it is the name of a loop. Also, it is important to note the value of `DOMAIN` can change if the loop is over a worklist.

DEFINITION: Let L be the name of a loop. Then, `PASS(p, L)` $\Leftrightarrow$ `0 <= p <= SIZE(DOMAIN(L) @ End of L))`.

Informally, `PASS(p,L)` is true if `p` is the number of a valid pass of `L`. If `D == DOMAIN(L)`, then the loop variable for pass `p` is `D[p]`.

For any loop `L`, `PASS(p, L)` $\wedge$ `PASS(p+1, L)` $\Leftrightarrow$ `(Bottom(p) of L == Top(p+1) of L)`.

DEFINITION: `FINAL(L) == SIZE( DOMAIN(L) @ End of L )`

13.5 Operations on Time-Sets

Subset: `S1` $\subseteq$ `S2`

The notation remains semi-formal. @-notation is often written so that it's intended to be distributed over other notation. For example:

`(SIZE(x @ A) == SIZE(x @ B)) @ P`

is an abbreviation for

`SIZE(x @ A @ P) == SIZE(x @ B @ P)`

The general idea is that in `(expression) @ P`, where `expression` is not a term or another time-set, `P` should be "pulled inside" `expression`. Rules for this are not explicitly formalized, but use of this syntax is limited to simple cases where it should be clear what is intended – certainly far more clear than the usual English language description would be.

14 Work List Lemma

Statement of the Lemma:

Let `L` be a work list iteration.

Let `W` be a work list.

Then `0 <= j <= SIZE( W @ End of L )`$\Rightarrow$`PASS( j, L )`

And `FINAL( SIZE( W @ End of L ), L)`

PROOF: The proof is from the definitions of loop and work list. `L` is defined as being an iteration over every item in `W`, even those added during `L`, so that any item added to `W` before `End of L` must be included in the iteration.

Only one operation is allowed on `W` – adding items to the end. This operation does not move or delete any items, so no item in `W` will ever be moved or deleted. That means any iterated item will be present in `W @( End of L)`, and in the same position as when it was iterated. Therefore, if there is a `W[j]@(End of L)`, the loop variable for pass j will be `W[j]`.

QED.

15 Globals

15.1 The Globals Assumption

For the purpose of the proof in this paper, it is assumed that AH2002 and Marpa are initialized with the same input token stream and grammar.

15.2 The Globals Lemmas

Statement of Lemma 1: Let G be a global.

Then G @ Start of Recognizer @ AH2002 == G @ Start of Recognizer @ Marpa.

Proof: From the definitions, we know that the Initialization code is shared by the two algorithms. From the Globals Assumption, we have that the results of Initialization depends only on the input tokens and the grammar.

QED.

STATEMENT OF LEMMA 2: Let G be global not in the Earley sets.

Let P be a time-set, P $\subseteq$Recognizer.

Let Q be a time-set, P $\subseteq$Recognizer.

Then For all G, where G is a global not in the Earley sets, G @ P == G @ Q.

Proof: The pseudocode and the definition of the Recognizer show that the only globals that change in the Recognizer are those in the Earley Sets.

Statement of Lemma 3: The Earley sets are changed only in the the following ways:

- S[0] is changed in Initialization;
- S[i] is changed in Completer @ Earley Set i Processing; and
- S[i+1] is changed in Scanner @ Earley Set i Processing.

Proof: From the pseudocode and the definition of Initialization.

QED

16 The Detailed Proof

16.1 AH2002 Halting Axiom

In this paper, AH2002 will be assumed to halt for all finite grammars and input token streams. AH2002 in their paper do not explicitly prove that this is true in general, but it is at the very least a safe conjecture.

16.2 The Completer Lemma

In this lemma A may be either Marpa or AH2002.

Statement of the Lemma:

Assume p is a pass in the Completer Loop.

Assume that `((state, parent, i) @ AH2002 == (state, parent, i) @ Marpa) @ Top of Completer Loop(p)`.

