# MARPA, A PRACTICAL GENERAL PARSER: THE RECOGNIZER

JEFFREY KEGLER

Abstract. The Marpa recognizer is described. Marpa is a practical and fully implemented algorithm for the recognition, parsing and evaluation of context-free grammars. The Marpa recognizer is the first to unite the improvements to Earley's algorithm found in Joop Leo's 1991 paper to those in Aycock and Horspool's 2002 paper. Marpa tracks the full state of the parse, at it proceeds, in a form convenient for the application. This greatly improves error detection and enables event-driven parsing. One such technique is "Ruby Slippers" parsing, in which the input is altered in response to the parser's expectations.

## 1. Introduction

Despite the promise of general context-free parsing, and the strong academic literature behind it, it has never been incorporated into a highly available tool like those that exist for LALR[6] or regular expressions. The Marpa project was intended to take the best results from the literature on Earley parsing off the pages of the journals and bring them to a wider audience. Marpa::XS[9], a stable version of this tool, was uploaded to the CPAN Perl archive on Solstice Day in 2011. This paper describes the algorithm of Marpa::R2[8], a later version.

As implemented, Marpa parses, without exception, all context-free grammars. Time bounds are the best of Leo[10] and Earley[3]. The Leo bound, $\mathcal{O}(\texttt{n})$ for LR-regular grammars, is especially relevant to Marpa's goal of being a practical parser: If a grammar is in a class of grammar currently in practical use, Marpa parses it in linear time.

Error-detection properties, extremely important, have been overlooked in the past. Marpa breaks new ground in this respect. Marpa has the immediate error detection property, but goes well beyond that:

*Date*: June 19, 2013.

it is fully aware of the state of the parse, and can report this to the user while tokens are being scanned.

Marpa allows the lexer to check its list of acceptable tokens before a token is scanned. Because rejection of tokens is easily and efficiently recoverable, the lexer is also free to take an event-driven approach. Error detection is no longer an act of desperation, but a parsing technique in its own right. If a token is rejected, the lexer is free to create a new token in the light of the parser's expectations. This approach can be described as making the parser's "wishes" come true, and we have called this "Ruby Slippers Parsing".

One use of the Ruby Slippers technique is to parse with a clean but oversimplified grammar, programming the lexical analyzer to make up for the grammar's short-comings on the fly. The author has implemented an HTML parser[7], based on a grammar that assumes that all start and end tags are present. Such an HTML grammar is too simple even to describe perfectly standard-conformant HTML, but the lexical analyzer is programmed to supply start and end tags as requested by the parser. The result is a very simply and cleanly designed parser that parses very liberal HTML and accepts all input files, in the worst case treating them as highly defective HTML.

Section 2 describes the notation and conventions of this paper. Section 3 deals with Marpa's grammar rewrites. Sections 4 and 5 introduce Earley's algorithm. Section 6 describes Leo's modification to Earley's algorithm. Section 7 describes the modifications proposed by Aycock and Horspool. Section 8 presents the pseudocode for Marpa's recognizer. Section 9 describes notation and deals with other preliminaries to the theoretical results. Section 10 contain a proof of Marpa's correctness, while Section 11 contains its complexity results. Finally, section 12 generalizes Marpa's input model.

## 2. Preliminaries

We assume familiarity with the theory of parsing, as well as Earley's algorithm. This paper will use subscripts to indicate commonly occurring types.

| | |
|---|---|
| $\texttt{X}_T$ | The variable $\texttt{X}$ of type $T$ |
| $\texttt{set-one}_{\{T\}}$ | The variable `set-one` of type set of $T$ |
| $SYM$ | The type for a symbol |
| $\texttt{a}_{SYM}$ | The variable $\texttt{a}$ of type $SYM$ |
| $\texttt{set-two}_{\{SYM\}}$ | The variable `set-two`, a set of symbols |

Subscripts may be omitted when the type is obvious from the context. The notation for constants is the same as that for variables. Multi-character variable names will be common, and operations will never be implicit.

| | |
|---|---|
| Multiplication | $\texttt{a} \times \texttt{b}$ |
| Concatenation | $\texttt{a}\,.\,\texttt{b}$ |
| Subtraction | $\texttt{symbol-count} - \texttt{terminal-count}$ |

Type names are often used in the text as a convenient way to refer to their type.

Where $\texttt{vocab}_{\{SYM\}}$ is non-empty set of symbols, let $\texttt{vocab}^*$ be the set of all strings (type $STR$) formed from those symbols. Where $\texttt{s}_{STR}$ is a string, let $|\texttt{s}_{STR}|$ be its length, counted in symbols. Let $\texttt{vocab}^+$ be

$$\bigl\{ \texttt{x}_{STR} \bigm| \texttt{x}_{STR} \in \texttt{vocab} * \land |\texttt{x}_{STR}| > 0 \bigr\}.$$

In this paper we use, without loss of generality, the grammar $\texttt{g}$, where $\texttt{g}$ is the 3-tuple

$$(\texttt{vocab}_{\{SYM\}}, \texttt{rules}, \texttt{accept}_{SYM}).$$

Here $\texttt{accept}_{SYM} \in \texttt{vocab}$. Call the language of $\texttt{g}$, $\texttt{L}(\texttt{g})$, where $\texttt{L}(\texttt{g}) \subseteq \texttt{vocab}^*$.

$\texttt{rules}_{\{RULE\}}$ is a set of rules (type $RULE$), where a rule is a duple of the form $[\texttt{lhs}_{SYM} \rightarrow \texttt{rhs}_{STR}]$, such that

$$\texttt{lhs}_{SYM} \in \texttt{vocab} \quad \text{and} \quad \texttt{rhs}_{STR} \in \texttt{vocab}^+.$$

$\texttt{lhs}_{SYM}$ is referred to as the left hand side (LHS) of $\texttt{r}_{RULE}$. $\texttt{rhs}_{STR}$ is referred to as the right hand side (RHS) of $\texttt{r}_{RULE}$. The LHS and RHS of $\texttt{r}_{RULE}$ may also be referred to as $\texttt{LHS}(\texttt{r}_{RULE})$ and $\texttt{RHS}(\texttt{r}_{RULE})$, respectively. This definition follows [2], which departs from tradition by disallowing an empty RHS.

The rules imply the traditional rewriting system, in which $\texttt{x}_{STR} \Rightarrow \texttt{y}_{STR}$ states that $\texttt{x}_{STR}$ derives $\texttt{y}_{STR}$ in exactly one step; $\texttt{x}_{STR} \overset{+}{\Rightarrow} \texttt{y}_{STR}$ states that $\texttt{x}_{STR}$ derives $\texttt{y}_{STR}$ in one or more steps; and $\texttt{x}_{STR} \overset{*}{\Rightarrow} \texttt{y}_{STR}$ states that $\texttt{x}_{STR}$ derives $\texttt{y}_{STR}$ in zero or more steps.

We say that symbol $\texttt{x}_{SYM}$ is **nullable** if and only if $\texttt{x}_{SYM} \overset{*}{\Rightarrow} \epsilon$. $\texttt{x}_{SYM}$ is **nonnull** if and only if it is not nullable. Following Aycock and Horspool[2], all nullable symbols in grammar $\texttt{g}$ are nulling – every symbol which can derive the null string always derives the null string. It is shown in [2] how to do this without losing generality or the ability to efficiently evaluate a semantics that is defined in terms of an original grammar that includes symbols which are both nullable and non-nulling, empty rules, etc.

Also without loss of generality, it is assumed that there is a dedicated acceptance rule, $\mathtt{accept}_{RULE}$ and a dedicated acceptance symbol, $\mathtt{accept}_{SYM} = \mathtt{LHS}(\mathtt{accept}_{RULE})$, such that for all $\mathtt{x}_{RULE}$,

$$\begin{aligned} & \mathtt{accept}_{SYM} \notin \mathtt{RHS}(\mathtt{x}_{RULE}) \\ \wedge \quad & (\mathtt{accept}_{SYM} = \mathtt{LHS}(\mathtt{x}_{RULE}) \implies \mathtt{accept}_{RULE} = \mathtt{x}_{RULE}). \end{aligned}$$

We define rightmost non-null symbol of a string as

$$\begin{aligned} \mathtt{Right\text{-}NN}(\mathtt{x}_{STR}) \underset{\text{def}}{\equiv} \mathtt{rnn}_{SYM} \quad & \text{such that} \quad \exists\, \mathtt{pre}_{STR}, \mathtt{post}_{STR} \mid \\ & \mathtt{x}_{STR} = \mathtt{pre}_{STR} . \mathtt{rnn}_{SYM} . \mathtt{post}_{STR} \\ & \wedge \mathtt{post}_{STR} \overset{*}{\Rightarrow} \epsilon \\ & \wedge \neg(\mathtt{rnn}_{SYM} \overset{*}{\Rightarrow} \epsilon). \end{aligned}$$

We define the rightmost non-null symbol of a rule as

$$\mathtt{Right\text{-}NN}([\mathtt{lhs}_{SYM} \rightarrow \mathtt{rhs}_{STR}]) \underset{\text{def}}{\equiv} \mathtt{Right\text{-}NN}(\mathtt{rhs}_{STR}).$$

A rule $\mathtt{x}_{RULE}$ is **directly right-recursive** if and only if

$$\mathtt{LHS}(\mathtt{x}_{RULE}) = \mathtt{Right\text{-}NN}(\mathtt{x}_{RULE}).$$

$\mathtt{x}_{RULE}$ is **indirectly right-recursive** if and only if

$$\exists\, \mathtt{y}_{STR} \mid \mathtt{Right\text{-}NN}(\mathtt{x}_{RULE}) \overset{+}{\Rightarrow} \mathtt{y}_{STR} \wedge \mathtt{Right\text{-}NN}(\mathtt{y}_{STR}) = \mathtt{LHS}(\mathtt{x}_{RULE}).$$

$\mathtt{x}_{RULE}$ is **right recursive**, $\mathtt{Right\text{-}Recursive}(\mathtt{x}_{RULE})$, if and only if is either directly or indirectly right-recursive.

The definition of $\mathtt{g}$ does not sharply distinguish terminals from non-terminals. MARPA's implementations allow terminals to be the LHS of rules, and every symbol except $\mathtt{accept}_{SYM}$ can be a terminal. The implementations have options that allow the user to reinstate the traditional restrictions, in part or in whole. Note that, as a result of these definitions, sentential forms will be of type $STR$.

Let the input to the parse be $\mathtt{w}$ such that $\mathtt{w} \in \mathtt{vocab}^{+}$. Locations in the input will be of type $LOC$. Let $|\mathtt{w}|$ be the length of the input, counted in symbols. When we state our complexity results later, they will often be in terms of $\mathtt{n}$, where $\mathtt{n} = |\mathtt{w}|$. Let $\mathtt{w}[\mathtt{i}]_{SYM}$ be character $\mathtt{i}$ of the input, $0 \leq \mathtt{i}_{LOC} < |\mathtt{w}|$.

The alert reader may have noticed that the previous definition of $\mathtt{w}$ did not allow zero-length inputs. To simplify the mathematics, we exclude null parses and trivial grammars from consideration. In its implementations, the Marpa parser deals with null parses and trivial grammars as special cases. (Trivial grammars are those that recognize only the null string.)

In this paper, Earley will refer to the Earley's original recognizer[3]. Leo will refer to Leo's revision of Earley as described in [10]. AH will refer to the Aycock and Horspool's revision of Earley as described in [2]. Marpa will refer to the parser described in this paper. Where Recce is a recognizer, $\mathtt{L}(\text{Recce}, \mathtt{g})$ will be the language accepted by Recce when parsing $\mathtt{g}$.

## 3. Rewriting the grammar

We have already noted that no rules of $\mathtt{g}$ have a zero-length RHS, and that all symbols must be either nulling or non-nullable. These restrictions follow Aycock and Horspool[2]. The elimination of empty rules and proper nullables is done by rewriting the grammar. [2] shows how to do this without loss of generality.

Because Marpa claims to be a practical parser, it is important to emphasize that all grammar rewrites in this paper are done in such a way that the semantics of the original grammar can be reconstructed simply and efficiently at evaluation time. As one example, when a rewrite involves the introduction of new rule, semantics for the new rule can be defined to pass its operands up to a parent rule as a list. Where needed, the original semantics of a pre-existing parent rule can be "wrapped" to reassemble these lists into operands that are properly formed for that original semantics.

As implemented, the Marpa parser allows users to associate semantics with an original grammar that has none of the restrictions imposed on grammars in this paper. The user of a Marpa parser may specify any context-free grammar, including one with properly nullable symbols, empty rules, etc. The user specifies his semantics in terms of this original, "free-form", grammar. Marpa implements the rewrites, and performs evaluation, in such a way as to keep them invisible to the user. From the user's point of view, the "free-form" of his grammar is the one being used for the parse, and the one to which his semantics are applied.

## 4. Earley's algorithm

Let $\mathtt{r}_{RULE} \in \mathtt{rules}$ be a rule, and $|\mathtt{r}|$ the length of its RHS. A dotted rule (type $DR$) is a duple, $[\mathtt{r}_{RULE}, \mathtt{pos}]$, where $0 \leq \mathtt{pos} \leq |\mathtt{r}_{RULE}|$. The position, $\mathtt{pos}$, indicates the extent to which the rule has been recognized, and is represented with a large raised dot, so that if

$$[\mathtt{A}_{SYM} \rightarrow \mathtt{X}_{SYM} \cdot \mathtt{Y}_{SYM} \cdot \mathtt{Z}_{SYM}]$$

is a rule,

$$[\mathtt{A}_{SYM} \rightarrow \mathtt{X}\,.\,\mathtt{Y} \bullet \mathtt{Z}]$$

is the dotted rule with the dot at $\mathtt{pos} = 2$, between $\mathtt{Y}_{SYM}$ and $\mathtt{Z}_{SYM}$.

