Leo 1991 and Aycock-Horspool 2002 Work Together: A Proof

Jeffrey Kegler

June 9, 2010

Abstract

Joop Leo in a 1991 article modified Earley's algorithm to run in O(n) time for LR-regular grammars. Aycock and Horspool introduced practical innovations to Earley's in their 2002 article. Earley items were defined differently in the two algorithms. Leo used Earley items containing a single dotted rule, which was their original form. Aycock and Horspool modified Earley's items to contain sets of dotted rules which were states of a finite automata (AHFA states). Proof is given that Marpa's adaptation of the Leo algorithm to use AHFA states is correct.

Contents

1 Introduction

Marpa is a new parser based on Earley's algorithm. It includes modifications to Earley's algorithm from Joop Leo's 1991 parser and from the parser described by Aycock and Horspool in their 2002 paper. This document assumes that the reader is familiar with LR(0) automata, Earley's algorithm, Leo's 1991 paper[1] and Aycock and Horspool's 2002 paper[2].

In this article the recognizer described in Joop Leo's 1991 paper will be called Leo1991. The recognizer described in Aycock and Horspool's 2002 paper will be called AH2002.

The major obstacle to combining the two algorithms was the difference in the way each algorithm defined an Earley item. Leo1991 followed Earley is having one dotted rule per Earley item. AH2002 combined dotted rules into states of a finite automaton. (That automaton will be called an AHFA in this document.) AHFA states are sets of one or more dotted rules.

When a grammar was right-recursive, Earley's original algorithm needed to add a chain of Earley items to every Earley set where the recursion might end. This chain was a series of completions, culminating in what Leo 1991 calls a "topmost" item. In these chains, most of the Earley items

[1] Leo, Joop, "A General Context-Free Parsing Algorithm Running in Linear Time on Every LR(k) Grammar Without Using Lookahead", *Theoretical Computer Science*, Vol. 82, No. 1, 1991, pp 165-176.

[2] Aycock, John and Horspool, R. Nigel, "Practical Earley Parsing", *The Computer Journal*, Vol. 45, No. 6, 2002, pp. 620-630.

were almost completely useless. They were used once when creating the chain and never after that. Only one chain, the chain that was final Earley set of the right recursion, went on to have any of its non-topmost items included in the parse.

As the recursion lengthened, the chain of useless Earley items lengthened with it. The effect was that Earley's time complexity was quadratic for right recursion.

Leo's idea was to create only topmost items until the semantic phase. At that point the chain in the final Earley set of the right recursion could be expanded, based on the information in the series of topmost items. In Marpa, these topmost items are called Leo completions.

This paper proves that Leo completions are always Aycock-Horspool singletons – Earley items whose AHFA states contain only a single dotted rule. AHFA singletons are in effect identical to the conventional Earley items used by Leo, because both correspond to a single dotted rule.

The proof that Leo completions are singletons is the only non-intuitive part of the correctness proof, and is the only part described in this document. The proof that non-topmost completions in the chains are also always AHFA singletons would simply repeat the arguments of the proof for Leo completions. It is clear that Leo's correctness proof in his 1991 can be extended to Marpa.

Most of this paper assumes the context is a context-free grammar, G. The AFHA is assumed to be constructed from G. This assumption is only made explicit when properties of the grammar itself are important.

In this document, the term **NFA** is often used in its inclusive sense, and can mean either a deterministic finite automaton (DFA) or a properly non-deterministic finite automaton. The AHFA is an proper NFA, while most of the steps in its construction are DFA's.

2 Marpa's Grammar and the AHFA States

Marpa follows AH2002 in rewriting its grammars. Here are some features of this rewrite that are relevant in this document.

1. Marpa augments its grammars by adding a special start symbol.
2. Marpa may have a null parse rule. If it exists, the null parse rule is an empty rule with Marpa's special start symbol on its LHS.
3. All symbols in Marpa are either nulling or non-nullable. A Marpa grammar contains no proper nullable symbols.
4. There are no empty rules. The null parse rule is the only exception to this.
5. Every rule contains at least one non-nullable symbol. The null parse rule is the only exception to this.