Assume `(S[i] @ A == S[i] @ B) @ Top of Completer Loop(p)`.

Then `(S[i] @ A == S[i] @ B) @ Bottom of Completer Loop(p)`.

Starting the proof:

`0 <= p <= SIZE(S[i] @ End of Completer Loop)`. By the Loop Range Property.

The proof proceeds in two cases – the Final Case `(p == SIZE(S[i] @ End of Completer Loop))` and the Kernel Case `(0 <= p < SIZE(S[i] @ End of Completer Loop))`.

Kernel case:

`((state, parent, i) @ AH2002 == (state, parent, i) @ Marpa) @ Top of Completer Loop(p` By assumption in the statement of the lemma.

`((state, parent, i) @ AH2002 == (state, parent, i) @ Marpa) @ Top of Completer @ Completer Loop(p)`. From the statement just previous and inspection of the psuedocode. These variables are never changed between `Top of Completer Loop(p)` and `Top of Completer @ Completer Loop(p)`.

For `G` in LHS, GOTO, COMPLETED, `(G @ Marpa == G @ AH2002) @ Top of Completer @ Completer Loop(p)`. By the Globals Lemma.

`(S[i] @ AH2002 == S[i] @ Marpa) @ Start of Completer @ Completer Loop(p)`. By assumption in the statement of the lemma.

`(S[i] @ AH2002 == S[i] @ Marpa) @ End of Completer @ Completer Loop(p)`. The Completer pseudocode shows that S[i] @ End of Completer is a deterministic function of S[i] itself, the

Completer's input variables and these globals: LHS, GOTO, COMPLETED. By the previous 3 grafs, these are the same in both algorithms. The Completer is also the same in both algorithms. Therefore the effects of the Completer must be the same in both algorithms.

`(S[i] @ AH2002 == S[i+f] @ Marpa) @ Bottom of Completer Loop(p)`. From the pseudocode. The current Earley set is never changed between `Bottom of Completer @ Completer Loop(p)` and `Bottom of Completer Loop(p)`.

This proves the Kernel case.

Final case:

`FINAL(p, Completer Loop)`.

`(S[i] @ Marpa == S[i] @ AH2002) @ Bottom of Completer Loop(p)`. From the previous graf and an assumption in the statement of this lemma. By the definition of the final pass, it is a no-op. The Completer is not called and `S[i]` remains to its value at the `Top Completer Loop(p)`.

This proves the final case.

With the two cases proved, the Lemma is shown.

QED.

16.3 The Completer Loop Lemma

When a task property Lemma is used in this Lemma as a justification, and no task is mentioned, the task is the Completer pseudocode, the loop is the Completer Loop and the offset is 1, and the Completer Task Lemma is to be considered part of the justification.

Statement of the Lemma:

Let `p` be a pass in `Completer Loop @ Earley Set i Processing`.

Assume for all globals G not in the Earley sets, `(G @ Marpa == G @ AH2002) @ Earley Set i Processing)`.

Assume `(i @ Marpa == i @ AH2002) @ Earley Set Processing`.

Assume `(S[i] @ Marpa == S[i] @ AH2002) @ Top(p) of Completer Loop`(p).

Then `(S[i] @ Marpa == S[i] @ AH2002) @ Bottom of Completer Loop(p)`.

Proof:

`(S[i][p] @ AH2002 == S[i][p] @ Marpa) @ Top of Completer Loop(p)`. By assumptions in for the lemma.

`((state, parent) @ AH2002 == (state, parent) @ Marpa) @ Top of Completer Loop(p)`. From the statement just previous. `(state, parent) == S[i][p]` because both are the loop variable.

`(i @ AH2002 == i @ Marpa) @ Top of Completer Loop (p)`. From an assumption for this lemma.

`(S[i] @ AH2002 == S[i] @ Marpa) @ Top of Completer Loop(p)`. An assumption for this lemma.