If we let $\mathtt{x}_{DR}$ be a dotted rule, such that

$$\mathtt{x}_{DR} = \big[[\mathtt{A}_{SYM} \rightarrow \mathtt{pre}_{STR}\,.\,\mathtt{next}_{SYM}\,.\,\mathtt{post}_{STR}], \mathtt{pos}\big],$$

then

$$\mathtt{LHS}(\mathtt{x}_{DR}) \underset{\text{def}}{\equiv} \mathtt{A}_{SYM}$$

$$\mathtt{Postdot}(\mathtt{x}_{DR}) \underset{\text{def}}{\equiv} \begin{cases} \mathtt{next}_{SYM}, & \text{if } \mathtt{x} = [\mathtt{A} \rightarrow \mathtt{pre} \bullet \mathtt{next}\,.\,\mathtt{post}] \\ \Lambda, & \text{if } \mathtt{x} = [\mathtt{A} \rightarrow \mathtt{pre}\,.\,\mathtt{next}\,.\,\mathtt{post} \bullet] \end{cases}$$

$$\mathtt{Next}(\mathtt{x}_{DR}) \underset{\text{def}}{\equiv} \begin{cases} [\mathtt{A} \rightarrow \mathtt{pre}\,.\,\mathtt{next} \bullet \mathtt{post}], \\ \qquad \text{if } \mathtt{Postdot}(\mathtt{x}_{DR}) = \mathtt{next} \\ \Lambda, \text{ otherwise} \end{cases}$$

$$\mathtt{Penult}(\mathtt{x}_{DR}) \underset{\text{def}}{\equiv} \begin{cases} \mathtt{next}_{SYM}, \quad \text{if} \\ \qquad \mathtt{Postdot}(\mathtt{x}_{DR}) = \mathtt{next} \\ \qquad \land \quad \mathtt{post}_{STR} \overset{*}{\Rightarrow} \epsilon \\ \qquad \land \quad \neg(\mathtt{next}_{SYM} \overset{*}{\Rightarrow} \epsilon) \\ \Lambda, \quad \text{otherwise} \end{cases}$$

A **penult** is a dotted rule $\mathtt{d}_{DR}$ such that $\mathtt{Penult}(\mathtt{d}) \neq \Lambda$. Note that $\mathtt{Penult}(\mathtt{x}_{DR})$ is never a nullable symbol. The **initial dotted rule** is

$$\mathtt{initial}_{DR} = [\mathtt{accept}_{SYM} \rightarrow \ \bullet \mathtt{start}_{SYM}].$$

A **predicted dotted rule** is a dotted rule, other than the initial dotted rule, with a dot position of zero, for example,

$$\mathtt{predicted}_{DR} = [\mathtt{A}_{SYM} \rightarrow \ \bullet \mathtt{alpha}_{STR}].$$

A **confirmed dotted rule** is the initial dotted rule, or a dotted rule with a dot position greater than zero. A **completed dotted rule** is a dotted rule with its dot position after the end of its RHS, for example,

$$\mathtt{completed}_{DR} = [\mathtt{A}_{SYM} \rightarrow \mathtt{alpha}_{STR} \bullet].$$

Predicted, confirmed and completed dotted rules are also called, respectively, **predictions**, **confirmations** and **completions**.

A traditional Earley item (type $EIMT$) is a duple

$$[\mathtt{dotted\text{-}rule}_{DR}, \mathtt{x}_{ORIG}]$$

of dotted rule and origin. (The origin is the location where recognition of the rule started. It is sometimes called the "parent".) For convenience, the type $ORIG$ will be a synonym for $LOC$, indicating that the variable designates the origin element of an Earley item.

An Earley parser builds a table of Earley sets,

$$\texttt{table}[\textsc{Earley}, \texttt{i}], \quad \text{where} \quad 0 \le \texttt{i}_{LOC} \le |\texttt{w}| .$$

Earley sets are of type $ES$. Earley sets are often named by their location, so that $\texttt{i}_{ES}$ means the Earley set at $\texttt{i}_{LOC}$. The type designator $ES$ is often omitted to avoid clutter, especially in cases where the Earley set is not named by location.

At points, we will need to compare the Earley sets produced by the different recognizers. $\texttt{table}[\textsc{Recce}, \texttt{i}]$ will be the Earley set at $\texttt{i}_{LOC}$ in the table of Earley sets of the Recce recognizer. For example, $\texttt{table}[\textsc{Marpa}, \texttt{j}]$ will be Earley set $\texttt{j}_{LOC}$ in Marpa's table of Earley sets. In contexts where it is clear which recognizer is intended, $\texttt{table}[\texttt{k}]$, or $\texttt{k}_{ES}$, will symbolize Earley set $\texttt{k}_{LOC}$ in that recognizer's table of Earley sets. If $\texttt{working}_{ES}$ is an Earley set, $|\texttt{working}_{ES}|$ is the number of Earley items in $\texttt{working}_{ES}$.

$\bigl|\texttt{table}[\textsc{Recce}]\bigr|$ is the total number of Earley items in all Earley sets for Recce,

$$\bigl|\texttt{table}[\textsc{Recce}]\bigr| = \sum_{\texttt{i}_{LOC}=0}^{|\texttt{w}|} \bigl|\texttt{table}[\textsc{Recce}, \texttt{i}]\bigr|.$$

For example, $\bigl|\texttt{table}[\textsc{Marpa}]\bigr|$ is the total number of Earley items in all the Earley sets of a Marpa parse.

Recall that there was a unique acceptance symbol, $\texttt{accept}_{SYM}$, in $\texttt{g}$. The input $\texttt{w}$ is accepted if and only if, for some $\texttt{rhs}_{STR}$,

$$\bigl[[\texttt{accept}_{SYM} \rightarrow \texttt{rhs}_{STR} \bullet], 0\bigr] \in \texttt{table}\bigl[|\texttt{w}|\bigr]$$

## 5. Operations of the Earley algorithm

In this section, each Earley operation is shown in the form of an inference rule, the conclusion of which is the set of the Earley items that is that operation's **result**. The Earley sets correspond to parse locations, and for any Earley operation there is a current parse location, $\texttt{current}_{LOC}$, and a current Earley set, $\texttt{current}_{ES}$.

Each location starts with an empty Earley set. For the purposes of this description of Earley, the order of the Earley operations when building an Earley set is non-deterministic. After each Earley operation is performed, its result is unioned with the current Earley set. When no more Earley items can be added, the Earley set is complete. The Earley sets are built in order from 0 to $|\texttt{w}|$.

## 5.1. **Initialization.**

$$\frac{\texttt{current}_{LOC} = 0}{\Big\{ \big[[\texttt{accept}_{SYM} \rightarrow \bullet\, \texttt{start}_{SYM}], 0\big] \Big\}}$$

Earley **initialization** has no operands and only takes place in Earley set 0.

## 5.2. **Scanning.**

$$\frac{\begin{gathered}\texttt{current}_{LOC} > 0 \\ \texttt{previous}_{LOC} = \texttt{current}_{LOC} - 1 \\ \texttt{token}_{SYM} = \texttt{w}\big[\texttt{previous}_{LOC}\big] \\ \texttt{predecessor}_{EIMT} = [\texttt{before}_{DR}, \texttt{predecessor}_{ORIG}] \\ \texttt{predecessor}_{EIMT} \in \texttt{previous}_{ES} \\ \texttt{Postdot}(\texttt{before}_{DR}) = \texttt{token}_{SYM}\end{gathered}}{\big\{[\texttt{Next}(\texttt{before}_{DR}), \texttt{predecessor}_{ORIG}]\big\}}$$

$\texttt{predecessor}_{EIMT}$ and $\texttt{token}_{SYM}$ are the operands of an Earley scan. $\texttt{predecessor}_{EIMT}$ is called the predecessor of the scanning operation. The token, $\texttt{token}_{SYM}$, is the transition symbol of the scanning operation.

## 5.3. **Reduction.**

$$\frac{\begin{gathered}\texttt{component}_{EIMT} = \big[[\texttt{lhs}_{SYM} \rightarrow \texttt{rhs}_{STR} \bullet], \texttt{component-orig}_{LOC}\big] \\ \texttt{component}_{EIMT} \in \texttt{current}_{ES} \\ \texttt{predecessor}_{EIMT} = [\texttt{before}_{DR}, \texttt{predecessor}_{ORIG}] \\ \texttt{predecessor}_{EIMT} \in \texttt{component-orig}_{ES} \\ \texttt{Postdot}(\texttt{before}_{DR}) = \texttt{lhs}_{SYM}\end{gathered}}{\big\{[\texttt{Next}(\texttt{before}_{DR}), \texttt{predecessor}_{ORIG}]\big\}}$$

$\texttt{component}_{EIMT}$ is called the component of the reduction operation.[1] $\texttt{predecessor}_{EIMT}$ is the predecessor of the reduction operation. $\texttt{lhs}_{SYM}$ is the transition symbol of the reduction operation.

$\texttt{predecessor}_{EIMT}$ and $\texttt{component}_{EIMT}$ are the operands of the reduction operation. In some contexts, it is convenient to treat the transition symbol as an operand, so that the operands are $\texttt{predecessor}_{EIMT}$ and $\texttt{lhs}_{SYM}$.

[1]The term "component" comes from Irons [5].

## 5.4. Prediction.

$$\frac{\begin{gathered}\texttt{predecessor}_{EIMT} = [\texttt{predecessor}_{DR}, \texttt{predecessor}_{ORIG}] \\ \texttt{predecessor}_{EIMT} \in \texttt{current}_{ES}\end{gathered}}{\left\{ \begin{array}{l} \Big[ \big[ \texttt{L}_{SYM} \rightarrow \bullet \texttt{rh}_{STR} \big], \texttt{current}_{LOC} \Big] \quad \text{such that} \\ \qquad \big[ \texttt{L}_{SYM} \rightarrow \texttt{rh}_{STR} \big] \in \texttt{rules} \\ \qquad \land \big( \exists \, \texttt{z}_{STR} \mid \texttt{Postdot}(\texttt{predecessor}_{DR}) \overset{*}{\Rightarrow} \texttt{L}_{SYM} \, . \, \texttt{z}_{STR} \big) \end{array} \right\}}$$

A prediction operation can add several Earley items to Earley set $\texttt{current}_{LOC}$. $\texttt{predecessor}_{EIMT}$ is called the predecessor of the prediction operation, and is its only operand.

5.5. **Causation.** An operand of an operation is also called a **cause** of that operation, and the set of operands for an Earley operation is its **causation**.

## 6. The Leo algorithm

In [10], Joop Leo presented a method for dealing with right recursion in $\mathcal{O}(\texttt{n})$ time. Leo shows that, with his modification, Earley's algorithm is $\mathcal{O}(\texttt{n})$ for all LR-regular grammars. (LR-regular is LR where lookahead is infinite length, but restricted to distinguishing between regular expressions.)

Summarizing Leo's method, it consists of spotting potential right recursions and memoizing them. Leo restricts the memoization to situations where the right recursion is unambiguous. Potential right recursions are memoized by Earley set, using what Leo called "transitive items". In this paper Leo's "transitive items" will be called Leo items. Leo items in the form that Marpa uses will be type $LIM$. "Traditional" Leo items, that is, those of the form used in Leo's paper[10], will be type $LIMT$.

In each Earley set, there is at most one Leo item per symbol. A traditional Leo item (LIMT) is the triple

$$[\texttt{top}_{DR}, \texttt{transition}_{SYM}, \texttt{top}_{ORIG}]$$

where $\texttt{transition}_{SYM}$ is the transition symbol, and

$$\texttt{top}_{EIMT} = [\texttt{top}_{DR}, \texttt{top}_{ORIG}]$$

is the Earley item to be added on reductions over $\texttt{transition}_{SYM}$.

Leo items memoize what would otherwise be sequences of Earley items. Leo items only memoize unambiguous (or deterministic) sequences, so that the top of the sequence can represent the entire sequence – the only role the other EIMT's in the sequence play in the parse is to derive the top EIMT. We will call these memoized sequences, Leo sequences.

To quarantee that a Leo sequence is deterministic, Leo enforced **Leo uniqueness**. Define containment of a dotted rule in a Earley set of EIMT's as

$$\begin{aligned}\mathtt{Contains}(\mathtt{i}_{ES}, \mathtt{d}_{DR}) \underset{\text{def}}{\equiv} \exists\, \mathtt{b}_{EIMT}, \mathtt{j}_{ORIG} \mid \\ \mathtt{b}_{EIMT} = [\mathtt{d}_{DR}, \mathtt{j}_{ORIG}] \land \mathtt{b}_{EIMT} \in \mathtt{i}_{ES}.\end{aligned}$$

A dotted rule $\mathtt{d}_{DR}$ is **Leo unique** in the Earley set at $\mathtt{i}_{ES}$ if and only if

$$\begin{aligned}&\mathtt{Penult}(\mathtt{d}_{DR}) \neq \Lambda \\ &\land \forall\, \mathtt{d2}_{DR} \big(\mathtt{Contains}(\mathtt{i}_{ES}, \mathtt{d2}_{DR}) \implies \\ &\qquad \mathtt{Postdot}(\mathtt{d}_{DR}) = \mathtt{Postdot}(\mathtt{d2}_{DR}) \implies \mathtt{d}_{DR} = \mathtt{d2}_{DR}\big).\end{aligned}$$

If $\mathtt{d}_{DR}$ is Leo unique, then the symbol $\mathtt{Postdot}(\mathtt{d}_{DR})$ is also said to be **Leo unique**. In cases where a symbol $\mathtt{transition}_{SYM}$ is Leo unique in $\mathtt{i}_{ES}$, we can speak of the dotted rule for $\mathtt{transition}_{SYM}$, and the rule for $\mathtt{transition}_{SYM}$, because there can be only one of each. In the previous definitions, it is important to emphasize that $\mathtt{d2}_{DR}$ ranges over all the dotted rules of Earley set $\mathtt{i}_{ES}$, even those which are ineligible for Leo memoization.