Marpa also follows AH2002 in its special treatment of dotted rules in the AHFA states. Here are special features of Marpa's dotted rules which are relevant in this document.

1. As previously noted, Earley items contain AHFA states, which represent sets of one or more dotted rules.
2. No dotted rule has a nulling symbol after the dot.
3. All transitions between AHFA states are over non-nullable symbols.

4. Any transition over a non-nullable symbol A in an AHFA state also moves the dot past any nulling symbols after the symbol A.

5. A prediction is a dotted rule where the dot is before the first **non-nullable** symbol. As a consequence, the dot in an AHFA's prediction is not necessarily before the first symbol of the rule.

6. A completion always has the dot after the last symbol of the rule. In the terminology of Leo 1991, AHFA states contain no quasi-completions.

7. Predictions are never completions. This is because rules must contain at least one non-nullable symbol. The null parse rule is the only exception to this.

3 The Strategy of the Proof

The proof defines "Leo conformance". This is a property of an NFA state. A Leo-conformant NFA state either is an AHFA singleton or else it contains no candidates for Leo completions. If all the states in a NFA are Leo-conformant, clearly all candidates for Leo completions in the NFA will be in AHFA singletons.

AH2002 Earley items use sets of dotted rules in their Earley items, instead of individual dotted rules. These sets of dotted rules are states of a NFA which Aycock and Horspool call a "split ϵ-DFA". In this document, a split ϵ-DFA is called an Aycock-Horspool Finite Automaton (AHFA).

The proof contains many inductions on the construction of the AHFA. The one which establishes the Leo conformity of the AHFA is the **main induction**. AHFA construction starts with the creation of the LR(0) DFA start states from the LR(0) NFA. As the basis for the induction, the LR(0) DFA start states are shown to be Leo-conformant. Each step of the construction after that is a step of the induction. It is shown that all new NFA states are Leo-conformant, and that all transformed NFA states preserve Leo conformance.

Most of the work of the proof is done in a series of theorems which preceed the main induction. Most of these prove "conservation" properties to be used in the induction. Intuitively, a property or object is conserved if, once it comes into existence, it never changes. The conservation properties depend on each other and must be established in order. They often use induction on the construction on the AHFA.

Key to the proof will be the idea of "shadowing". It is possible for completions to be bound together in the same AFHA state, and this would appear to be an obstacle, perhaps a fatal one, to using Leo1991 with AHFA states. But Leo1991's additional logic is only applied when the postdot symbol in the completion's parent (or predecessor) Earley set is unique. The concept of shadowing converts this uniqueness requirement into a property of dotted rule instances. In the form of shadowing, the uniqueness requirement of Leo1991 can be applied in an induction on the steps in the construction of a DFA.

Leo 1991 had its own reasons for requiring that postdot symbols be unique – it enabled Leo's algorithm to have linear time-complexity for LR-regular grammars. But the result is very fortunate for adapting Leo1991 to AHFA states. It is an interesting question whether this is simply a happy coincidence or the consequence of a deep property.

4 Definitions

4.1 Dotted Rules and Instances

A **dotted rule** is a duple $(rule, dot)$, where $rule$ is a rule of G and dot is a number from 0 to n, where n is the number of symbols in $rule$'s right hand side. A dotted rule is more traditionally represented by showing the rule as a production with a raised dot to indicate the dot position: $A \rightarrow \alpha \bullet \beta$.

A **dotted rule instance** is a triple $(rule, dot, id)$. $rule$ and dot are as in the definition of a dotted rule. id is an unique identifier. More formally, an id is assigned to a dotted rule instance when it is created, such that id will never be equal to the id of any other dotted rule instance. Where the meaning is clear, a dotted rule instance is called a **rule instance** or an **instance.**

NFA states are defined as sets of dotted rule instances. Within an NFA state there is never more than one dotted rule instance for any dotted rule. But an NFA may contain different instances of the same dotted rule. Without the use of instances, confusion can arise when dealing with the movement of an element from one NFA state into another NFA state.