`(S[i] @ AH2002 == S[i] @ Marpa) @ Bottom of Completer Loop(p)`. By the Completer Lemma with `p` set to its value in this lemma, and by the three previous grafs.

This proves the lemma.

16.4 The Current Earley Set Lemma

Statement of the Lemma:

Assume `0 <= i <= SIZE(x)`.

Assume `(S @ AH2002 == S @ Marpa) @ Start of Earley Set i Processing`.

Then, (S[i] @ AH2002 == S[i] @ Marpa) @ End of Earley Set i Processing.

Starting the Proof:

Sublemma:

Assume (S[i] @ AH2002 == S[i] @ Marpa) @ Top of Completer Loop(0).

Assume p <= SIZE(S[i]) @ AH2002 @ Bottom of Completer Loop(p) or p <= SIZE(S[i]) @ Marpa @ Bottom of Completer Loop(p).

Then (S[i] @ AH2002 == S[i] @ Marpa) @ Bottom of Completer Loop(p).

Note: In this sublemma, it is assumed in the statement of the sublemma that pass p exists for one algorithm, but not for the other. Loop operations are not defined for non-existent passes, and it must be proved that pass p exists for both AH2002 and Marpa.

Starting the proof:

Induction Property: (S[i] @ A == S[i] @ Marpa) @ Bottom of Completer Loop(p), where the induction is on p.

Induction Basis: (S[i] @ A == S[i] @ Marpa) @ Bottom of Completer Loop(0). By the assumptions in the statement of the sublemma and the Completer Loop Lemma with p := 0. Pass 0 exists for every loop, by definition.

This proves the induction basis.

Induction hypothesis:

Assume (S[i] @ AH2002 == S[i] @ Marpa) @ Bottom of Completer Loop(n).

Inductive step:

Note: Next it must be shown that n+1 is a pass in both AH2002 and Marpa.

Subsublemma: Let (A,B) be either (AH2002, Marpa) or (Marpa, AH2002).

Assume PASS(n+1, Completer Loop @ A).

Then PASS(n+1, Completer Loop @ B).

Starting the Proof:

0 <= n < SIZE(DOMAIN(Completer Loop) @ Bottom of Completer Loop(n) @ A)). By the assumption for the sublemma, and the definition of PASS.

DOMAIN(Completer Loop) is S[i]. By the definition of DOMAIN.

0 <= n+1 <= SIZE(S[i] @ Bottom of Completer Loop(n) @ A). By the previous 2 grafs. [Current Set Size Graf.]

(S[i] @ AH2002 == S[i] @ Marpa) @ Bottom of Completer Loop(n). By the Induction Hypothesis. The assumption for the case is that f == 0.

(S[i] @ A == S[i] @ B) @ Bottom of Completer Loop(n). By the previous, substituting A and B, and transposing the equality if necessary.

0 < n+1 <= SIZE(S[i] @ Bottom of Completer Loop(n) @ B). By the previous graf and Current Set Size Graf.

0 < n+1 <= SIZE(DOMAIN(Completer Loop) @ Bottom of Completer Loop(n) @ A)). By the previous graf and the definition of DOMAIN. The domain of the Completer Loop is S[i].

PASS(n+1, Completer Loop @ B). By the previous graf and the definition of PASS.

This proves the subsublemma.

PASS(n+1, Completer Loop @ AH2002) $\Rightarrow$ PASS(n+1, Completer Loop @ Marpa). By the sublemma, with (A, B) := (AH2002, Marpa).

PASS(n+1, Completer Loop @ Marpa) $\Rightarrow$ PASS(n+1, Completer Loop @ AH2002). By the sublemma, with (A, B) := (Marpa, AH2002).

Pass Equivalence Graf: PASS(n+1, Completer Loop @ AH2002) $\Leftrightarrow$PASS(n+1, Completer Loop @ Marpa). By the previous two grafs.

`PASS(n+1, Completer Loop @ AH2002)`. By the assumption for Sublemma 1 and the Pass Equivalence Graf.