Let $\mathtt{n}$ be the length of a Leo sequence. In Earley, each such sequence would be expanded in every Earley set that is the origin of an EIMT included in the sequence, and the total number of EIMT's would be $\mathcal{O}(\mathtt{n}^2)$.

With Leo memoization, a single EIMT stands in for the sequence. There are $\mathcal{O}(1)$ Leo items per Earley set, so the cost of the sequence is $\mathcal{O}(1)$ per Earley set, or $\mathcal{O}(\mathtt{n})$ for the entire sequence. If, at evaluation time, it is desirable to expand the Leo sequence, only those items actually involved in the parse need to be expanded. All the EIMT's of a potential right-recursion will be in one Earley set and the number of EIMT's will be $\mathcal{O}(\mathtt{n})$, so that even including expansion of the Leo sequence for evaluation, the time and space complexity of the sequence remains $\mathcal{O}(\mathtt{n})$.

Recall that we call a dotted rule $\mathtt{d}_{DR}$ a **penult** if $\mathtt{Penult}(\mathtt{d}) \neq \Lambda$. In Leo's original algorithm, any penult was treated as a potential right-recursion. Marpa applies the Leo memoizations in more restricted circumstances. For Marpa to consider a dotted rule

$$\mathtt{candidate}_{DR} = [\mathtt{candidate}_{RULE}, \mathtt{i}]$$

for Leo memoization, $\mathtt{candidate}_{DR}$ must be a penult and $\mathtt{candidate}_{RULE}$ must be right-recursive.

By restricting Leo memoization to right-recursive rules, Marpa incurs the cost of Leo memoization only in cases where Leo sequences could be infinitely long. This more careful targeting of the memoization is for efficiency reasons. If all penults are memoized, many memoizations will be performed where the longest potential Leo sequence is short, and the payoff is therefore very limited. One future extension might be to identify non-right-recursive rules which generate Leo sequences long enough to justify inclusion in the Leo memoizations. Such cases are unusual, but may occur.

Omission of a memoization does not affect correctness, so Marpa's restriction of Leo memoization preserves the correctness as shown in Leo[10]. Later in this paper we will show that this change also leaves the complexity results of Leo[10] intact.

Implementing the Leo logic requires adding Leo reduction as a new basic operation, adding a new premise to the Earley reduction operation, and extending the Earley sets to memoize Earley items as LIMT's.

## 6.1. **Leo reduction.**

$$\frac{\begin{array}{c}\texttt{component}_{EIMT} = \big[[\texttt{lhs}_{SYM} \rightarrow \texttt{rhs}_{STR} \bullet], \texttt{component-origin}_{LOC}\big] \\ \texttt{component}_{EIMT} \in \texttt{current}_{ES} \\ \texttt{predecessor}_{LIMT} = [\texttt{top}_{DR}, \texttt{lhs}_{SYM}, \texttt{top}_{ORIG}] \\ \texttt{predecessor}_{LIMT} \in \texttt{component-orig}_{ES}\end{array}}{\big\{[\texttt{top}_{DR}, \texttt{top}_{ORIG}]\big\}}$$

The new Leo reduction operation resembles the Earley reduction operation, except that it looks for an LIMT, instead of a predecessor EIMT. $\texttt{predecessor}_{LIMT}$ and $\texttt{component}_{EIMT}$ are the operands of the Leo reduction operation. $\texttt{lhs}_{SYM}$ is the transition symbol of the Leo reduction. As with Earley reduction, it may be convenient to treat the transition symbol as an operand, so that the operands are $\texttt{predecessor}_{LIMT}$ and $\texttt{lhs}_{SYM}$.

## 6.2. **Changes to Earley reduction.**

Earley reduction still applies, with an additional premise:

$$\begin{aligned} \neg\exists\, \texttt{x}_{LIMT} \;\mid\; & \texttt{x}_{LIMT} \in \texttt{component-orig}_{ES} \\ & \wedge\, \texttt{x}_{LIMT} = [\texttt{x}_{DR}, \texttt{lhs}_{SYM}, \texttt{x}_{ORIG}] \end{aligned}$$

The additional premise prevents Earley reduction from being applied where there is an LIMT with $\texttt{lhs}_{SYM}$ as its transition symbol. This reflects the fact that Leo reduction replaces Earley reduction if and only if there is a Leo memoization.

6.3. **Leo memoization.** We define uniqueness of a penult in an Earley set as

$$\begin{aligned}
&\texttt{Penult-Unique}(\texttt{penult}_{SYM}, \texttt{i}_{ES}) \underset{\text{def}}{\equiv} \\
&\qquad \forall\, \texttt{y}_{DR} \big(\texttt{Contains}(\texttt{current}_{ES}, \texttt{y}_{DR}) \land \texttt{penult}_{SYM} = \texttt{Penult}(\texttt{y}_{DR})\big) \\
&\qquad\qquad \implies \texttt{x}_{DR} = \texttt{y}_{DR}.
\end{aligned}$$

We define Leo uniqueness as

$$\begin{aligned}
&\texttt{Leo-Unique}(\texttt{x}_{DR}, \texttt{current}_{LOC}) \underset{\text{def}}{\equiv} \texttt{Contains}(\texttt{current}_{ES}, \texttt{x}_{DR}) \\
&\qquad \land \texttt{Penult}(\texttt{x}_{DR}) \neq \Lambda \\
&\qquad \land \texttt{Penult-Unique}(\texttt{Penult}(\texttt{x}_{DR}), \texttt{current}_{ES})
\end{aligned}$$

and Leo eligibility as

$$\begin{aligned}
&\texttt{Leo-Eligible}(\texttt{x}_{DR}, \texttt{current}_{LOC}) \underset{\text{def}}{\equiv} \\
&\qquad \exists\, \texttt{x}_{RULE}, \texttt{x}_{ORIG} \;\mid\; \texttt{x}_{DR} = [\texttt{x}_{RULE}, \texttt{i}_{ORIG}] \\
&\qquad \land \texttt{Right-Recursive}(\texttt{x}_{RULE}) \\
&\qquad \land \texttt{Leo-Unique}(\texttt{current}_{ES}, \texttt{x}_{DR}).
\end{aligned}$$

For convenience, we define a relation that is true if $\texttt{pred}_{LIMT}$ is the LIMT predecessor of an EIMT, and false otherwise:

$$\begin{aligned}
&\texttt{LIMT-Predecessor}(\texttt{pred}_{LIMT}, \texttt{bottom}_{EIMT}) \underset{\text{def}}{\equiv} \\
&\quad \exists \texttt{bottom-origin}_{ES}, \texttt{bottom}_{DR}, \texttt{pred}_{DR}, \\
&\qquad \texttt{pred-origin}_{LOC}, \texttt{bottom-origin}_{LOC} \quad \text{such that} \\
&\qquad\quad \texttt{bottom}_{EIMT} = [\texttt{bottom}_{DR}, \texttt{bottom-origin}_{LOC}] \\
&\qquad\quad \land \texttt{pred}_{LIMT} = [\texttt{pred}_{DR}, \texttt{LHS}(\texttt{bottom}_{DR}), \texttt{pred-origin}_{LOC}] \\
&\qquad\quad \land \texttt{pred}_{LIMT} \in \texttt{bottom-origin}_{ES}
\end{aligned}$$

We are now ready to define an inference rule which holds if a LIMT predecessor can be found for an EIMT $\texttt{bottom}_{EIMT}$

$$\frac{\begin{array}{c}
\texttt{LIMT-Predecessor}(\texttt{pred}_{LIMT}, \texttt{bottom}_{EIMT}) \\
\texttt{pred}_{LIMT} = [\texttt{pred}_{DR}, \texttt{LHS}(\texttt{bottom}_{DR}), \texttt{pred}_{ORIG}] \\
\texttt{bottom}_{EIMT} = [\texttt{bottom}_{DR}, \texttt{bottom}_{LOC}] \\
\texttt{bottom}_{EIMT} \in \texttt{current}_{ES} \\
\texttt{Leo-Eligible}(\texttt{bottom}_{DR})
\end{array}}{\{[\texttt{pred}_{DR}, \texttt{Penult}(\texttt{bottom}_{DR}), \texttt{pred}_{ORIG}]\}}$$

and another, which holds if $\texttt{bottom}_{EIMT}$ has no predecessor LIMT,

$$\frac{\begin{array}{c}\neg\,\texttt{LIMT-Predecessor}(\texttt{pred}_{LIMT},\texttt{bottom}_{EIMT})\\ \texttt{bottom}_{EIMT} = [\texttt{bottom}_{DR},\texttt{bottom}_{ORIG}]\\ \texttt{bottom}_{EIMT} \in \texttt{current}_{ES}\\ \texttt{Leo-Eligible}(\texttt{bottom}_{DR})\end{array}}{\big\{[\texttt{Next}(\texttt{bottom}_{DR}),\texttt{Penult}(\texttt{bottom}_{DR}),\texttt{bottom}_{ORIG}]\big\}}$$

## 7. The Aycock-Horspool finite automaton

In this paper a "split LR(0) $\epsilon$-DFA" as described by Aycock and Horspool[2], will be called an Aycock-Horspool Finite Automaton, or AHFA. This section will summarize the ideas from [2] that are central to Marpa.

Aycock and Horspool based their AHFA's on a few observations.

- In practice, Earley items sharing the same origin, but having different dotted rules, often appear together in the same Earley set.
- There is in the literature a method for associating groups of dotted rules that often appear together when parsing. This method is the LR(0) DFA used in the much-studied LALR and LR parsers.
- The LR(0) items that are the components of LR(0) states are, exactly, dotted rules.
- By taking into account symbols that derive the null string, the LR(0) DFA could be turned into an LR(0) $\epsilon$-DFA, which would be even more effective at grouping dotted rules that often occur together into a single DFA state.

AHFA states are, in effect, a shorthand for groups of dotted rules that occur together frequently. Aycock and Horspool realized that, by changing Earley items to track AHFA states instead of individual dotted rules, the size of Earley sets could be reduced, and Earley's algorithm made faster in practice.

As a reminder, the original Earley items (EIMT's) were duples, $[\texttt{x}_{DR},\texttt{x}_{ORIG}]$, where $\texttt{x}_{DR}$ is a dotted rule. An Aycock-Horspool Earley item is a duple

$$[\texttt{y}_{AH},\texttt{y}_{ORIG}],$$

where $\texttt{y}_{AH}$ is an AHFA state.

Marpa uses Earley items of the form created by Aycock and Horspool. A Marpa Earley item has type $EIM$, and a Marpa Earley item is often referred to as an EIM.

Aycock and Horspool did not consider Leo's modifications, but Marpa incorporates them, and Marpa also changes its Leo items to use AHFA states. Marpa's Leo items (LIM's) are triples of the form

$$[\mathtt{top}_{AH}, \mathtt{transition}_{SYM}, \mathtt{top}_{ORIG}],$$

where $\mathtt{transition}_{SYM}$ and $\mathtt{top}_{ORIG}$ are as in the traditional Leo items, and $\mathtt{top}_{AH}$ is an AHFA state. A Marpa Leo item has type $LIM$.

[2] also defines a partial transition function for pairs of AHFA state and symbol,

$$\mathtt{GOTO} : \mathtt{fa}, (\epsilon \cup \mathtt{vocab}) \mapsto \mathtt{fa}.$$

$\mathtt{GOTO}(\mathtt{from}_{AH}, \epsilon)$ is a **null transition**. (AHFA's are not fully deterministic.) If $\mathtt{predicted}_{AH}$ is the result of a null transition, it is called a **predicted** AHFA state. If an AHFA state is not a **predicted** AFHA state, it is called a **confirmed** AHFA state. The initial AHFA state is a confirmed AHFA state. (In [2] confirmed states are called "kernel states", and predicted states are called "non-kernel states".)

The states of an AHFA are not a partition of the dotted rules – a single dotted rule can occur in more than one AHFA state. In combining the improvements of Leo [10] and Aycock and Horspool[2], the following theorem is crucial.

**Theorem 7.1.** *If a Marpa Earley item (EIM) is the result of a Leo reduction, then its AHFA state contains only one dotted rule.*

*Proof.* Let the EIM that is the result of the Leo reduction be

$$\mathtt{result}_{EIM} = [\mathtt{result}_{AH}, \mathtt{result}_{ORIG}]$$

Let the Earley set that contains $\mathtt{result}_{EIM}$ be $\mathtt{i}_{ES}$. Since $\mathtt{result}_{EIM}$ is the result of a Leo reduction we know, from the definition of a Leo reduction, that

$$\mathtt{complete}_{DR} \in \mathtt{result}_{AH}$$

where $\mathtt{complete}_{DR}$ is a completed rule. Let $\mathtt{c}_{RULE}$ be the rule of $\mathtt{complete}_{DR}$, and $\mathtt{cp}$ its dot position,

$$\mathtt{complete}_{DR} = [\mathtt{c}_{RULE}, \mathtt{cp}].$$

$\mathtt{cp} > 0$ because, in Marpa grammars, completions are never predictions.

Suppose, for a reduction to absurdity, that the AHFA state contains another dotted rule, $\mathtt{other}_{DR}$, that is, that

$$\mathtt{other}_{DR} \in \mathtt{result}_{AH},$$

where $\texttt{complete}_{DR} \neq \texttt{other}_{DR}$. Let $\texttt{o}_{RULE}$ be the rule of $\texttt{other}_{DR}$, and $\texttt{op}$ its dot position,

$$\texttt{other}_{DR} = [\texttt{o}_{RULE}, \texttt{op}].$$

AHFA construction never places a prediction in the same AHFA state as a completion, so $\texttt{other}_{DR}$ is not a prediction. Therefore, $\texttt{op} > 0$. To show this outer reduction to absurdity, we first prove by a first inner reductio that $\texttt{c}_{RULE} \neq \texttt{o}_{RULE}$, then by a second inner reductio that $\texttt{c}_{RULE} = \texttt{o}_{RULE}$.