We say that a dotted rule instance is **in an Earley item**, if that dotted rule instance is an element of the AHFA state in that Earley item. We say that a dotted rule instance is **in an Earley set**, if that dotted rule instance is in any Earley item in that Earley set.

The reader may wish to note that even dotted rule instances are not necessarily unique within an Earley set. The same AHFA state can occur in multiple Earley items in the same Earley set. This does not seem to present any problem for the proofs in this document, but it was a complication that needed to be watched.

A s**ingleton** is an NFA state that contains only one dotted rule instance. A **completion** is a dotted rule instance with its dot position after its last symbol. A **singleton completion** is an NFA state whose only element is a completion.

A **prediction** is a dotted rule instance with its dot position before the first non-nulling symbol of its rule. An **initial** instance is a prediction of a start rule.

The **source** of a dotted rule instance is another dotted rule instance that was used in its creation. Initial instances do not have sources. If the instance is a non-prediction, the **source** is also called a predecessor.

4.2 Location Mappings

Location mappings are properties of dotted rule instances and NFA states. Let w be a string in the symbol vocabulary of the current grammar: $w \in V^*$. Let G be the current grammar. Let V be the symbol vocabulary of G. The location mapping for an AHFA state s is $Locate_s(w)$, where $Locate_s(w)$ is the set of Earley sets such that s is the AFHA state of some Earley item in the Earley set. The location mapping for a dotted rule instance d is $Locate_d(w)$, where $Locate_d(w)$ is the $Locate_s(w)$ such that $d \in s$. Note that w is not necessarily in the language of G, $L(G)$, that the parse is not necessarily successful, and that the set of Earley sets may be infinite.

Intuitively, the location mapping describes the locations where dotted rule instances are found when Marpa parses w according to the AHFA generated from G. There may be more than one Earley set occurrence of the same dotted rule instance, but $Locate_d(w)$ is not defined as a multiset. Even if the same dotted rule instance d is present in more than Earley item in an Earley set when string w is parsed, that Earley set is only a single element in the range of the mapping $Locate_d(w)$.

4.3 Ties

An Earley set that contains an AHFA state also contains all the dotted rule instances in that AHFA state. For this reason, dotted rule instances in an AHFA state have the same location mapping as that AHFA state.

Two dotted rule instances are **tied** if they have the same location mapping. From the above, it is clear that all pairs of dotted rule instances in an AHFA state are tied.

We say that an NFA state is **tied** if all the pairs of dotted rule instances in it are tied. AHFA states are **tied**. It will be proved (section 9 on page 9) that all NFA states arising in the consruction of an AHFA are tied.

Two AHFA states are **tied together** if they have the same location mapping. The tying together of AHFA states will become relevant when dealing with the NFA State Splitting operation (section 5.3 on the next page). (The result of NFA State Splitting is always an AHFA state.)

Theorem: If two AHFA states are tied togther, every pair of dotted rule instances from either state is tied. **Proof**: Because they are tied together, the two AHFA states have the same location mapping, call it m. Because they are AHFA states, all dotted rule instances in both states also have mapping m. Therefore all pairs of these are tied. QED.

4.4 Relevant Ties

By definition, dotted rule instances are tied if they have the same location mapping. This property is conserved during AHFA construction, but that proof would be complex because a coincidence of location mappings can arise in many ways. A weaker theorem, which restricts itself to relevant ties, is much easier to prove and is all that is necessary to prove the main results in this document.

A tie between dotted rule instances is **relevant** if it is between two dotted rule instances in the same NFA state, or if it is between dotted rule instances in two AHFA states which are tied together. This definition does not require that a relevant tie be a tie, but that will be proved in section 10 on page 10.

4.5 Shadowing

The predecessor of a dotted rule instance may be "shadowed". This definition is key to the proof. A predecessor p is shadowed by another dotted rule instance q, if

- q is relevantly tied to p,
- $p \neq q$, and
- p and q have the same postdot symbol.

Where q is not important, we simply say that p is shadowed. Intuitively, shadowing captures the notion that, whenever a dotted rule instance with p as its predecessor occurs in the recognizer, Leo1991's uniqueness requirement fails to be met.