`PASS(n+1, Completer Loop @ Marpa)`. By the assumption for Sublemma 1 and the Pass Equivalence Graf.

`(S[i+f] @ AH2002 == S[i+f] @ Marpa) @ Bottom of Completer Loop(p)`. By the previous 2 grafs and the Earley Item Induction Property

THIS PROVES THE INDUCTION.

THIS PROVES THE SUBLEMMA.

NOTE: The sublemma does most of the work of this Lemma. In statements that follow, I show `(S[i] @ AH2002 == S[i] @ Marpa)` holds as we proceed through the Earley Set Processing. To do this, I use the sublemma for the portion of the Earley Set Processing where `S[i]` might change. Where `S[i]` remains unchanged in both algorithms, this is established by referring to the pseudocode.

`(S[i] @ AH2002 == S[i] @ Marpa) @ Start of Earley Set i Processing`. From the assumption in the Statement of the Lemma, substituting `S[i]` for `S`.

`((S[i] @ AH2002 == S[i] @ Marpa) @ Top of Completer Loop(0)) @ Earley Set i Processin` From the pseudocode and the previous graf. [Top Identical Graf.]

Let `z` be the pass such that `(AH2002 @ End of Completer Loop == AH2002 @ Bottom of Completer Loop(z)) @ Earley Set i Processing`. By the AH2002 Halting Axiom there is such a pass.

`(S[i] @ AH2002 @ Bottom of Completer Loop(z) == S[i] @ Marpa @ Bottom of Completer Loop(z)) @ Earley Set i Processing`. By the previous two statements and sublemma. Pass `z` exists for AH2002 by definition. That it exists for Marpa will be shown by the sublemma.

`(S[i] @ Marpa @ End of Completer Loop == S[i] @ Marpa @ Bottom of Completer Loop(z)) @ Earley Set i Processing`. By the last two statements and the loop definitions. If at pass z the Earley sets are identical, and the Earley set in one algorithm ends the loop, it must also end the loop in the other algorithm at pass z.

`(S[i] @ AH2002 == S[i] @ Marpa) @ End of Completer Loop @ Earley Set i Processing`. By the three statements just previous.

`(S[i] @ AH2002 == S[i] @ Marpa) @ End of Earley Set i Processing`. From the pseudocode.

THIS PROVES THE LEMMA.

16.5 The Scanner Lemma

STATEMENT OF THE LEMMA:

Assume `PASS(p, Scanner Loop)`.

Assume that `((state, parent) @ AH2002 == (state, parent) @ Marpa) @ Top of Scanner Loop`(p).

Assume that `(i @ AH2002 == i @ Marpa) @ Top of Scanner Loop(p)`.

Assume `(S[i+1] @ AH2002 == S[i+1] @ Marpa) @ Top of Scanner Loop(p)`.

Then `(S[i+1] @ AH2002 == S[i+1] @ Marpa) @ Bottom of Scanner Loop(p)`.

STARTING THE PROOF:

`0 <= p <= SIZE(S[i] @ End of Scanner Loop)`. From as assumption for the lemma and the definition of `PASS`.

The proof proceeds in two cases – the Final Case (`p == SIZE(S[i] @ End of Scanner Loop)`) and the Kernel Case (`0 <= p < SIZE(S[i] @ End of Scanner Loop)`).

KERNEL CASE:

`((state, parent, i) @ AH2002 == (state, parent, i) @ Marpa) @ Top of Scanner Loop(p)`. By assumption in the statement of the lemma.

`((state, parent, i) @ AH2002 == (state, parent, i) @ Marpa) @ Top of Scanner @ Scanner Loop(p)`. From the graf just previous and the psuedocode. These variables are never changed between `Top of Scanner Loop(p)` and `Top of Scanner @ Scanner Loop(p)`. [INPUT VARIABLES GRAF]

`G` $\in$ `(x, GOTO)` $\Rightarrow$ `(G @ Marpa == G @ AH2002) @ Top of Scanner @ Scanner Loop(p)`. By the Globals Lemmas. [GLOBALS GRAF]

`(S[i+1] @ AH2002 == S[i+1] @ Marpa) @ Top of Scanner Loop(p)`. By an assumption in the statement of the lemma.

