On LR Parsing with Selective Delays

Eberhard Bertsch, Mark-Jan Nederhof, Sylvain Schmitz

On LR Parsing with Selective Delays

Eberhard Bertsch[1] Mark-Jan Nederhof[2]
Sylvain Schmitz[3]

[1] Ruhr University, Faculty of Mathematics, Bochum, Germany
[2] School of Computer Science, University of St Andrews, UK
[3] LSV, ENS Cachan & CNRS, Cachan, France

Abstract

The paper investigates an extension of LR parsing that allows the delay of parsing decisions until a sufficient amount of context has been processed. We provide two characterizations for the resulting class of grammars, one based on grammar transformations, the other on the direct construction of a parser. We also report on experiments with a grammar collection.

1 Introduction

From a grammar engineer's standpoint, LR parsing techniques, like the LALR(1) parsers generated by yacc or GNU/bison, suffer from the troublesome existence of *conflicts*, which appear sooner or later in any grammar development. Tracing the source of such conflicts and refactoring the grammar to solve them is a difficult task, for which we refer the reader to the accounts of Malloy et al. (2002) on the development of a C# grammar, and of Gosling et al. (1996) on that of the official Java grammar.

In the literature, different ways have been considered to solve conflicts automatically while maintaining a *deterministic* parsing algorithm—which, besides efficiency considerations, also has the considerable virtue of ruling out ambiguities—, such as unbounded regular lookaheads (Čulik and Cohen, 1973), noncanonical parsers (Szymanski and Williams, 1976), and delays before reductions (Leermakers, 1992). Bertsch and Nederhof (2007) have made a rather counter-intuitive observation on the latter technique: increasing delays uniformly throughout the grammar can in some cases introduce new conflicts.

In this paper we propose a parsing technique that *selects* how long a reduction must be delayed depending on the context. More interestingly, and unlike many techniques that extend LR parsing, we provide a *characterization*, using grammar transformations, of the class of grammars that can be parsed in a LR fashion with selective delays. More precisely,

- we motivate in Section 2 the interest of ML(k, m) parsing on an exerpt of the C++ grammar, before stating the first main contribution of the paper: we reformulate the technique of Bertsch and Nederhof (2007) as a grammar transformation, and show how selective delays can capture non-ML(k, m) grammars,

- we define the class selML(k, m) accordingly through a nondeterministic grammar transformation, which allows us to investigate its properties (Section 3),
- in Section 4 we propose an algorithm to generate parsers with selective delays, and prove that it defines the same class of grammars.
- We implemented a Java proof of concept for this algorithm (see `http://www.cs.st-andrews.ac.uk/~mjn/code/mlparsing/`), and report in Section 5 on the empirical value of selective delays, by applying the parser on a test suite of small unambiguous grammars (Basten, 2008; Schmitz, 2010).
- We conclude with a discussion of related work, in Section 6.

Due to space constraints, many technical details, including several proofs, had to be added in the form of appendices.

Preliminaries. We assume the reader to be familiar with LR parsing, but nonetheless recall some definitions and standard notation.

A *context-free grammar* (CFG) is a tuple $\mathcal{G} = \langle N, \Sigma, P, S\rangle$ where N is a finite set of *nonterminal* symbols, Σ a finite set of *terminal* symbols with $N \cap \Sigma = \emptyset$—together they define the *vocabulary* $V = N \uplus \Sigma$—, $P \subseteq N \times V^*$ is a finite set of *productions* written as rewrite rules "$A \to \alpha$", and $S \in N$ the *start* symbol. The associated *derivation* relation $\Rightarrow$ over V^* is defined as $\Rightarrow = \{(\delta A\gamma, \delta\alpha\gamma) \mid A \to \alpha \in P\}$; a derivation is *rightmost*, denoted $\Rightarrow_{\mathrm{rm}}$, if γ is restricted to be in Σ^* in the above definition. The *language* of a CFG is $L(\mathcal{G}) = \{w \in \Sigma^* \mid S \Rightarrow^* w\} = \{w \in \Sigma^* \mid S \Rightarrow^*_{\mathrm{rm}} w\}$.