4.6 Candidates and Rejects

As a reminder, a **Leo completion** is one of Earley items that replace chains of Earley items when the methods of Leo1991 are used. The Leo completion is a stand-in for the entire chain, and especially for its topmost item.

A dotted rule instance is a **candidate for Leo completion**, if it is a completion with an unshadowed predecessor. A candidate for Leo completion is usually called just a **candidate**, or sometimes **Leo candidate**. An NFA state is a **candidate** if any of its dotted rule instances is a candidate. Intuitively, a candidate is an dotted rule instance or an NFA state which could be included in a Leo completion.

A dotted rule instance is a **reject** if it is not a candidate. An NFA state is a **reject** if all of its dotted rule instances are rejects. Where confusion might arise, rejects are called Leo rejects.

4.7 Leo Conformance

Leo conformance is a property of an NFA state. A state of a NFA may be **Leo-conformant** in one of two ways. An NFA state is Leo-conformant if it is a singleton completion. A state of an NFA is also Leo-conformant if it is a Leo reject. An NFA is **Leo-conformant** if all of its states are Leo-conformant.

4.8 Conservation

The term **conservation** will be used for a series of related concepts in this document. The concept will be formalized differently in each case, but the cases share a common intuition. A property is **conserved** if the AHFA construction never changes it. An object is **conserved** if, once it comes into existence, it never disappears.

5 Operations in Constructing the AHFA

In the construction of the AHFA, three operations on NFA states occur: creation, expansion, and splitting. The pre-operation NFA state which provides data is called an **operand**. There will be only one operand, which may or may not be changed by the operation. The NFA states which are potentially changed or which are created by an operation are its **results**. Each operation has one or two results.

5.1 NFA State Creation

When a NFA state is created, the operand is an existing NFA state, s, which is not changed. A symbol transition from s is used to created dotted rule instances to put into the result, s'. s' is created during the operation. All dotted rule instances in s' will be non-predictions, and will have predecessors in s.

5.2 NFA State Expansion

When a NFA state is created, the operand is an existing NFA state, s. s is also the result. NFA state expansion adds a new dotted rule instance, D, to s. D is a non-initial prediction. D will have as its source a dotted rule instance in s.

5.3 NFA State Splitting

When a NFA state is split, the operand is an existing NFA state, k. The result of the split will be k and optionally, a new state nk. The result state k is called the kernel state. nk is the non-kernel

state.

Every non-initial prediction instance, D, is moved from s into nk. D is not changed in the move. Since by definition D is the triple $(rule, dot, id)$, $rule$, dot, and id are also not changed. Because there will always be either an initial rule instance or a non-prediction instance in s, an NFA State Splitting operation always leaves k with at least one dotted rule instance.

By the definitions of the Marpa and AH2002 algorithms, for every parse, whenever k occurs in an Earley set, nk occurs along with it. In the terminology of this document, k and nk are tied together.

NFA State Splitting is always the last operation on any operand. This means that its result states are AHFA states, and are never the operands of any operation.

6 Instances are Conserved

Statement of the Theorem:

1. Dotted rule instances are not deleted once created.
2. Where a dotted rule instance is the triple $(rule, dot, id)$, $rule$, dot, and id never change.
3. For the properties of being a completion rule instance, being a prediction rule instance, and their negations, once a dotted rule instance has that property, it keeps that property.
4. For the properties of being an initial rule instance and its negation, once a dotted rule instance has that property, it keeps that property.

Proof: 1. By the definitions of NFA State Creation, NFA State Expansion and NFA State Splitting, no dotted rule instance is ever deleted. So by induction on the AHFA construction, no dotted rule instance is ever deleted. This establishes part 1 of the theorem.

2. A dotted rule instance D is the triple $(rule, dot, id)$ as a matter of definition. From part 1, we know that D is not deleted. This establishes part 2 of the theorem.

3. The properties listed are all consequences of rule and dot position. From part 2, we know that rule and dot position never change. Therefore the properties listed cannot change. This establishes part 3 of the theorem.