Assume, for the first inner reductio, that $\texttt{c}_{RULE} = \texttt{o}_{RULE}$. By the construction of an AHFA state, both $\texttt{complete}_{DR}$ and $\texttt{other}_{DR}$ resulted from the same series of transitions. But the same series of transitions over the same rule would result in the same dot position, $\texttt{cp} = \texttt{op}$, so that if $\texttt{c}_{RULE} = \texttt{o}_{RULE}$, $\texttt{complete}_{DR} = \texttt{other}_{DR}$, which is contrary to the assumption for the outer reductio. This shows the first inner reductio.

Next, we assume for the second inner reductio that $\texttt{c}_{RULE} \neq \texttt{o}_{RULE}$. Since both $\texttt{complete}_{DR}$ and $\texttt{other}_{DR}$ are in the same EIM and neither is a prediction, both must result from transitions, and their transitions must have been from the same Earley set. Since they are in the same AHFA state, by the AHFA construction, that transition must have been over the same transition symbol, call it $\texttt{transition}_{SYM}$. But Leo uniqueness applies to $\texttt{complete}_{DR}$, and requires that the transition over $\texttt{transition}_{SYM}$ be unique in $\texttt{i}_{ES}$.

But if $\texttt{c}_{RULE} \neq \texttt{o}_{RULE}$, $\texttt{transition}_{SYM}$ was the transition symbol of two different dotted rules, and the Leo uniqueness requirement does not hold. The conclusion that the Leo uniqueness requirement both does and does not hold is a contradiction, which shows the second inner reductio. Since the assumption for the second inner reductio was that $\texttt{c}_{RULE} \neq \texttt{o}_{RULE}$, we conclude that $\texttt{c}_{RULE} = \texttt{o}_{RULE}$.

By the two inner reductio's, we have both $\texttt{c}_{RULE} \neq \texttt{o}_{RULE}$ and $\texttt{c}_{RULE} = \texttt{o}_{RULE}$, which completes the outer reduction to absurdity. For the outer reductio, we assumed that $\texttt{other}_{DR}$ was a second dotted rule in $\texttt{result}_{AH}$, such that $\texttt{other}_{DR} \neq \texttt{complete}_{DR}$. We can therefore conclude that

$$\texttt{other}_{DR} \in \texttt{result}_{AH} \implies \texttt{other}_{DR} = \texttt{complete}_{DR}.$$

If $\texttt{complete}_{DR}$ is a dotted rule in the AHFA state of a Leo reduction EIM, then it must be the only dotted rule in that AHFA state. $\square$

## 8. The Marpa Recognizer

### 8.1. **Complexity.**

Alongside the pseudocode of this section are observations about its space and time complexity. In what follows, we will charge all time and space resources to Earley items, or to attempts to add Earley items. We will show that, to each Earley item actually added, or to each attempt to add a duplicate Earley item, we can charge amortized $\mathcal{O}(1)$ time and space.

At points, it will not be immediately convenient to speak of charging a resource to an Earley item or to an attempt to add a duplicate Earley item. In those circumstances, we speak of charging time and space

- to the parse; or
- to the Earley set; or
- to the current procedure's caller.

We can charge time and space to the parse itself, as long as the total time and space charged is $\mathcal{O}(1)$. Afterwards, this resource can be recharged to the initial Earley item, which is present in all parses. Soft and hard failures of the recognizer use worst-case $\mathcal{O}(1)$ resource, and are charged to the parse.

We can charge resources to the Earley set, as long as the time or space is $\mathcal{O}(1)$. Afterwards, the resource charged to the Earley set can be re-charged to an arbitrary member of the Earley set, for example, the first. If an Earley set is empty, the parse must fail, and the time can be charged to the parse.

In a procedure, resource can be "caller-included". Caller-included resource is not accounted for in the current procedure, but passed upward to the procedure's caller, to be accounted for there. A procedure to which caller-included resource is passed will sometimes pass the resource upward to its own caller, although of course the top-level procedure does not do this.

For each procedure, we will state whether the time and space we are charging is inclusive or exclusive. The exclusive time or space of a procedure is that which it uses directly, ignoring resource charges passed up from called procedures. Inclusive time or space includes resource passed upward to the current procedure from called procedures.

Earley sets may be represented by $\texttt{i}_{ES}$, where $\texttt{i}$ is the Earley set's locaiton $\texttt{i}_{LOC}$. The two notations should be regarded as interchangeable. The actual implementation of either should be the equivalent of a pointer to a data structure containing, at a minium, the Earley items, a memoization of the Earley set's location as an integer, and a per-set-list. Per-set-lists will be described in Section 8.12.

**Algorithm 1** Marpa Top-level

```
procedure MAIN
    INITIAL
    for i, $0 \le i \le |w|$ do
                        ▷ At this point, $x_{ES}$ is complete, for $0 \le x < i$
        SCAN PASS(i, w[i − 1])
        if $|i_{ES}| = 0$ then
            reject w and return
        end if
        REDUCTION PASS(i)
    end for
    for every $[x_{AH}, 0] \in table[|w|]$ do
        if $accept_{DR} \in x_{AH}$ then
            accept w and return
        end if
    end for
    reject w
end procedure
```

8.2. **Top-level code.** Exclusive time and space for the loop over the Earley sets is charged to the Earley sets. Inclusive time and space for the final loop to check for $\texttt{accept}_{DR}$ is charged to the Earley items at location $|\texttt{w}|$. Overhead is charged to the parse. All these resource charges are obviously $\mathcal{O}(1)$.

8.3. **Ruby Slippers parsing.** This top-level code represents a significant change from AH. SCAN PASS and REDUCTION PASS are separated. As a result, when the scanning of tokens that start at location $\texttt{i}_{LOC}$ begins, the Earley sets for all locations prior to $\texttt{i}_{LOC}$ are complete. This means that the scanning operation has available, in the Earley sets, full information about the current state of the parse, including which tokens are acceptable during the scanning phase.

**Algorithm 2** Initialization

```
procedure INITIAL
    ADD EIM PAIR($0_{ES}, start_{AH}, 0$)
end procedure
```

8.4. **Initialization.** Inclusive time and space is $\mathcal{O}(1)$ and is charged to the parse.

**Algorithm 3** Marpa Scan pass

```
procedure SCAN PASS($\mathtt{i}_{LOC}, \mathtt{a}_{SYM}$)
    Note: Each pass through this loop is an EIM attempt
    for each $\mathtt{predecessor}_{EIM} \in \mathtt{transitions}((\mathtt{i} - 1), \mathtt{a})$ do
        $[\mathtt{from}_{AH}, \mathtt{origin}_{LOC}] \leftarrow \mathtt{predecessor}_{EIM}$
        $\mathtt{to}_{AH} \leftarrow \mathtt{GOTO}(\mathtt{from}_{AH}, \mathtt{a}_{SYM})$
        ADD EIM PAIR($\mathtt{i}_{ES}, \mathtt{to}_{AH}, \mathtt{origin}_{LOC}$)
    end for
end procedure
```

8.5. **Scan pass.** `transitions` is a set of tables, one per Earley set. The tables in the set are indexed by symbol. Symbol indexing is $\mathcal{O}(1)$, since the number of symbols is a constant, but since the number of Earley sets grows with the length of the parse, it cannot be assumed that Earley sets can be indexed by location in $\mathcal{O}(1)$ time. For the operation $\mathtt{transitions}(\mathtt{l}_{LOC}, \mathtt{s}_{SYM})$ to be in $\mathcal{O}(1)$ time, $\mathtt{l}_{LOC}$ must represent a link directly to the Earley set. In the case of scanning, the lookup is always in the previous Earley set, which can easily be tracked in $\mathcal{O}(1)$ space and retrieved in $\mathcal{O}(1)$ time. Inclusive time and space can be charged to the $\mathtt{predecessor}_{EIM}$. Overhead is charged to the Earley set at $\mathtt{i}_{LOC}$.

**Algorithm 4** Reduction pass

```
procedure REDUCTION PASS($\mathtt{i}_{LOC}$)
    Note: $\mathtt{table}[\mathtt{i}]$ may include EIM's added by
          by REDUCE ONE LHS and
          the loop must traverse these
    for each Earley item $\mathtt{work}_{EIM} \in \mathtt{table}[\mathtt{i}]$ do
        $[\mathtt{work}_{AH}, \mathtt{origin}_{LOC}] \leftarrow \mathtt{work}_{EIM}$
        $\mathtt{lh\text{-}sides}_{\{SYM\}} \leftarrow$ a set containing the LHS
            of every completed rule in $\mathtt{work}_{EIM}$
        for each $\mathtt{lhs}_{SYM} \in \mathtt{lh\text{-}sides}_{\{SYM\}}$ do
            REDUCE ONE LHS($\mathtt{i}_{LOC}$, $\mathtt{origin}_{LOC}$, $\mathtt{lhs}_{SYM}$)
        end for
    end for
    MEMOIZE TRANSITIONS($\mathtt{i}_{LOC}$)
end procedure
```

8.6. **Reduction pass.** The loop over `table`[`i`] must also include any items added by REDUCE ONE LHS. This can be done by implementing `table`[`i`] as an ordered set and adding new items at the end.

Exclusive time is clearly $\mathcal{O}(1)$ per $\texttt{work}_{EIM}$, and is charged to the $\texttt{work}_{EIM}$. Additionally, some of the time required by Reduce one LHS is caller-included, and therefore charged to this procedure. Inclusive time from Reduce one LHS is $\mathcal{O}(1)$ per call, as will be seen in section 8.8, and is charged to the $\texttt{work}_{EIM}$ that is current during that call to Reduce one LHS. Overhead may be charged to the Earley set at $\texttt{i}_{LOC}$.

**Algorithm 5** Memoize transitions

```
1:  procedure Memoize transitions($\texttt{i}_{LOC}$)
2:      for every $\texttt{postdot}_{SYM}$, a postdot symbol of $\texttt{i}_{ES}$ do
3:          Note: $\texttt{postdot}_{SYM}$ is “Leo eligible” if it is
4:              Leo unique and its rule is right recursive
5:          if $\texttt{postdot}_{SYM}$ is Leo eligible then
6:              Set $\texttt{transitions}(\texttt{i}_{LOC}, \texttt{postdot}_{SYM})$
7:                  to a LIM
8:          else
9:              Set $\texttt{transitions}(\texttt{i}_{LOC}, \texttt{postdot}_{SYM})$
10:                 to the set of EIM’s that have
11:                 $\texttt{postdot}_{SYM}$ as their postdot symbol
12:         end if
13:     end for
14: end procedure
```

8.7. **Memoize transitions.** The $\texttt{transitions}$ table for $\texttt{i}_{ES}$ is built once all EIMs have been added to $\texttt{i}_{ES}$. We first look at the resource, excluding the processing of Leo items. The non-Leo processing can be done in a single pass over $\texttt{i}_{ES}$, in $\mathcal{O}(1)$ time per EIM. Inclusive time and space are charged to the Earley items being examined. Overhead is charged to $\texttt{i}_{ES}$.

We now look at the resource used in the Leo processing. A transition symbol $\texttt{transition}_{SYM}$ is Leo eligible if it is Leo unique and its rule is right recursive. (If $\texttt{transition}_{SYM}$ is Leo unique in $\texttt{i}_{ES}$, it will be the postdot symbol of only one rule in $\texttt{i}_{ES}$.) All but one of the determinations needed to decide if $\texttt{transition}_{SYM}$ is Leo eligible can be precomputed from the grammar, and the resource to do this is charged to the parse. The precomputation, for example, for every rule, determines if it is right recursive.

One part of the test for Leo eligibility cannot be done as a precomputation. This is the determination whether there is only one dotted rule in $\texttt{i}_{ES}$ whose postdot symbol is $\texttt{transition}_{SYM}$. This can be done in a single pass over the EIM's of $\texttt{i}_{ES}$ that notes the postdot symbols as they are encountered and whether any is enountered twice. The time and space, including that for the creation of a LIM if necessary, will be $\mathcal{O}(1)$ time per EIM examined, and can be charged to EIM being examined.

**Algorithm 6** Reduce one LHS symbol

```
1: procedure REDUCE ONE LHS($\texttt{i}_{LOC}$, $\texttt{origin}_{LOC}$, $\texttt{lhs}_{SYM}$)
2:     Note: Each pass through this loop is an EIM attempt
3:     for each pim $\in$ $\texttt{transitions}(\texttt{origin}_{LOC}, \texttt{lhs}_{SYM})$ do
4:                 ▷ pim is a "postdot item", either a LIM or an EIM
5:         if pim is a LIM, $\texttt{pim}_{LIM}$ then
6:             Perform a LEO REDUCTION OPERATION
7:                 for operands $\texttt{i}_{LOC}$, $\texttt{pim}_{LIM}$
8:         else
9:             Perform a EARLEY REDUCTION OPERATION
10:                for operands $\texttt{i}_{LOC}$, $\texttt{pim}_{EIM}$, $\texttt{lhs}_{SYM}$
11:        end if
12:    end for
13: end procedure
```

8.8. **Reduce one LHS.** To show that

$$\texttt{transitions}(\texttt{origin}_{LOC}, \texttt{lhs}_{SYM})$$

can be traversed in $\mathcal{O}(1)$ time, we note that the number of symbols is a constant and assume that $\texttt{origin}_{LOC}$ is implemented as a link back to the Earley set, rather than as an integer index. This requires that $\texttt{work}_{EIM}$ in REDUCTION PASS carry a link back to its origin. As implemented, Marpa's Earley items have such links.

Inclusive time for the loop over the EIM attempts is charged to each EIM attempt. Overhead is $\mathcal{O}(1)$ and caller-included.