`(S[i+1] @ AH2002 == S[i+1] @ Marpa) @ Top of Scanner @ Scanner Loop(p)`. From the graf just previous and inspection of the pseudocode. This Earley set is never changed outside of Scanner. [NEXT EARLEY SET GRAF]

`S[i+1] @ AH2002 @ Bottom of Scanner @ Scanner Loop(p) == S[i+1] @ Marpa @ Bottom of Scanner @ Scanner Loop(p)`. From the pseudocode, the Input Variables Statement, the Globals Statement and Destination Statement. Inspection of Scanner shows that `S[i+1] @ End of Scanner` is a deterministic function. Inputs of the function are `S[i+1]` itself, the `Scanner`'s input variables and the globals `x` and `GOTO`. These inputs are the same in both algorithms by the Next Earley Set Graf, the Input Variables Graf and the Globals Graf. The `Scanner` is also the same in both algorithms. Therefore the result must be the same.

`(S[i+1] @ AH2002 == S[i+f] @ Marpa) @ Bottom of Scanner Loop(p)`. From the pseudocode. This Earley set is never changed between `Bottom of Scanner @ Scanner Loop(p)` and `Bottom of Scanner Loop(p)`.

THIS PROVES THE KERNEL CASE.

FINAL CASE:

ASSUMPTION FOR THIS CASE: `FINAL(p, Scanner Loop)`.

`(S[i+1] @ Marpa == S[i+1] @ AH2002) @ Bottom of Scanner Loop(p)`. From an assumption in the statement of this lemma, the graf just previous and the loop definitions. The final pass of a loop is always a no-op.

THIS PROVES THE FINAL CASE.

With the two cases proved, the Lemma is shown.

QED.

16.6 The Scanner Loop Lemma

STATEMENT OF THE LEMMA:

Assume that `PASS(p, Scanner Loop @ Earley Set i Processing)`.

Assume that `(S @ AH2002 == S @ Marpa) @ End of Earley Set i Processing`.

Assume `(i @ Marpa == i @ AH2002) @ Earley Set Processing`.

Assume `(S[i+1] @ Marpa == S[i+1] @ AH2002) @ Top of Scanner Loop(p)`.

Then `(S[i+1] @ Marpa == S[i+1] @ AH2002) @ Bottom of Scanner Loop (p)`.

PROOF:

`(S[i] @ AH2002 == S[i] @ Marpa) @ End of Earley Set i Processing`. By an assumption for the lemma, with S[i] substituted for S.

`(S[i][p] @ AH2002 == S[i][p] @ Marpa) @ End of Earley Set i Processing`. From the statement just previous. p exists in both algorithms because S[i] is the domain of Scanner Loop,

and by assumption for the lemma, p is a pass.

`End of Scanner Loop @ Earley Set i Processing == End of Earley Set i Processing`. From the pseudocode. Recall that `Combined Loop @ AH2002 == Scanner Loop @ AH2002`.

`(S[i][p] @ AH2002 == S[i][p] @ Marpa) @ End of Scanner Loop @ Earley Set i Processing` From the previous 2 grafs.

`(S[i][p] @ Top of Scanner Loop(p) == S[i][p] @ End of Scanner Loop) @ Earley Set i Processing`. S[i][p] exists at `Top of Scanner Loop(p)` by the definition of PASS. It is identical to S[i][p] at End of Scanner Loop because worklist items, once they exist, are never changed or deleted.

`(S[i][p] @ AH2002 == S[i][p] @ Marpa) @ Top of Scanner Loop(p) @ Earley Set i Process` From the previous 2 grafs.