4. An initial rule is a prediction whose rule is a start rule. Therefore being an initial rule instance, and its negation, are consequences of the rule and the dot position. From part 2, we know that rule and dot position never change. Therefore the properties of being initial and of being non-initial cannot change. This establishes part 4 of the theorem. QED.

7 Sources are Conserved

Statement of the Theorem: Let D be a dotted rule instance and $S = (srule, sdot, sstate)$ its source. If S ever exists, $srule$ and $sdot$ never change.

Proof: When a non-inital rule instance is added, it has a source. By the Instance Conservation Theorem, the source will continue to exist and its rule and dot position will never change. QED.

8 Singleton Completions are Conserved

Statement of the Theorem: Once an NFA state becomes a singleton completion, it will remain one for the entire AHFA construction.

Proof: By the Instance Conservation Theorem, completions are never deleted and never lose the property of being a completion. It remains to show that singletons are conserved when their only dotted rule instance is a completion.

Case for NFA State Creation: By its definition, NFA State Creation does not add instances to any existing NFA states. This establishes the case for NFA State Creation.

Case for NFA State Expansion: The only NFA state to which NFA State Expansion adds instances is its operand. However, the operand of an NFA State Expansion will never be a singleton completion. This is because the added rule instances are predictions, and in Earley's algorithm these are made using postdot symbols. A singleton completion has only one rule instance, a completion. Completions have no postdot symbols. Therefore, NFA State Expansion will never affect a singleton completion. This establishes the case for NFA State Expansion.

Case for NFA State Splitting: The dotted rule instance in a singleton completion is either an initial instance (the null parse rule instance) or a non-prediction. In either case, it remains in the kernel state. No dotted rule instances are ever added to the operand when it becomes the kernel state, so the operand of NFA State Splitting remains a singleton. Since it remains a singleton and its sole dotted rule instance remains a completion, the operand remains a singleton completion when it becomes the kernel state. This established the case for NFA State Splitting.

Concluding the Proof: We have shown that all of the operations in an AHFA construction preserve singleton completions. By induction on the AHFA construction, singleton completions, once created, are preserved. QED.

9 All NFA States are Tied

Statement of the Theorem: At every point in an AHFA construction, every NFA state is a tied state.

Starting the Proof: The proof that the NFA states in the AHFA Construction are tied proceeds by induction on its construction. The induction proceeds in reverse from the usual order. The basis of the induction is the final product, the AHFA states. The steps of the induction will be toward the initial states. We show the steps of the induction by cases, one for each operation.

Basis of the Induction: It was mentioned above that AHFA states are tied. That followed immediately from the definitions of the Marpa and AH2002 algorithms: The Earley items for these algorithms contain AHFA states, so that all the dotted rule instances in an AHFA state must always occur together in an Earley set.

Case for NFA State Splitting: NFA State Splitting is always the last operation in the construction of the NFA states involved. The one or two NFA states which result (k and nk) are AHFA states. As AHFA states, k and nk are tied. We need to show that the operand is tied. Call this operand, s.

By the definition of NFA State Splitting, k and nk are not only tied states, they are tied together. This means that states k and nk have the same location mapping. Since they are AHFA states, all the dotted rule instances in k and nk have the same location mapping as their AHFA states, and as each other. The dotted rule instances in s are exactly the union of those in k and nk. Therefore all the dotted rule instances in s have the same location mapping. This means that all pairs of dotted

rule instances in s are tied. So, by the definition of a tied state, s is tied. This establishes the case for NFA State Splitting.

Case for NFA State Expansion: We assume, from the reverse induction, that s', the result of NFA State Expansion, is tied. We seek to prove that s, the operand of NFA State Expansion, is tied.

Since s' is tied, all pairs of dotted rule instances in s' are tied. The dotted rule instances in s are a subset of those in s'. Therefore, all pairs of dotted rule instances in s are tied. From the definition of a tied state, this means that s is tied. This establishes the case for NFA State Expansion.

Case for NFA State Creation: Because this operation does not alter its operand, this case is true trivially.

Concluding the Proof: We have show that the final product of AHFA construction, the AFHA states, are tied. We have shown that for every operation in the construction, if the NFA states which result are tied, then the operands are tied. This completes the induction and shows that all NFA states in an AHFA construction are tied. QED.