**Algorithm 7** Earley reduction operation

```
1: procedure EARLEY REDUCTION OPERATION($\texttt{i}_{LOC}$, $\texttt{from}_{EIM}$,
   $\texttt{trans}_{SYM}$)
2:     $[\texttt{from}_{AH}, \texttt{origin}_{LOC}] \leftarrow \texttt{from}_{EIM}$
3:     $\texttt{to}_{AH} \leftarrow \texttt{GOTO}(\texttt{from}_{AH}, \texttt{trans}_{SYM})$
4:     ADD EIM PAIR($\texttt{i}_{ES}$, $\texttt{to}_{AH}$, $\texttt{origin}_{LOC}$)
5: end procedure
```

8.9. **Earley Reduction operation.** Exclusive time and space is clearly $\mathcal{O}(1)$. EARLEY REDUCTION OPERATION is always called as part of an EIM attempt, and inclusive time and space is charged to the EIM attempt.

**Algorithm 8** Leo reduction operation

```
1: procedure LEO REDUCTION OPERATION($\texttt{i}_{LOC}$, $\texttt{from}_{LIM}$)
2:     $[\texttt{from}_{AH}, \texttt{trans}_{SYM}, \texttt{origin}_{LOC}] \leftarrow \texttt{from}_{LIM}$
3:     $\texttt{to}_{AH} \leftarrow \texttt{GOTO}(\texttt{from}_{AH}, \texttt{trans}_{SYM})$
4:     ADD EIM PAIR($\texttt{i}_{ES}$, $\texttt{to}_{AH}$, $\texttt{origin}_{LOC}$)
5: end procedure
```

8.10. **Leo reduction operation.** Exclusive time and space is clearly $\mathcal{O}(1)$. LEO REDUCTION OPERATION is always called as part of an EIM attempt, and inclusive time and space is charged to the EIM attempt.

**Algorithm 9** Add EIM pair

```
1: procedure ADD EIM PAIR($\texttt{i}_{ES}, \texttt{confirmed}_{AH}, \texttt{origin}_{LOC}$)
2:     $\texttt{confirmed}_{EIM} \leftarrow [\texttt{confirmed}_{AH}, \texttt{origin}_{LOC}]$
3:     $\texttt{predicted}_{AH} \leftarrow \texttt{GOTO}(\texttt{confirmed}_{AH}, \epsilon)$
4:     if $\texttt{confirmed}_{EIM}$ is new then
5:         Add $\texttt{confirmed}_{EIM}$ to $\texttt{table}[\texttt{i}]$
6:     end if
7:     if $\texttt{predicted}_{AH} \neq \Lambda$ then
8:         $\texttt{predicted}_{EIM} \leftarrow [\texttt{predicted}_{AH}, \texttt{i}_{LOC}]$
9:         if $\texttt{predicted}_{EIM}$ is new then
10:            Add $\texttt{predicted}_{EIM}$ to $\texttt{table}[\texttt{i}]$
11:        end if
12:    end if
13: end procedure
```

8.11. **Adding a pair of Earley items.** This operation adds a confirmed EIM item and, if it exists, the EIM for its null-transition. Inclusive time and space is charged to the calling procedure. Trivially, the space is $\mathcal{O}(1)$ per call.

We show that time is also $\mathcal{O}(1)$ by singling out the two non-trivial cases: checking that an Earley item is new, and adding it to the Earley set. MARPA checks whether an Earley item is new in $\mathcal{O}(1)$ time by using a data structure called a PSL. PSL's are the subject of Section 8.12. An Earley item can be added to the current set in $\mathcal{O}(1)$ time if

Earley set is seen as a linked list, to the head of which the new Earley item is added.

The resource used by Add EIM Pair is always caller-included. No time or space is ever charged to a predicted Earley item. At most one attempt to add a $\texttt{predicted}_{EIM}$ will be made per attempt to add a $\texttt{confirmed}_{EIM}$, so that the total resource charged remains $\mathcal{O}(1)$.

8.12. **Per-set lists.** In the general case, where $\texttt{x}$ is an arbitrary datum, it is not possible to use duple $[\texttt{i}_{ES}, x]$ as a search key and expect the search to use $\mathcal{O}(1)$ time. Within Marpa, however, there are specific cases where it is desirable to do exactly that. This is accomplished by taking advantage of special properties of the search.

If it can be arranged that there is a link direct to the Earley set $\texttt{i}_{ES}$, and that $0 \le \texttt{x} < \texttt{c}$, where $\texttt{c}$ is a constant of reasonable size, then a search can be made in $\mathcal{O}(1)$ time, using a data structure called a PSL. Data structures identical to or very similar to PSL's are briefly outlined in both [3, p. 97] and [1, Vol. 1, pages 326-327]. But neither source gives them a name. The term PSL ("per-Earley set list") is new with this paper.

A PSL is a fixed-length array of integers, indexed by an integer, and kept as part of each Earley set. While Marpa is building a new Earley set, $\texttt{j}_{ES}$, the PSL for every previous Earley set, $\texttt{i}_{LOC}$, tracks the Earley items in $\texttt{j}_{ES}$ that have $\texttt{i}_{LOC}$ as their origin. The maximum number of Earley items that must be tracked in each PSL is the number of AHFA states, $|\texttt{fa}|$, which is a constant of reasonable size that depends on $\texttt{g}$.

It would take more than $\mathcal{O}(1)$ time to clear and rebuild the PSL's each time that a new Earley set is started. This overhead is avoided by "time-stamping" each PSL entry with the Earley set that was current when that PSL entry was last updated.

As before, where $\texttt{i}_{ES}$ is an Earley set, let $\texttt{i}_{LOC}$ be its location, and vice versa. $\texttt{i}_{LOC}$ is an integer which is assigned as Earley sets are created. Let $\texttt{ID}(\texttt{x}_{AH})$ be the integer ID of an AHFA state. Numbering the AHFA states from 0 on up as they are created is an easy way to create $\texttt{ID}(\texttt{x}_{AH})$. Let $\texttt{PSL}[\texttt{x}_{ES}][\texttt{y}]$ be the entry for integer $\texttt{y}$ in the PSL in the Earley set at $\texttt{x}_{LOC}$.

Consider the case where Marpa is building $\texttt{j}_{ES}$ and wants to check whether Earley item $\texttt{x}_{EIM}$ is new, where $\texttt{x}_{EIM} = [\texttt{x}_{AH}, \texttt{x}_{ORIG}]$. To check if $\texttt{x}_{EIM}$ is new, Marpa checks

$$\texttt{time-stamp} = \texttt{PSL}[\texttt{x}_{ES}][\texttt{ID}(\texttt{x}_{AH})]$$

If the entry has never been used, we assume that $\texttt{time-stamp} = \Lambda$. If $\texttt{time-stamp} \neq \Lambda \wedge \texttt{time-stamp} = \texttt{j}_{LOC}$, then $\texttt{x}_{EIM}$ is not new, and will not be added to the Earley set.

If $\texttt{p}_{LOC} = \Lambda \vee \texttt{time-stamp} \neq \texttt{j}_{LOC}$, then $\texttt{x}_{EIM}$ is new. $\texttt{x}_{EIM}$ is added to the Earley set, and a new time-stamp is set, as follow:

$$\texttt{PSL}[\texttt{x}_{ES}][\texttt{ID}(\texttt{x}_{AH})] \leftarrow \texttt{j}_{LOC}.$$

8.13. **Complexity summary.** For convenience, we collect and summarize here some of the observations of this section.

*Observation* 8.1. The time and space charged to an Earley item which is actually added to the Earley sets is $\mathcal{O}(1)$.

*Observation* 8.2. The time charged to an attempt to add a duplicate Earley item to the Earley sets is $\mathcal{O}(1)$.

For evaluation purposes, MARPA adds a link to each EIM that records each attempt to add that EIM, whether originally or as a duplicate. Traditionally, complexity results treat parsers as recognizers, and such costs are ignored. This will be an issue when the space complexity for unambiguous grammars is considered.

*Observation* 8.3. The space charged to an attempt to add a duplicate Earley item to the Earley sets is $\mathcal{O}(1)$ if links are included, zero otherwise.

As noted in Section 8.11, the time and space used by predicted Earley items and attempts to add them is charged elsewhere.

*Observation* 8.4. No space or time is charged to predicted Earley items, or to attempts to add predicted Earley items.

## 9. Preliminaries to the theoretical results

9.1. **Nulling symbols.** Recall that Marpa grammars, without loss of generality, contain neither empty rules or properly nullable symbols. This corresponds directly to a grammar rewrite in the MARPA implementation, and its reversal during MARPA's evaluation phase. For the correctness and complexity proofs in this paper, we assume an additional rewrite, this time to eliminate nulling symbols.

Elimination of nulling symbols is also without loss of generality, as can be seen if we assume that a history of the rewrite is kept, and that the rewrite is reversed after the parse. Clearly, whether a grammar $\texttt{g}$ accepts an input $\texttt{w}$ will not depend on the nulling symbols in its rules.

In its implementation, MARPA does not directly rewrite the grammar to eliminate nulling symbols. But nulling symbols are ignored in creating the AHFA states, and must be restored during MARPA's evaluation phase, so that the implementation and this simplification for theory purposes track each other closely.

## 9.2. Comparing Earley items.

**Definition.** A Marpa Earley item **corresponds** to a traditional Earley item $\texttt{x}_{EIMT} = [\texttt{x}_{DR}, \texttt{x}_{ORIG}]$ if and only if the Marpa Earley item is a $\texttt{y}_{EIM} = [\texttt{y}_{AH}, \texttt{x}_{ORIG}]$ such that $\texttt{x}_{DR} \in \texttt{y}_{AH}$. A traditional Earley item, $\texttt{x}_{EIMT}$, corresponds to a Marpa Earley item, $\texttt{y}_{EIM}$, if and only if $\texttt{y}_{EIM}$ corresponds to $\texttt{x}_{EIMT}$.

**Definition.** A set of EIM's is **consistent** with respect to a set of EIMT's, if and only if each of the EIM's in the first set corresponds to at least one of the EIMT's in the second set. A Marpa Earley set $\texttt{table}[\text{MARPA}, \texttt{i}]$ is **consistent** if and only if all of its EIM's correspond to EIMT's in $\texttt{table}[\text{LEO}, \texttt{i}]$.

**Definition.** A set of EIM's is **complete** with respect to a set of EIMT's, if and only if for every EIMT in the second set, there is a corresponding EIM in the first set. A Marpa Earley set $\texttt{table}[\text{MARPA}, \texttt{i}]$ is **complete** if and only if for every traditional Earley item in $\texttt{table}[\text{LEO}, \texttt{i}]$ there is a corresponding Earley item in $\texttt{table}[\text{MARPA}, \texttt{i}]$.

**Definition.** A Marpa Earley set is **correct** if and only that Marpa Earley set is complete and consistent.

## 9.3. About AHFA states.

Several facts from [2] will be heavily used in the following proofs. For convenience, they are restated here.

*Observation* 9.1. Every dotted rule is an element of one or more AHFA states, that is,

$$\forall \, \texttt{x}_{DR} \, \exists \, \texttt{y}_{AH} \mid \texttt{x}_{DR} \in \texttt{y}_{AH}.$$

*Observation* 9.2. AHFA confirmation is consistent with respect to the dotted rules. That is, for all $\texttt{from}_{AH}$, $\texttt{t}_{SYM}$, $\texttt{to}_{AH}$, $\texttt{to}_{DR}$ such that

$$\begin{aligned} & \texttt{GOTO}(\texttt{from}_{AH}, \texttt{t}_{SYM}) = \texttt{to}_{AH} \\ \wedge \quad & \texttt{to}_{DR} \in \texttt{to}_{AH}, \end{aligned}$$

there exists $\texttt{from}_{DR}$ such that

$$\begin{aligned} & \texttt{from}_{DR} \in \texttt{from}_{AH} \\ \wedge \quad & \texttt{t}_{SYM} = \texttt{Postdot}(\texttt{from}_{DR}) \\ \wedge \quad & \texttt{Next}(\texttt{from}_{DR}) = \texttt{to}_{DR}. \end{aligned}$$

*Observation* 9.3. AHFA confirmation is complete with respect to the dotted rules. That is, for all $\texttt{from}_{AH}$, $\texttt{t}_{SYM}$, $\texttt{from}_{DR}$, $\texttt{to}_{DR}$ if

$$\begin{aligned} & \texttt{from}_{DR} \in \texttt{from}_{AH} \\ \wedge \quad & \texttt{Postdot}(\texttt{from}_{DR}) = \texttt{t}_{SYM}, \\ \wedge \quad & \texttt{Next}(\texttt{from}_{DR}) = \texttt{to}_{DR} \end{aligned}$$

then there exists $\texttt{to}_{AH}$ such that

$$\begin{aligned} & \texttt{GOTO}(\texttt{from}_{AH}, \texttt{t}_{SYM}) = \texttt{to}_{AH} \\ \wedge \quad & \texttt{to}_{DR} \in \texttt{to}_{AH}. \end{aligned}$$

*Observation* 9.4. AHFA prediction is consistent with respect to the dotted rules. That is, for all $\texttt{from}_{AH}$, $\texttt{to}_{AH}$, $\texttt{to}_{DR}$ such that

$$\texttt{GOTO}(\texttt{from}_{AH}, \epsilon) = \texttt{to}_{AH} \wedge \texttt{to}_{DR} \in \texttt{to}_{AH},$$

there exists $\texttt{from}_{DR}$ such that

$$\texttt{from}_{DR} \in \texttt{from}_{AH} \wedge \texttt{to}_{DR} \in \texttt{Predict}(\texttt{from}_{DR}).$$

*Observation* 9.5. AHFA prediction is complete with respect to the dotted rules. That is, for all $\texttt{from}_{AH}$, $\texttt{from}_{DR}$, $\texttt{to}_{DR}$, if

$$\texttt{from}_{DR} \in \texttt{from}_{AH} \wedge \texttt{to}_{DR} \in \texttt{Predict}(\texttt{from}_{DR}),$$

then there exists $\texttt{to}_{AH}$ such that

$$\texttt{to}_{DR} \in \texttt{to}_{AH} \wedge \texttt{GOTO}(\texttt{from}_{AH}, \epsilon) = \texttt{to}_{AH}$$

## 10. Marpa is correct

### 10.1. Marpa's Earley sets grow at worst linearly.

**Theorem 10.1.** *For a context-free grammar, and a parse location* $i_{LOC}$*,*

$$\left|\texttt{table}[\text{Marpa}, \texttt{i}]\right| = \mathcal{O}(\texttt{i}).$$

*Proof.* EIM's have the form $[\texttt{x}_{AH}, \texttt{x}_{ORIG}]$. $\texttt{x}_{ORIG}$ is the origin of the EIM, which in Marpa cannot be after the current Earley set at $\texttt{i}_{LOC}$, so that

$$0 \le \texttt{x}_{ORIG} \le \texttt{i}_{LOC}.$$

The possibilities for $\texttt{x}_{AH}$ are finite, since the number of AHFA states is a constant, $|\texttt{fa}|$, which depends on $\texttt{g}$. Since duplicate EIM's are never added to an Earley set, the maximum size of Earley set $\texttt{i}_{LOC}$ is therefore

$$\texttt{i}_{LOC} \times |\texttt{fa}| = \mathcal{O}(\texttt{i}_{LOC}). \qquad \square$$

## 10.2. Marpa's Earley sets are correct.

**Theorem 10.2.** *Marpa's Earley sets are correct.*

The proof is by triple induction, that is, induction with a depth down to 3 levels. We number the levels of induction 0, 1 and 2, starting with the outermost. The level 0 induction is usually called the outer induction. The level 1 induction is usually called the inner induction. Level 2 induction is referred to by number.