`((state, parent) == S[i][p]) @ Top of Scanner Loop(p) @ Earley Set i Processing`. Because (state, parent) and S[i][p] are two different names for the loop variable, they must be identical.

`((state, parent) @ AH2002 == (state, parent) @ Marpa) @ Top of Scanner Loop(p) @ Earley Set i Processing`. From the previous two statements.

`(i @ AH2002 == i @ Marpa) @ Top of Scanner Loop (p)`. From an assumption in the statement of this lemma.

`(S[i+1] @ AH2002 == S[i+1] @ Marpa) @ Top of Scanner Loop(p)`. An assumption in the statement of this lemma.

`(S[i+1] @ AH2002 == S[i+1] @ Marpa) @ Bottom of Scanner Loop(p)`. By the Scanner Lemma with p set to its value in this Lemma, and by the three statements just previous.

This proves the lemma.

16.7 The Next Earley Set Lemma

Statement of the Lemma:

Assume `SIZE(S[i] @ AH2002) == SIZE(S[i] @ Marpa) @ End of Earley Set i Processing`.

Assume `0 <= i <= SIZE(x)`.

Assume `(S @ AH2002 == S @ Marpa) @ Start of Earley Set i Processing`.

Then, `(S[i+1] @ AH2002 == S[i+1] @ Marpa) @ End of Earley Set i Processing`.

Starting the Proof:

Sublemma:

Assume `(S[i+1] @ AH2002 == S[i+1] @ Marpa) @ Top of Scanner Loop(0)`.

Assume `p <= SIZE(S[i+f]) @ AH2002 @ Bottom of Scanner Loop(p)` or `p <= SIZE(S[i+f]) @ Marpa @ Bottom of Scanner Loop(p)`.

Then `(S[i+f] @ AH2002 == S[i+f] @ Marpa) @ Bottom of Scanner Loop(p)`.

Note: In this sublemma, it is assumed in the statement of the sublemma that pass p exists for one algorithm, but not for the other. Loop operations are not defined for non-existent passes, and it must be proved that pass p exists for both AH2002 and Marpa.

Starting the proof:

Induction Property: `(S[i+f] @ A == S[i+f] @ Marpa) @ Bottom of Scanner Loop(p)`, where the induction is on `p`.

Induction Basis: `(S[i+f] @ A == S[i+f] @ Marpa) @ Bottom of Scanner Loop(0)`. By the assumptions in the statement of Sublemma 1 and the Earley Item Induction Property Lemma with p == 0. Pass 0 always exists for every loop, by definition.

This proves the induction basis.

INDUCTION HYPOTHESIS:

Assume `(S[i+f] @ AH2002 == S[i+f] @ Marpa) @ Bottom of Scanner Loop(n)`.

INDUCTIVE STEP:

Note: Next it must be shown that `n+1` is a pass in both `AH2002` and `Marpa`.

SUBSUBLEMMA: Let `(A,B)` be either `(AH2002, Marpa)` or `(Marpa, AH2002)`.

Assume `PASS(n+1, Scanner Loop @ A)`.

Then `PASS(n+1, Scanner Loop @ B)`.

`SIZE(S[i] @ AH2002) == SIZE(S[i] @ Marpa) @ End of Earley Set i Processing`. From an assumption for this lemma.

`(S[i] @ A == S[i] @ B) @ End of Earley Set i Processing`. By the previous, substituting A and B, and transposing the equality if necessary.

`(S[i] @ A == S[i] @ B) @ End of Scanner Loop`. From the previous graf and the pseudocode. Nothing between `End of Scanner Loop` and `End of Earley Set i Processing` changes `S[i]`. [END OF SCANNER LOOP EQUALITY GRAF]

`0 <= n < SIZE(DOMAIN(Scanner Loop) @ Bottom of Scanner Loop(n) @ A))`. By assumption for the sublemma and the definition of `PASS`.

`0 <= n+1 <= SIZE(S[i] @ Bottom of Scanner Loop(n) @ A)`. By the previous graf, the loop definitions. By the definition of `DOMAIN`, `DOMAIN(Scanner Loop) == S[i]`.