10 Relevant Ties are Ties

Statement of the Theorem: All relevant ties are ties.

Proof: The definition of a relevant tie has two cases: relevant ties within an NFA state and relevant ties between NFA states. The proof is by these cases.

Case for Within: By a previous theorem (section 9 on the previous page), all NFA states are tied. By the definition of a tied NFA state, every pair of dotted rule instances which are elements of the same NFA state is tied. This establishes the case for relevant ties between dotted rule instances within a single NFA state.

Case for Between: By another previous theorem (section 4.3 on page 6) all dotted rule instances are tied if they are in two AHFA states which are tied together. This establishes the case for relevant ties between dotted rule instances in NFA states which are distinct, but tied together.

Concluding the Proof: Since in both cases, every pair of dotted rule instances which is relevantly tied is also tied, all relevant ties are ties. QED.

11 Relevant Ties are Conserved

Lemma

Statement of the Lemma: The operand of an operation is never an AHFA state tied to another state.

Proof of the Lemma: An NFA state tied to another state is always the result of NFA State Splitting. NFA State Splitting is always the the last operation on any NFA state. Therefore, the results of NFA State Splitting never become the operand of any operation, including another NFA State Splitting operation. QED.

Theorem: Relevant Ties are Conserved

Statement of the Theorem: Once two dotted rule instances become relevantly tied in the course of AHFA construction, they remain relevantly tied.

Starting the Proof: This proof is by induction on the construction of the AHFA.

Basis of the Induction: The basis of the induction is the first step of the AHFA construction: the initial states of the LR(0) DFA construction. The basis is vacuously true, since prior to the initial states are no dotted rule instances, no ties between them and therefore no relevant ties between them.

Inductive Step:

We first note that it is vacuously true that all relevant ties between dotted rule instances in two distinct NFA states are preserved by the induction steps. This is because a relevant tie between dotted rule instances in distinct NFA states must be due to one NFA state being tied to another. By the previous Lemma, an NFA state tied to another NFA state is never the operand for any operation.

It remains to show that relevant ties between dotted rule instances in a single NFA state are preserved. This is shown by cases, one for each operation.

Case for NFA State Creation: NFA State Creation does not change the dotted rule instances in any existing NFA states, including its operand. So all pre-existing relevant ties will be preserved. This established the case for NFA State Creation.

Case for NFA State Expansion: The only NFA state whose dotted rule instances are changed by NFA State Expansion is its operand. All relevant ties not in the operand will be preserved.

All pairs of dotted rule instances in the operand are preserved in the result. The result is a single NFA state, so all such instance pairs are relevantly tied. This means that any relevant ties pre-existing the operation are preserved. This establishes the case for NFA State Expansion.

Case for NFA State Splitting: The only NFA state whose dotted rule instances are changed by NFA State Splitting is its operand, so it is only necessary to show that any relevant ties between dotted rule instances in the operand are preserved.

Call the two result states, k and nk. Consider two arbitrary dotted rule instances from the operand, $D1$ and $D2$.

We need to deal with two subcases. First, $D1$ and $D2$ might be in the same AHFA state. Second, they might be in different AHFA states.

First subcase: If $D1$ and $D2$ are in the same AHFA state, they are relevantly tied, so that any relevant tie they had previously is preserved. This establishes the first subcase.

Second subcase: Consider the other subcase, where $D1$ and $D2$ are in different AHFA states. Without loss of generality, put $D1$ into n and $D2$ into nk. By the definition of NFA State Splitting, n and nk are tied together. Therefore $D1$ and $D2$ are relevantly tied, and any relevant tie they had previously is preserved. This establishes the second subcase.

Concluding the case: Since the choice of $D1$ and $D2$ was arbitrary, every pair of dotted rule instances from either k or nk is relevantly tied. Therefore any relevant tie which existed previously is conserved. This establishes the case for NFA State Splitting.

Concluding the proof: We have shown that relevant ties are vacuously preserved when they are due to NFA states tied together. We have shown that relevant ties of dotted rule instances in a single NFA state are preserved in the basis of the induction and in its steps. This establishes the theorem.