The outer induction is on the Earley sets. The outer induction hypothesis is that all Earley sets $\texttt{table}[\textsc{Marpa}, \texttt{i}]$, $0 \le \texttt{i}_{LOC} \le \texttt{n}_{LOC}$, are complete and consistent, and therefore correct. We leave it as an exercise to show, as the basis of the induction, that $\texttt{table}[\textsc{Marpa}, 0]$ is complete and consistent.

To show the outer induction step, we show first consistency, then completeness. We show consistency by an inner induction on the Marpa operations. The inner induction hypothesis is that $\texttt{table}[\textsc{Marpa}, \texttt{i}]$, as so far built, is consistent with respect to $\texttt{table}[\textsc{Leo}, \texttt{i}]$.

As the basis of the inner induction, an empty Marpa Earley set is consistent, trivially. We show the step of the inner induction by cases:

- Marpa scanning operations;
- Marpa reductions when there are no Leo reductions; and
- Marpa's Leo reductions

10.2.1. *Marpa scanning is consistent.* For Marpa's scanning operation, we know that the predecessor EIM is correct by the outer induction hypothesis, and that the token is correct by the definitions in the preliminaries. We know, from Section 8.5, that at most two EIM's will be added. We now examine them in detail.

Let

$$\texttt{confirmed}_{AH} = \texttt{GOTO}(\texttt{predecessor}_{AH}, \texttt{token}_{SYM})$$

If $\texttt{confirmed}_{AH} = \Lambda$, the pseudocode of Section 8.5 shows that we do nothing. If we do nothing, since $\texttt{table}[\textsc{Marpa}, \texttt{i}]$ is consistent by the inner induction hypothesis, it remains consistent, trivially.

Otherwise, let $\texttt{confirmed}_{EIM} = [\texttt{confirmed}_{AH}, \texttt{i}_{LOC}]$. We see that $\texttt{confirmed}_{EIM}$ is consistent with respect to $\texttt{table}[\textsc{Leo}, \texttt{i}]$, by the definition of Earley scanning (Section 5.2) and Observation 9.2. Consistency is invariant under union, and since $\texttt{table}[\textsc{Marpa}, \texttt{i}]$ is consistent by the inner induction, $\texttt{table}[\textsc{Marpa}, \texttt{i}]$ remains consistent after $\texttt{confirmed}_{EIM}$ is added.

For predictions, if $\texttt{confirmed}_{AH} \ne \Lambda$, let

$$\texttt{predicted}_{AH} = \texttt{GOTO}(\texttt{confirmed}_{AH}, \epsilon)$$

If $\texttt{predicted}_{AH} = \Lambda$, the pseudocode of Section 8.11 shows that we do nothing. If we do nothing, since $\texttt{table}[\textsc{Marpa}, \texttt{i}]$ is consistent by the inner induction hypothesis, it remains consistent, trivially. Otherwise, let

$$\texttt{predicted}_{EIM} = [\texttt{predicted}_{AH}, \texttt{i}_{LOC}].$$

We see that $\texttt{predicted}_{EIM}$ is consistent with respect to $\texttt{table}[\textsc{Leo}, \texttt{i}]$, by the definition of Earley prediction (Section 5.4) and Observation 9.4. Consistency is invariant under union and, since $\texttt{table}[\textsc{Marpa}, \texttt{i}]$ is consistent by the inner induction, $\texttt{table}[\textsc{Marpa}, \texttt{i}]$ remains consistent after $\texttt{predicted}_{EIM}$ is added.

10.2.2. *Earley reduction is consistent.* Next, we show that Marpa's reduction operation is consistent, in the case where there is no Leo reduction. There will be two cause EIM's, $\texttt{predecessor}_{EIM}$ and $\texttt{component}_{EIM}$. $\texttt{predecessor}_{EIM}$ will be correct by the outer induction hypothesis and $\texttt{component}_{EIM}$ will be consistent by the inner induction hypothesis. From $\texttt{component}_{EIM}$, we will find zero or more transition symbols, $\texttt{lhs}_{SYM}$. From this point, the argument is very similar to that for the case of the scanning operation.

Let

$$\texttt{confirmed}_{AH} = \texttt{GOTO}(\texttt{predecessor}_{AH}, \texttt{lhs}_{SYM})$$

If $\texttt{confirmed}_{AH} = \Lambda$, we do nothing, and $\texttt{table}[\textsc{Marpa}, \texttt{i}]$ remains consistent, trivially. Otherwise, let

$$\texttt{confirmed}_{EIM} = [\texttt{confirmed}_{AH}, \texttt{i}_{LOC}].$$

We see that $\texttt{confirmed}_{EIM}$ is consistent with respect to $\texttt{table}[\textsc{Leo}, \texttt{i}]$ by the definition of Earley reduction (Section 5.3), and Observation 9.2. By the invariance of consistency under union, $\texttt{table}[\textsc{Marpa}, \texttt{i}]$ remains consistent after $\texttt{confirmed}_{EIM}$ is added.

For predictions, the argument exactly repeats that of Section 10.2.1. $\texttt{table}[\textsc{Marpa}, \texttt{i}]$ remains consistent, whether or not a $\texttt{predicted}_{EIM}$ is added.

10.2.3. *Leo reduction is consistent.* We now show consistency for Marpa's reduction operation, in the case where there is a Leo reduction. If there is a Leo reduction, it is signaled by the presence of $\texttt{predecessor}_{LIM}$,

$$\texttt{predecessor}_{LIM} = [\texttt{top}_{AH}, \texttt{lhs}_{SYM}, \texttt{top}_{ORIG}]$$

in the Earley set where we would look for the $\texttt{predecessor}_{EIM}$. We treat the logic to create $\texttt{predecessor}_{LIM}$ as a matter of memoization of the previous Earley sets, and its correctness follows from the outer induction hypothesis.

As the result of a Leo reduction, LEO will add $[\mathtt{top}_{DR}, \mathtt{top}_{ORIG}]$ to $\mathtt{table}[\text{LEO}, \mathtt{j}]$. Because the MARPA LIM is correct, using Observations 9.2 and 9.3 and Theorem 7.1, we see that $\mathtt{top}_{AH}$ is the singleton set $\{\mathtt{top}_{DR}\}$. From Section 8.10, we see that, as the result of the Leo reduction, MARPA will add

$$\mathtt{leo}_{EIM} = [\mathtt{top}_{AH}, \mathtt{top}_{ORIG}]$$

to $\mathtt{table}[\text{MARPA}, \mathtt{j}]$. The consistency of $\mathtt{leo}_{EIM}$ follows from the definition of EIM consistency. The consistency of $\mathtt{table}[\text{MARPA}, \mathtt{i}]$, once $\mathtt{leo}_{EIM}$ is added, follows by the invariance of consistency under union.

10.2.4. *Marpa's Earley sets are consistent.* Sections 10.2.1, 10.2.2 and 10.2.3 show the cases for the step of the inner induction, which shows the induction. It was the purpose of the inner induction to show that consistency of $\mathtt{table}[\text{MARPA}, \mathtt{i}]$ is invariant under Marpa's operations.

10.2.5. *The inner induction for completeness.* It remains to show that, when Marpa's operations are run as described in the pseudocode of Section 8, that $\mathtt{table}[\text{MARPA}, \mathtt{i}]$ is complete. To do this, we show that at least one EIM in $\mathtt{table}[\text{MARPA}, \mathtt{i}]$ corresponds to every EIMT in $\mathtt{table}[\text{LEO}, \mathtt{i}]$. We will proceed by cases, where the cases are LEO operations. For every operation that LEO would perform, we show that MARPA performs an operation that produces a corresponding Earley item. Our cases for the operations of LEO are Earley scanning operations; Earley reductions; Leo reductions; and Earley predictions.

10.2.6. *Scanning is complete.* For scanning, the Marpa pseudocode shows that a scan is attempted for every pair

$$[\mathtt{predecessor}_{EIM}, \mathtt{token}_{SYM}],$$

where $\mathtt{predecessor}_{EIM}$ is an EIM in the previous Earley set, and $\mathtt{token}_{SYM}$ is the token scanned at $\mathtt{i}_{LOC}$. (The pseudocode actually finds $\mathtt{predecessor}_{EIM}$ in a set returned by $\mathtt{transitions}()$. This is a memoization for efficiency and we will ignore it.)

By the preliminary definitions, we know that $\mathtt{token}_{SYM}$ is the same in both EARLEY and LEO. By the outer induction hypothesis we know that, for every traditional Earley item in the previous Earley set, there is at least one corresponding Marpa Earley item. Therefore, MARPA performs its scan operation on a complete set of correct operands.

Comparing the Marpa pseudocode (section 8.5), with the Earley scanning operation (section 5.2) and using Observations 9.3 and 9.5, we see that a Earley item will be added to $\mathtt{table}[\text{MARPA}, \mathtt{i}]$ corresponding to every scanned Earley item of $\mathtt{table}[\text{LEO}, \mathtt{i}]$. We also see, from the pseudocode of Section 8.11, that the MARPA scanning operation will add to $\texttt{table}[\textsc{Marpa}, \texttt{i}]$ an Earley item for every prediction that results from a scanned Earley item in $\texttt{table}[\textsc{Leo}, \texttt{i}]$.

10.2.7. *Earley reduction is complete.* We now examine Earley reduction, under the assumption that there is no Leo transition. The Marpa pseudocode shows that the Earley items in $\texttt{table}[\textsc{Marpa}, \texttt{i}]$ are traversed in a single pass for reduction.

To show that we traverse a complete and consistent series of component Earley items, we stipulate that the Earley set is an ordered set, and that new Earley items are added at the end. From Theorem 10.1, we know that the number of Earley items is finite, so a traversal of them must terminate.

Consider, for the purposes of the level 2 induction, the reductions of LEO to occur in generations. Let the scanned Earley items be generation 0. An EIMT produced by a reduction is generation $\texttt{n} + 1$ if its component Earley item was generation was $\texttt{n}$. Predicted Earley items do not need to be assigned generations. In Marpa grammars they can never contain completions, and therefore can never act as the component of a reduction.

The induction hypothesis for the level 2 induction is that for some $\texttt{n}$, the Earley items of $\texttt{table}[\textsc{Marpa}, \texttt{i}]$ for generations 0 through $\texttt{n}$ are complete and consistent. From Section 10.2.4, we know that all Earley items in Marpa's sets are consistent. In Section 10.2.6, we showed that generation 0 is complete – it contains Earley items corresponding to all of the generation 0 EIMT's of LEO. This is the basis of the level 2 induction.

Since we stipulated that MARPA adds Earley items at the end of each set, we know that they occur in generation order. Therefore MARPA, when creating Earley items of generation $\texttt{n} + 1$ while traversing $\texttt{table}[\textsc{Marpa}, \texttt{i}]$, can rely on the level 2 induction hypothesis for the completeness of Earley items in generation $\texttt{n}$.

Let $\texttt{working}_{EIM} \in \texttt{i}_{ES}$ be the Earley item currently being considered as a potential component for an Earley reduction operation. From the pseudocode, we see that reductions are attempted for every pair $\texttt{predecessor}_{EIM}$, $\texttt{working}_{EIM}$. (Again, $\texttt{transitions}()$ is ignored as a memoization.) By the outer induction hypothesis we know that, for every traditional Earley item in the previous Earley set, there is at least one corresponding Marpa Earley item. We see from the pseudocode, therefore, that for each $\texttt{working}_{EIM}$ that MARPA performs its reduction operation on a complete set of correct predecessors. Therefore MARPA performs its reduction operations on a complete set of operand pairs.

Comparing the Marpa pseudocode (Section 8.9) with the Earley reduction operation (Section 5.3) and using Observations 9.3 and 9.5, we see that a Earley reduction result of generation $\texttt{n}+1$ will be added to $\texttt{table}[\textsc{Marpa}, \texttt{i}]$ corresponding to every Earley reduction result in generation $\texttt{n}+1$ of $\texttt{table}[\textsc{Leo}, \texttt{i}]$, as well as one corresponding to every prediction that results from an Earley reduction result of generation $\texttt{n}+1$ in $\texttt{table}[\textsc{Leo}, \texttt{i}]$. This shows the level 2 induction and the case of reduction completeness.