`SIZE(S[i] @ Bottom of Scanner Loop(n) @ A) <= SIZE(S[i] @ End of Scanner Loop @ A)`. By the loop definitions. End of Scanner Loop is at or after Bottom of Scanner Loop(n) and the loop domain never decreases in size.

`0 <= n+1 <= SIZE(S[i] @ End of Scanner Loop @ A)`. From the previous 2 grafs.

`0 <= n+1 <= SIZE(S[i] @ End of Scanner Loop @ B)`. From the previous and the `End of Scanner Loop Equality Graf`.

`PASS(n+1, Scanner Loop @ B)`. From the previous graf and definition of `PASS`.

THIS PROVES THE SUBSUBLEMMA.

`PASS(n+1, Scanner Loop @ AH2002)` $\Rightarrow$ `PASS(n+1, Scanner Loop @ Marpa)`. By the subsublemma, with `(A, B) := (AH2002, Marpa)`.

`PASS(n+1, Scanner Loop @ Marpa)` $\Rightarrow$ `PASS(n+1, Scanner Loop @ AH2002)`. By the subsublemma, with `(A, B) := (Marpa, AH2002)`.

`PASS(n+1, Scanner Loop @ AH2002)` $\Leftrightarrow$`PASS(n+1, Scanner Loop @ Marpa)`. By the previous two grafs. [PASS EQUIVALENCE GRAF]

`PASS(n+1, Scanner Loop @ AH2002)`. By the assumption for the sublemma and the Pass Equivalence Graf.

`PASS(n+1, Scanner Loop @ Marpa)`. By the assumption for the sublemma and the Pass Equivalence Graf.

`(S[i+f] @ AH2002 == S[i+f] @ Marpa) @ Top of Scanner Loop(p+1)`. By the loop definitions and the induction hypothesis. The two grafs just previous show that pass `n+1` exists for both algorithms.

`(S[i+f] @ AH2002 == S[i+f] @ Marpa) @ Bottom of Scanner Loop(p+1)`. By the previous graf and the Scanner Loop Lemma.

THIS PROVES THE INDUCTION.

THIS PROVES SUBLEMMA.

NOTE: The sublemma does most of the work of this lemma. In statements that follow, I show `(S[i+1] @ AH2002 == S[i+1] @ Marpa)` holds as we proceed through the `Earley Set Processing`. To do this, I use the sublemma for the portion of the `Earley Set Processing` where

S[i+1] might change. In most cases S[i+1] remains unchanged in both algorithms, as is clear from the pseudocode.

(S[i+1] @ AH2002 == S[i+1] @ Marpa) @ Start of Earley Set i Processing. From the assumption in the Statement of the Lemma, substituting S[i+1] for S.

((S[i+1] @ AH2002 == S[i+1] @ Marpa) @ Top of Scanner Loop(0)) @ Earley Set i Process From the pseudocode and the previous graf. [Top Identical Graf]

Let z be the pass such that (AH2002 @ End of Scanner Loop == AH2002 @ Bottom of Scanner Loop(z)) @ Earley Set i Processing. By the AH2002 Halting Axiom there is such a pass.

(S[i+1] @ AH2002 == S[i+f] @ Marpa) @ Bottom of Scanner Loop(z) @ Earley Set i Proces By the previous two statements and sublemma. Pass z exists for AH2002 by definition. That it exists for Marpa will be shown by the sublemma.

(S[i+1] @ Marpa @ End of Scanner Loop == S[i+1] @ Marpa @ Bottom of Scanner Loop(z)) @ Earley Set i Processing. By the last two statements and the loop definitions. If at pass z the Earley sets are identical, and the Earley set in one algorithm ends the loop, it must also end the loop in the other algorithm at pass z.

(S[i+1] @ AH2002 == S[i+1] @ Marpa) @ End of Scanner Loop @ Earley Set i Processing. By the three statements just previous.