QED.

12 Shadowing is Conserved

Statement of the Theorem: If a dotted rule instance becomes shadowed during AHFA construction, it remains shadowed.

Proof: By the definition of shadowing, it is based on existence of two dotted rule instances, their postdot symbols and the relevant tie between them. The postdot symbol is a function of the rule and dot position in a dotted rule instance and is therefore conserved. Since the existence of dotted rule instances, their postdot symbols, and relevant ties between them are conserved, shadowing is conserved. QED.

13 Reject Instances are Conserved

Statement of the Theorem: If a dotted rule instance becomes a Leo reject instance during AHFA construction, it remains a Leo reject instance.

Proof: A dotted rule instance may be a Leo reject because it is a non-completion or because its predecessor is shadowed. Dotted rule instances, predecessor relationships, shadowing and non-completion are all conserved objects or properties. Therefore, Leo reject instances are conserved. QED

14 Reject States are Conserved

Statement of the Theorem: If an NFA state becomes a Leo reject state during AHFA construction, it remains a Leo reject state.

Proof: A Leo reject state is one all of whose dotted rule instances are Leo rejects.

Case for NFA State Creation: NFA State Creation does not affect existing states, so it conserves all Leo reject states.

Case for NFA State Expansion: NFA State Expansion adds non-initial predictions to existing states. Non-initial predictions are always non-completions and therefore Leo rejects. Adding a Leo reject instance to a Leo reject NFA state results in a Leo reject NFA state.

Case for NFA State Splitting: NFA State Splitting produces two new states from an existing one. Each of these will contain only dotted rule instances from the original state. If the operand was a Leo reject, then it contained only Leo reject instances. It was proved in section 13 that Leo reject instances are conserved, including by NFA State Splitting. Therefore both new NFA states (kernel and non-kernel) will contain only Leo reject instances. And therefore, both the kernel and non-kernel NFA states will be Leo rejects.

Concluding the Proof: Since all the operations conserve Leo reject NFA states, Leo reject NFA states are conserved. QED.

Note: Leo candidates are **not** conserved.

15 Leo Conformance is Conserved

Statement of the Theorem: If an NFA state becomes Leo conformant during AHFA construction, it remains Leo conformant.

Proof: An NFA state is Leo-conformant either because it is singleton completion, or because it is a Leo reject state. As we have shown, both these properties are conserved. Therefore, once an NFA state is Leo-conformant, it remains Leo-conformant. QED.

16 NFA State Creation is Leo-conformant

Statement of the Theorem: The NFA states produced by the NFA State Creation operation are Leo-conforming.

Proof: NFA State Creation is by transition over a symbol T from its operand, s. The operand is also the source of the transition. Call the new NFA state r. We distinguish three cases.

Singleton Completion Case: r contains a single dotted rule instance, and that instance is a completion. Then r is a singleton completion, and therefore Leo-conforming. This establishes the singleton completion case.

Singleton Non-completion Case: r contains a single dotted rule instance, D, and D is a non-completion. Therefore, D is a Leo reject. Since D is the only dotted rule instance in r, all the rule instances in r are Leo reject instances, and r is a Leo reject. As a Leo reject, r is Leo conforming. This establishes the singleton non-completion case.

Non-singleton Case: There is more than one dotted rule instance in r. These dotted rule instances were created by transition over a single symbol, T. Call one of the dotted rule instances in r, $D1$. Call another dotted rule instance in r, $D2$. We are allowed to assume $D1 \neq D2$, because the assumption for this case is that there is more than one dotted rule instance in r. Call the predecessor of $D1$, $P1$. Call the predecessor of $D2$, $P2$.

We know that $P1 \neq P2$, because they are predecessors of $D1$ and $D2$ through the same transition over symbol T, and if $P1 = P2$, then $D1 = D2$, which we have assumed not to be the case.

$P1$ and $P2$ are both in s and therefore are relevantly tied. Since they are relevantly tied and share the same postdot symbol, T, and since $P1 \neq P2$, $P1$ shadows $P2$.