10.2.8. *Leo reduction is complete.* We now show completeness for Marpa's reduction operation, in the case where there is a Leo reduction. In Section 10.2.3, we found that where Leo would create the EIMT $[\texttt{top}_{DR}, \texttt{top}_{ORIG}]$, Marpa adds $[\texttt{top}_{AH}, \texttt{top}_{ORIG}]$ such that $\texttt{top}_{DR} \in \texttt{top}_{AH}$. Since $\texttt{top}_{DR}$ is a completed rule, there are no predictions. This shows the case immediately, by the definition of completeness.

10.2.9. *Prediction is complete.* Predictions result only from items in the same Earley set. In Sections 10.2.6, 10.2.7 and 10.2.8, we showed that, for every prediction that would result from an item added to $\texttt{table}[\textsc{Leo}, \texttt{i}]$, a corresponding prediction was added to $\texttt{table}[\textsc{Marpa}, \texttt{i}]$.

10.2.10. *Finishing the proof.* Having shown the cases in Sections 10.2.6, 10.2.7, 10.2.8 and 10.2.9, we know that Earley set $\texttt{table}[\textsc{Marpa}, \texttt{i}]$ is complete. In section 10.2.4 we showed that $\texttt{table}[\textsc{Marpa}, \texttt{i}]$ is consistent. It follows that $\texttt{table}[\textsc{Marpa}, \texttt{i}]$ is correct, which is the step of the outer induction. Having shown its step, we have the outer induction, and the theorem. □

10.3. **Marpa is correct.** We are now is a position to show that Marpa is correct.

**Theorem 10.3.** $\texttt{L}(\textsc{Marpa}, \texttt{g}) = \texttt{L}(\texttt{g})$

*Proof.* From Theorem 10.2, we know that

$$[\texttt{accept}_{DR}, 0] \in \texttt{table}[\textsc{Leo}, |\texttt{w}|]$$

if and only there is a

$$[\texttt{accept}_{AH}, 0] \in \texttt{table}[\textsc{Marpa}, |\texttt{w}|]$$

such that $\texttt{accept}_{DR} \in \texttt{accept}_{AH}$. From the acceptance criteria in the Leo definitions and the Marpa pseudocode, it follows that

$$\texttt{L}(\textsc{Marpa}, \texttt{g}) = \texttt{L}(\textsc{Leo}, \texttt{g}).$$

By Theorem 4.1 in [10], we know that

$$\mathtt{L}(\textsc{Leo}, \mathtt{g}) = \mathtt{L}(\mathtt{g}).$$

The theorem follows from the previous two equalities. □

## 11. Marpa recognizer complexity

### 11.1. **Complexity of each Earley item.**

For the complexity proofs, we consider only Marpa grammars without nulling symbols. We showed that this rewrite is without loss of generality in Section 9.1, when we examined correctness. For complexity we must also show that the rewrite and its reversal can be done in amortized $\mathcal{O}(1)$ time and space per Earley item.

**Lemma 11.1.** *All time and space required to rewrite the grammar to eliminate nulling symbols, and to restore those rules afterwards in the Earley sets, can be allocated to the Earley items in such a way that each Earley item requires $\mathcal{O}(1)$ time and space.*

*Proof.* The time and space used in the rewrite is a constant that depends on the grammar, and is charged to the parse. The reversal of the rewrite can be done in a loop over the Earley items, which will have time and space costs per Earley item, plus a fixed overhead. The fixed overhead is $\mathcal{O}(1)$ and is charged to the parse. The time and space per Earley item is $\mathcal{O}(1)$ because the number of rules into which another rule must be rewritten, and therefore the number of Earley items into which another Earley item must be rewritten, is a constant that depends on the grammar. □

**Theorem 11.2.** *All time in* Marpa *can be allocated to the Earley items, in such a way that each Earley item, and each attempt to add a duplicate Earley item, requires $\mathcal{O}(1)$ time.*

**Theorem 11.3.** *All space in* Marpa *can be allocated to the Earley items, in such a way that each Earley item requires $\mathcal{O}(1)$ space and, if links are not considered, each attempt to add a duplicate Earley item adds no additional space.*

**Theorem 11.4.** *If links are considered, all space in* Marpa *can be allocated to the Earley items in such a way that each Earley item and each attempt to add a duplicate Earley item requires $\mathcal{O}(1)$ space.*

*Proof of Theorems 11.2, 11.3, and 11.4.* These theorems follows from the observations in Section 8 and from Lemma 11.1. □

11.2. **Duplicate dotted rules.** The same complexity results apply to Marpa as to Leo, and the proofs are very similar. Leo's complexity results[10] are based on charging resource to Earley items, as were the results in Earley's paper[3]. But both assume that there is one dotted rule per Earley item, which is not the case with Marpa.

Marpa's Earley items group dotted rules into AHFA states, but this is not a partitioning in the strict sense – dotted rules can fall into more than one AHFA state. This is an optimization, in that it allows dotted rules, if they often occur together, to be grouped together aggressively. But it opens up the possibility that, in cases where Earley and Leo disposed of a dotted rule once and for all, Marpa might have to deal with it multiple times.

From an efficiency perspective, Marpa's duplicate rules are by all the evidence, a plus. And they do not change the complexity results, although the price of showing this is the theoretical apparatus of this section.

**Theorem 11.5.**

$$\left|\texttt{table}[\textsc{Marpa}]\right| < \texttt{c} \times \left|\texttt{table}[\textsc{Leo}]\right|,$$

*where* $\texttt{c}$ *is a constant that depends on the grammar.*

*Proof.* We know from Theorem 10.2 that every Marpa Earley item corresponds to one of Leo's traditional Earley items. If an EIM corresponds to an EIMT, the AHFA state of the EIM contains the EIMT's dotted rule, while their origins are identical. Even in the worst case, a dotted rule cannot appear in every AHFA state, so that the number of Marpa items corresponding to a single traditional Earley item must be less than $|\texttt{fa}|$. Therefore,

$$\left|\texttt{table}[\textsc{Marpa}]\right| < |\texttt{fa}| \times \left|\texttt{table}[\textsc{Leo}]\right| \qquad \square$$

Earley[3] shows that, for unambiguous grammars, every attempt to add an Earley item will actually add one. In other words, there will be no attempts to add duplicate Earley items. Earley's proof shows that for each attempt to add a duplicate, the causation must be different – that the EIMT's causing the attempt differ in either their dotted rules or their origin. Multiple causations for an Earley item would mean multiple derivations for the sentential form that it represents. That in turn would mean that the grammar is ambiguous, contrary to assumption.

In Marpa, there is an slight complication. A dotted rule can occur in more than one AHFA state. Because of that, it is possible that two of Marpa's operations to add an EIM will represent identical Earley causations, and therefore will be consistent with an unambiguous grammar. Dealing with this complication requires us to prove a result that is weaker than that of [3], but that is still sufficient to produce the same complexity results.

**Theorem 11.6.** *For an unambiguous grammar, the number of attempts to add Earley items will be less than or equal to*

$$\texttt{c} \times \left|\texttt{table}[\text{Marpa}]\right|,$$

*where* $\texttt{c}$ *is a constant that depends on the grammar.*

*Proof.* Let `initial-tries` be the number of attempts to add the initial item to the Earley sets. For Earley set 0, it is clear from the pseudocode that there will be no attempts to add duplicate EIM's:

$$\texttt{initial-tries} = \left|\texttt{table}[0]\right|$$

Let `leo-tries` be the number of attempted Leo reductions in Earley set $\texttt{j}_{LOC}$. For Leo reduction, we note that by its definition, duplicate attempts at Leo reduction cannot occur. Let `max-AHFA` be the maximum number of dotted rules in any AHFA state. From the pseudo-code of Sections 8.8 and 8.10, we know there will be at most one Leo reduction for each each dotted rule in the current Earley set, $\texttt{j}_{LOC}$.

$$\texttt{leo-tries} \le \texttt{max-AHFA} \times \left|\texttt{table}[\texttt{j}]\right|$$

Let `scan-tries` be the number of attempted scan operations in Earley set $\texttt{j}_{LOC}$. Marpa attempts a scan operation, in the worst case, once for every EIM in the Earley set at $\texttt{j}_{LOC} - 1$. Therefore, the number of attempts to add scans must be less than equal to $\left|\texttt{table}[\texttt{j} - 1]\right|$, the number of actual Earley items at $\texttt{j}_{LOC} - 1$.

$$\texttt{scan-tries} \le \left|\texttt{table}[\texttt{j} - 1]\right|$$

Let `predict-tries` be the number of attempted predictions in Earley set $\texttt{j}_{LOC}$. Marpa includes prediction in its scan and reduction operations, and the number of attempts to add duplicate predicted EIM's must be less than or equal to the number of attempts to add duplicate confirmed EIM's in the scan and reduction operations.

$$\texttt{predict-tries} \le \texttt{reduction-tries} + \texttt{scan-tries}$$

The final and most complicated case is Earley reduction. Recall that $\texttt{j}_{ES}$ is the current Earley set. Consider the number of reductions attempted. Marpa attempts to add an Earley reduction result once for every triple

$$[\texttt{predecessor}_{EIM}, \texttt{transition}_{SYM}, \texttt{component}_{EIM}].$$

where

$$\begin{aligned} & \texttt{component}_{EIM} = [\texttt{component}_{AH}, \texttt{component-origin}_{LOC}] \\ \wedge \quad & \texttt{component}_{DR} \in \texttt{component}_{AH} \\ \wedge \quad & \texttt{transition}_{SYM} = \texttt{LHS}(\texttt{component}_{DR}). \end{aligned}$$

We now put an upper bound on number of possible values of this triple. The number of possibilities for $\texttt{transition}_{SYM}$ is clearly at most $|\texttt{symbols}|$, the number of symbols in $\texttt{g}$. We have $\texttt{component}_{EIM} \in \texttt{j}_{ES}$, and therefore there are at most $\left|\texttt{j}_{ES}\right|$ choices for $\texttt{component}_{EIM}$.

We can show that the number of possible choices of $\texttt{predecessor}_{EIM}$ is at most the number of AHFA states, $|\texttt{fa}|$, by a reductio. Suppose, for the reduction, there were more than $|\texttt{fa}|$ possible choices of $\texttt{predecessor}_{EIM}$. Then there are two possible choices of $\texttt{predecessor}_{EIM}$ with the same AHFA state. Call these $\texttt{choice1}_{EIM}$ and $\texttt{choice2}_{EIM}$. We know, by the definition of Earley reduction, that $\texttt{predecessor}_{EIM} \in \texttt{j}_{ES}$, and therefore we have $\texttt{choice1}_{EIM} \in \texttt{j}_{ES}$ and $\texttt{choice2}_{EIM} \in \texttt{j}_{ES}$. Since all EIM's in an Earley set must differ, and $\texttt{choice1}_{EIM}$ and $\texttt{choice2}_{EIM}$ both have the same AHFA state, they must differ in their origin. But two different origins would produce two different derivations for the reduction, which would mean that the parse was ambiguous. This is contrary to the assumption for the theorem that the grammar is unambiguous. This shows the reductio and that the number of choices for $\texttt{predecessor}_{EIM}$, compatible with $\texttt{component}_{ORIG}$, is as most $|\texttt{fa}|$.

Collecting the results we see that the possibilities for each $\texttt{component}_{EIM}$ are

$$\begin{array}{rll} & |\texttt{fa}| & \text{choices of } \texttt{predecessor}_{EIM} \\ \times & |\texttt{symbols}| & \text{choices of } \texttt{transition}_{SYM} \\ \times & |\texttt{j}_{ES}| & \text{choices of } \texttt{component}_{EIM} \end{array}$$

The number of reduction attempts will therefore be at most

$$\texttt{reduction-tries} \le |\texttt{fa}| \times |\texttt{symbols}| \times \left|\texttt{j}_{ES}\right|.$$

Summing

$$\begin{aligned} \texttt{tries} = {} & \texttt{scan-tries} + \texttt{leo-tries} + \\ & \texttt{predict-tries} + \texttt{reduction-tries} + \texttt{initial-tries}, \end{aligned}$$

we have, where $\texttt{n} = |\texttt{w}|$, the size of the input,

$$\begin{aligned}
&\left|\texttt{table}[0]\right| && \text{initial EIM's}\\
&+ \sum_{i=0}^{n} \texttt{max-AHFA} \times \left|\texttt{table}[\texttt{j}]\right| && \text{LIM's}\\
&+ 2 \times \sum_{i=1}^{n} \left|\texttt{table}[\texttt{j} - 1]\right| && \text{scanned EIM's}\\
&+ 2 \times \sum_{i=0}^{n} |\texttt{fa}| \times |\texttt{symbols}| \times \left|\texttt{j}_{ES}\right| && \text{reduction EIM's.}
\end{aligned}$$

In this summation, `prediction-tries` was accounted for by counting the scanned and predicted EIM attempts twice. Since `max-AHFA` and $|\texttt{symbols}|$ are both constants that depend only on g, if we collect the terms of the summation, we will find a constant c such that

$$\texttt{tries} \le \texttt{c} \times \sum_{i=0}^{n} \left|\texttt{table}[\texttt{j}]\right|,$$

and

$$\texttt{tries} \le \texttt{c} \times \left|\texttt{table}[\textsc{Marpa}]\right|,$$

where c is a constant that depends on g. □

As a reminder, we follow tradition by stating complexity results in terms of n, setting $\texttt{n} = |\texttt{w}|$, the length of the input.