(S[i+1] @ AH2002 == S[i+1] @ Marpa) @ End of Earley Set i Processing. From the pseudocode.

This proves the Lemma.

16.8 Earley Set Processing Lemma

Statement of the Lemma:

Assume for (S @ Marpa == S @ AH2002) @ Start of Earley Set n Processing.

Assume also, that 0 < n < SIZE(x).

Then (S @ Marpa == S @ AH2002) @ End of Earley Set n Processing.

Starting the Proof:

For all x, (x == n or x == n+1 or S[x] @ Marpa == S[x] @ AH2002) @ End of Earley Set n Processing. From the Globals Lemma, and the assumption in the Statement of the Lemma.

(S[n] @ Marpa == S[n] @ AH2002) @ End of Earley Set n Processing. From the Current Earley Set Lemma (i := n), and the assumptions in the Statement of the Lemma.

(S[n+1] @ Marpa == S[n+1] @ AH2002) @ End of Earley Set n Processing. From the Next Earley Set Lemma (i := n), the previous graf, and the assumptions in the Statement of the Lemma.

For all x, (S[x] @ Marpa == S[x] @ AH2002) @ End of Earley Set n Processing. Combining the three statements just previous.

(S @ Marpa == S @ AH2002) @ End of Earley Set n Processing. From the previous statement.

QED.

16.9 AH2002 and Marpa are Equivalent

We regard AH2002 and Marpa as equivalent if, given the Globals Assumption, the Earley sets are identical at the end of each algorithm. The Globals Assumption has been assumed throughout this proof.

Statement of the Theorem: S @ End of AH2002 == S @ End of Marpa.

lemma: Assume 0 < s < SIZE(x).

Then `(S @ AH2002 == S @ Marpa) @ End of Recognizer Step s`.

Starting the Proof: The proof is by induction on the Recognizer Steps.

Induction Property: `(S @ AH2002 == S @ Marpa) @ End of Recognizer Step t`, where the induction is on `t`.

Induction Base:

`(S @ AH2002 == S @ Marpa) @ Start of Recognizer Step 0`. By the Globals Lemmas.

`(S @ AH2002 == S @ Marpa) @ End of Recognizer Step 0`. By the just previous statement, the Globals Lemmas and the Earley Sets Processing Lemma with `n := 0`. There will always be an Earley set 0 and therefore by the definition of the Recognizer, a Recognizer Step 0.

Induction Hypothesis:

`(S @ AH2002 == S @ Marpa) @ End of Recognizer Step t`.

Inductive Step:

`0 <= t+1 < SIZE(x)`. From the statement of the Sublemma. [Induction Limit Graf]

`(S @ AH2002 == S @ Marpa) @ Start of Recognizer Step t+1`. By the definition of the Recognizer, the Induction Hypothesis and the Induction Limit Graf.

`(S @ AH2002 == S @ Marpa) @ End of Recognizer Step t+1`. By the just previous statement, the Induction Limit Graf, and the Earley Set Processing Lemma with `(n := s+1)`.

This completes the induction.

This proves the lemma.

`(S @ AH2002 == S @ Marpa) @ End of Recognizer Loop(SIZE(x) - 1)`. By the lemma with `s := SIZE(x)-1`.

`(S @ AH2002 == S @ Marpa) @ Start of Recognizer Loop(SIZE(x))`. By the graf just previous and the loop definitions. SIZE(x) is a pass by the definition of the Recognizer.

`(S @ AH2002 == S @ Marpa) @ End of Recognizer Loop(SIZE(x))`. By the loop definitions and the previous graf. SIZE(x) is the final pass of Recognizer Loop, by its definition. The final pass is a no-op, so the values of S will not change in either algorithm.

`(S @ AH2002 == S @ Marpa) @ End of Recognizer`. By the definition of the Recognizer and the previous graf.

`S @ End of AH2002 == S @ End of Marpa`. By the graf just previous and by the definition of the two algorithms.

QED.