Because $P1$ shadows $P2$, and $P2$ is the predecessor of $D2$, $D2$ is a Leo reject. Since the choice of $D2$ was arbitrary, all dotted rule instances in r are Leo rejects. Since all dotted rule instances in r are Leo rejects, r is a Leo reject. Since r is a Leo reject, it is Leo conforming.

This establishes the non-singleton case.

Concluding the Proof: It has been shown in all three cases that the NFA states created by the NFA State Creation operation are Leo conforming. QED.

17 Non-kernel States are Leo-conformant

Statement of the Theorem: When the NFA State Splitting operation produces a non-kernel state as a result, that non-kernel state is Leo-conforming.

Proof: By the definiton of NFA State Splitting, only non-initial predictions go into the non-kernel state. The only prediction which is a completion is the null parse rule instance, and that is an initial instance. Therefore, all dotted rule instances in the non-kernel state are non-completions.

Since dotted rule instances in the non-kernel state are non-completions, they are all Leo rejects. Since all rule instances in the non-kernel state are Leo rejects, the non-kernel state is a Leo reject. Since the non-kernel state is a Leo reject, it is Leo-conformant. QED.

18 Construction of the AHFA

This section reviews the AHFA construction to show that everything in it is accounted for by the operations listed above.

18.1 Construction of the LR(0) DFA

It is assumed that the reader is familiar with the procedure for constructing an LR(0) DFA. After creation of the initial states of the LR(0) DFA, construction of its accessible states proceeds by transitions over symbols and by ϵ-transitions. These correspond to NFA State Creation and NFA State Expansion operations, respectively.

18.2 Construction of the ϵ-DFA

Aycock and Horspool, in their presentation of the construction of the AHFA, show an intermediate transformation – conversion of an LR(0) DFA into a ϵ-DFA. Aycock and Horspool then go on to suggest a grammar rewriting, which they call NNF. This grammar rewriting accomplishes the same thing more easily. Their ϵ-DFA is apparently only for pedagogic purposes.

In this proof the grammar is assumed to be the result of the Marpa grammar rewritings as described above. Marpa's grammar rewritings incorporate the NNF rewriting as suggested by Aycock and Horspool. The grammar rewriting is done before construction of the LR(0) NFA, and the result is that the LR(0) DFA is also an ϵ-DFA.

18.3 Construction of the Split ϵ-DFA

The final stage splits NFA states into one or two non-empty NFA states. This is the operation described above as NFA State Splitting. The result of NFA State Splitting is what Aycock and Horspool call a "split ϵ-DFA". It is the final product of the construction. The "split ϵ-DFA" is what this document has been calling an AHFA.

19 The Main Induction

19.1 Basis

Proof: The construction of the AHFA begins with one or two initial states. The null parse start state is Leo conformant because it is a singleton completion. The non-null parse start state is Leo conformant because all dotted rule instances in it are non-completions and therefore Leo rejects. Therefore all initial states are Leo conformant.

19.2 Inductive Step

Most of the work of the induction steps was done in the preceeding theorems. All states once created remain Leo-conformant, as proved in the section 15 on page 12.

By the definition of the operations, new NFA states are created in three ways:

1. As initial states. These were proved Leo-conformant in section 19.1 and do not form part of the inductive step.

2. In the NFA State Creation operation. These were proved Leo-conformant in section 16 on the previous page.

3. As non-kernel states, during the NFA State Splitting operations. These were proved Leo-conformant in section 17 on the preceding page.

20 All Leo Candidates are in Singleton AHFA States

Theorem: All Leo candidates are in singleton AHFA states.

Proof: The main induction (section 19 on the previous page) shows that all the states in the AHFA are Leo conformant. Therefore, by the defiinition of Leo-conformant, all AHFA states are either contain no Leo candidates, or are singletons. This proves the theorem. QED.

21 All Leo Completions are in Singleton AFHA States

Theorem: All Leo completions are in singleton AFHA States

Proof: An Earley item can only be a Leo completion if its AHFA state contains a candidate for Leo completion. The previous section (20) showed that these these AHFA states are always singletons. Therefore Leo completions will always be singletons. QED