**Theorem 11.7.** *For a context-free grammar,*

$$\left|\texttt{table}[\textsc{Marpa}]\right| = \mathcal{O}(\texttt{n}^2).$$

*Proof.* By Theorem 10.1, the size of the Earley set at $\texttt{i}_{LOC}$ is $\mathcal{O}(\texttt{i})$. Summing over the length of the input, $|\texttt{w}| = \texttt{n}$, the number of EIM's in all of Marpa's Earley sets is

$$\sum_{\texttt{i}_{LOC}=0}^{\texttt{n}} \mathcal{O}(\texttt{i}) = \mathcal{O}(\texttt{n}^2). \qquad \square$$

**Theorem 11.8.** *For a context-free grammar, the number of attempts to add Earley items is $\mathcal{O}(\boldsymbol{n}^3)$.*

*Proof.* Reexamining the proof of Theorem 11.6, we see that the only bound that required the assumption that g was unambiguous was `reduction-tries`, the count of the number of attempts to add Earley reductions. Let `other-tries` be attempts to add EIM's other than as the result of Earley reductions. By Theorem 11.7,

$$\left|\texttt{table}[\textsc{Marpa}]\right| = \mathcal{O}(\texttt{n}^2),$$

and by Theorem 11.6,

$$\texttt{other-tries} \le \texttt{c} \times \left|\texttt{table}[\textsc{Marpa}]\right|,$$

so that $\texttt{other-tries} = \mathcal{O}(\texttt{n}^2)$.

Looking again at `reduction-tries` for the case of ambiguous grammars, we need to look again at the triple

$$[\texttt{predecessor}_{EIM}, \texttt{transition}_{SYM}, \texttt{component}_{EIM}].$$

We did not use the fact that the grammar was unambigous in counting the possibilities for $\texttt{transition}_{SYM}$ or $\texttt{component}_{EIM}$, but we did make use of it in determining the count of possibilities for $\texttt{predecessor}_{EIM}$. We know still know that

$$\texttt{predecessor}_{EIM} \in \texttt{component-origin}_{ES},$$

where $\texttt{component-origin}_{LOC}$ is the origin of $\texttt{component}_{EIM}$. Worst case, every EIM in $\texttt{component-origin}_{ES}$ is a possible match, so that the number of possibilities for $\texttt{predecessor}_{EIM}$ now grows to $|\texttt{component-origin}_{ES}|$, and

$$\texttt{reduction-tries} = \left|\texttt{component-origin}_{ES}\right| \times |\texttt{symbols}| \times \left|\texttt{j}_{ES}\right|.$$

In the worst case $\texttt{component-origin} \simeq \texttt{j}$, so that by Theorem 10.1,

$$|\texttt{component-origin}_{ES}| \times |\texttt{j}_{ES}| = \mathcal{O}(\texttt{j}^2).$$

Adding `other-tries` and summing over the Earley sets, we have

$$\mathcal{O}(\texttt{n}^2) + \sum_{\texttt{j}_{LOC}=0}^{n} \mathcal{O}(\texttt{j}^2) = \mathcal{O}(\texttt{n}^3). \qquad \square$$

**Theorem 11.9.** *Either a right derivation has a step that uses a right recursive rule, or it has length is at most* `c`*, where* `c` *is a constant which depends on the grammar.*

*Proof.* Let the constant `c` be the number of symbols. Assume, for a reductio, that a right derivation expands to a Leo sequence of length $\texttt{c} + 1$, but that none of its steps uses a right recursive rule.

Because it is of length $\texttt{c} + 1$, the same symbol must appear twice as the rightmost symbol of a derivation step. (Since for the purposes of these complexity results we ignore nulling symbols, the rightmost symbol of a string will also be its rightmost non-nulling symbol.) So part of the rightmost derivation must take the form

$$\texttt{earlier-prefix}_{STR} \cdot \texttt{A}_{SYM} \stackrel{+}{\Rightarrow} \texttt{later-prefix}_{STR} \cdot \texttt{A}_{SYM}.$$

But the first step of this derivation sequence must use a rule of the form

$$\mathtt{A}_{SYM} \rightarrow \mathtt{rhs\text{-}prefix}_{STR} \,.\, \mathtt{rightmost}_{SYM},$$

where $\mathtt{rightmost}_{SYM} \overset{+}{\Rightarrow} \mathtt{A}_{SYM}$. Such a rule is right recursive by definition. This is contrary to the assumption for the reductio. We therefore conclude that the length of a right derivation must be less than or equal to $\mathtt{c}$, unless at least one step of that derivation uses a right recursive rule. □

11.3. **The complexity results.** We are now in a position to show specific time and space complexity results.

**Theorem 11.10.** *For every LR-regular grammar,* Marpa *runs in* $\mathcal{O}(\mathtt{n})$ *time and space.*

*Proof.* By Theorem 4.6 in [10, p. 173], the number of traditional Earley items produced by Leo when parsing input $\mathtt{w}$ with an LR-regular grammar $\mathtt{g}$ is

$$\mathcal{O}(|\mathtt{w}|) = \mathcal{O}(\mathtt{n}).$$

Marpa may produce more Earley items than Leo for two reasons: First, Marpa does not apply Leo memoization to Leo sequences which do not contain right recursion. Second, Marpa's Earley items group dotted rules into states and this has the potential to increase the number of Earley items.

By theorem 7.1, the definition of an EIMT, and the construction of a Leo sequence, it can be seen that a Leo sequence corresponds step-for-step with a right derivation. It can therefore be seen that the number of EIMT's in the Leo sequence and the number of right derivation steps in its corresponding right derivation will be the same.

Consider one EIMT that is memoized in Leo. By theorem 7.1 it corresponds to a single dotted rule, and therefore a single rule. If not memoized because it is not a right recursion, this EIMT will be expanded to a sequence of EIMT's. How long will this sequence of non-memoized EIMT's be, if we still continue to memoize EIMT's which correspond to right recursive rules? The EIMT sequence, which was formerly a memoized Leo sequence, will correspond to a right derivation that does not include any steps that use right recursive rules. By Theorem 11.9, such a right derivation can be of length at most $\mathtt{c1}$, where $\mathtt{c1}$ is a constant that depends on $\mathtt{g}$. As noted, this right derivation has the same length as its corresponding EIMT sequence, so that each EIMT not memoized in Marpa will expand to at most $\mathtt{c1}$ EIMT's.

By Theorem 11.5, when EIMT's are replaced with EIM's, the number of EIM's Marpa requires is at worst, $\mathtt{c2}$ times the number of EIMT's, where $\texttt{c2}$ is a constant that depends on $\texttt{g}$. Therefore the number of EIM's per Earley set for an LR-regular grammar in a MARPA parse is less than

$$\texttt{c1} \times \texttt{c2} \times \mathcal{O}(\texttt{n}) = \mathcal{O}(\texttt{n}).$$

LR-regular grammar are unambiguous, so that by Theorem 11.6, the number of attempts that MARPA will make to add EIM's is less than or equal to $\texttt{c3}$ times the number of EIM's, where $\texttt{c3}$ is a constant that depends on $\texttt{g}$. Therefore, by Theorems 11.2 and 11.4, the time and space complexity of MARPA for LR-regular grammars is

$$\texttt{c3} \times \mathcal{O}(\texttt{n}) = \mathcal{O}(\texttt{n}).$$ □

**Theorem 11.11.** *For every unambiguous grammar,* MARPA *runs in $\mathcal{O}(n^2)$ time and space.*

*Proof.* By assumption, $\texttt{g}$ is unambiguous, so that by Theorem 11.6, and Theorem 11.7, the number of attempts that MARPA will make to add EIM's is

$$\texttt{c} \times \mathcal{O}(\texttt{n}^2),$$

where $\texttt{c}$ is a constant that depends on $\texttt{g}$. Therefore, by Theorems 11.2 and 11.4, the time and space complexity of MARPA for unambiguous grammars is $\mathcal{O}(\texttt{n}^2)$. □

**Theorem 11.12.** *For every context-free grammar,* MARPA *runs in $\mathcal{O}(\boldsymbol{n}^3)$ time.*

*Proof.* By Theorem 11.2, and Theorem 11.8. □

**Theorem 11.13.** *For every context-free grammar,* MARPA *runs in $\mathcal{O}(\boldsymbol{n}^2)$ space, if it does not tracks links.*

*Proof.* By Theorem 11.3 and Theorem 11.7. □

Traditionally only the space result stated for a parsing algorithm is that without links, as in 11.13. This is sufficiently relevant if the parser is only used as a recognizer. In practice, however, algorithms like MARPA are typically used in anticipation of an evaluation phase, for which links are necessary.

**Theorem 11.14.** *For every context-free grammar,* MARPA *runs in $\mathcal{O}(\boldsymbol{n}^3)$ space, including the space for tracking links.*

*Proof.* By Theorem 11.4, and Theorem 11.8. □

## 12. The Marpa Input Model

In this paper, up to this point, the traditional input stream model has been assumed. As implemented, Marpa generalizes the idea of input streams beyond the traditional model.

Marpa's generalized input model replaces the input $\texttt{w}$ with a set of tokens, $\texttt{tokens}$, whose elements are triples of symbol, start location and length:

$$[\texttt{t}_{SYM}, \texttt{start}_{LOC}, \texttt{length}]$$

such that $\texttt{length} \geq 1$ and $\texttt{start}_{LOC} \geq 0$. The size of the input, $|\texttt{w}|$, is the maximum over $\texttt{tokens}$ of $\texttt{start}_{LOC} + \texttt{length}$.

Multiple tokens can start at a single location. (This is how Marpa supports ambiguous tokens.) The variable-length, ambiguous and overlapping tokens of Marpa bend the conceptual framework of "parse location" beyond its breaking point, and a new term for parse location is needed. Start and end of tokens are described in terms of **earleme** locations, or simply **earlemes**. Token length is also measured in earlemes.

Like standard parse locations, earlemes start at 0, and run up to $|\texttt{w}|$. Unlike standard parse locations, there is not necessarily a token "at" any particular earleme. (A token is considered to be "at an earleme" if it ends there, so that there is never a token "at" earleme 0.) In fact, there may be earlemes at which no token either starts or ends, although for the parse to succeed, such an earleme would have to be properly inside at least one token. Here "properly inside" means after the token's start earleme and before the token's end earleme.

In the Marpa input stream, tokens may interweave and overlap freely, but gaps are not allowed. That is, for all $\texttt{i}_{LOC}$ such that $0 \leq \texttt{i}_{LOC} < |\texttt{w}|$, there must exist

$$\texttt{token} = [\texttt{t}_{SYM}, \texttt{start}_{LOC}, \texttt{length}]$$

such that

$$\texttt{token} \in \texttt{tokens} \quad \text{and}$$
$$\texttt{start}_{LOC} \leq \texttt{i}_{LOC} < \texttt{start}_{LOC} + \texttt{length}.$$

The intent of Marpa's generalized input model is to allow users to define alternative input models for special applications. An example that arises in current practice is natural language, features of which are most naturally expressed with ambiguous tokens. The traditional input stream can be seen as the special case of the Marpa input model where for all $\texttt{x}_{SYM}$, $\texttt{y}_{SYM}$, $\texttt{x}_{LOC}$, $\texttt{y}_{LOC}$, $\texttt{xlength}$, $\texttt{ylength}$, if we have both of

$$[\mathtt{x}_{SYM}, \mathtt{x}_{LOC}, \mathtt{xlength}] \in \mathtt{tokens} \quad \text{and}$$
$$[\mathtt{y}_{SYM}, \mathtt{y}_{LOC}, \mathtt{ylength}] \in \mathtt{tokens},$$

then we have both of

$$\mathtt{xlength} = \mathtt{ylength} = 1 \quad \text{and}$$
$$\mathtt{x}_{LOC} = \mathtt{y}_{LOC} \implies \mathtt{x}_{SYM} = \mathtt{y}_{SYM}.$$

The correctness results hold for Marpa input streams, but to preserve the time complexity bounds, restrictions must be imposed. In stating them, let it be understood that

$$[\mathtt{x}_{SYM}, \mathtt{x}_{LOC}, \mathtt{length}]_{TOK} \in \mathtt{tokens}$$

We require that, for some constant $\mathtt{c}$, possibly dependent on the grammar $\mathtt{g}$, that every token length be less than $\mathtt{c}$,

$$\forall\, [\mathtt{x}_{SYM}, \mathtt{x}_{LOC}, \mathtt{length}]_{TOK},\ \mathtt{length} < \mathtt{c}, \tag{1}$$

and that the cardinality of the set of tokens starting at any one location be less than $\mathtt{c}$,

$$\forall \mathtt{i}_{LOC},\ \left| \left\{ [\mathtt{x}_{SYM}, \mathtt{x}_{LOC}, \mathtt{length}]_{TOK} \;\middle|\; \mathtt{x}_{LOC} = \mathtt{i}_{LOC} \right\} \right| < \mathtt{c} \tag{2}$$

Restrictions 1 and 2 impose little or no obstacle to the practical use of Marpa's generalized input model. And with them, the complexity results for Marpa stand.

## References

[1] Alfred H. Aho and Jeffrey D. Ullman. The Theory of Parsing, Translation, and Computing Prentice-Hall, Englewood Cliff, N.J., 1972.

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

[3] J. Earley. An efficient context-free parsing algorithm. *Communications of the Association for Computing Machinery*, 13(2):94–102, 1970.

[4] Dirk Grune and Ceriel J.H Jacobs. *Parsing Techniques: A Practical Guide.* Springer, Amsterdam, 2008.

[5] Edgar T. Irons. A syntax-directed compiler for ALGOL 60. *Communications of the Association for Computing Machinery*, 4(1):51-55, Jan. 1961

[6] Stephen C. Johnson. Yacc: Yet another compiler-compiler. In *Unix Programmer's Manual Supplementary Documents 1.* 1986.

[7] Jeffrey Kegler, 2011: Marpa-HTML. `http://search.cpan.org/dist/Marpa-HTML/`.

[8] Jeffrey Kegler, 2013: Marpa-R2. `http://search.cpan.org/dist/Marpa-R2/`.

[9] Jeffrey Kegler, 2011: Marpa-XS-1.002000. `http://search.cpan.org/dist/Marpa-XS/`.

[10] J. M. I. M. Leo. A general context-free parsing algorithm running in linear time on every LR($k$) grammar without using lookahead. *Theoretical Computer Science*, 82:165–176, 1991.

## Contents