We employ the usual conventions for symbols: nonterminals in N are denoted by the first few upper-case Latin letters A, B, ..., terminals in Σ by the first few lower-case Latin letters a, b, ..., symbols in V by the last few upper-case Latin letters X, Y, Z, sequences of terminals in Σ^* by the last few lower-case Latin letters u, v, w, ..., and mixed sequences in V^* by Greek letters α, β, etc. The empty string is denoted by ε.

Given $\mathcal{G} = \langle N, \Sigma, P, S\rangle$, its *$k$-extension* is the grammar $\langle N \uplus \{S^\dagger\}, \Sigma \uplus \{\#\}, P \cup \{S^\dagger \to S\#^k\}, S^\dagger\rangle$ where $\#$ is a fresh symbol. A grammar is *LR(m)* (Knuth, 1965; Sippu and Soisalon-Soininen, 1990) if it is reduced—i.e. every nonterminal is both accessible and productive—and the following *conflict* situation does not arise in its m-extension:

$$S^\dagger \Rightarrow^*_{\mathrm{rm}} \delta Au \Rightarrow_{\mathrm{rm}} \delta\alpha u = \gamma u \qquad \delta \neq \delta' \text{ or } A \neq B \text{ or } \alpha \neq \beta$$
$$S^\dagger \Rightarrow^*_{\mathrm{rm}} \delta' Bv \Rightarrow_{\mathrm{rm}} \delta'\beta v = \gamma w v \qquad m : u = m : wv$$

where "$m : u$" denotes the prefix of length m of u, or the whole of u if $|u| \leq m$.

2 Marcus-Leermakers Parsing

The starting point of this paper is the formalization proposed by Leermakers (1992) of a parsing technique due to Marcus (1980), which tries to imitate the way humans parse natural language sentences. Bertsch and Nederhof (2007)

have given another, equivalent, formulation, and dubbed it "ML" for Marcus-Leermakers.

The idea of uniform ML parsing is that all the reductions are delayed to take place after the recognition of a fixed number k of *right context* symbols, which can contain nonterminal symbols. Bertsch and Nederhof (2007) expanded this class by considering m further symbols of terminal lookahead, thereby defining ML(k, m) grammars. (This construction is recalled in Appendix A.) In Section 2.2, we provide yet another view on uniform ML(k, m) grammars, before motivating the use of selective delays in Section 2.3. Let us start with a concrete example taken from the C++ grammar from the 1998 standard (ISO, 1998).

2.1 C++ Qualified Identifiers

First designed as a preprocessor for C, the C++ language has evolved into a complex standard. Its rather high level of syntactic ambiguity calls for nondeterministic parsing methods, and therefore the published grammar makes no attempt to fit in the LALR(1) class.

We are interested in one particular issue with the syntax of *identifier expressions*, which describe a full name specifier and identifier, possibly instantiating template variables; for instance, "`A::B<C::D>::E`" denotes an identifier "`E`" with name specifier "`A::B<C::D>`", where the template argument of "`B`" is "`D`" with specifier "`C`".

The syntax of identifier expressions is given in the official C++ grammar by the following (simplified) grammar rules:

$$I \to U \mid Q, \quad U \to i \mid T, \quad Q \to N\,U, \quad N \to U :: N \mid U ::, \quad T \to i \texttt{<} I \texttt{>}.$$

An identifier expression I can derive either an *unqualified identifier* through nonterminal U, or a *qualified identifier* through Q, which is qualified through a *nested name specifier* derived from nonterminal N, i.e. through a sequence of unqualified identifiers separated by double colons "`::`", before the identifier i itself. Moreover, each unqualified identifier can be a *template identifier* T, where the *template argument*, between angle brackets "`<`" and "`>`", can again be any identifier expression.

Example 1. A shift/reduce conflict appears with this set of rules. A parser fed with "`A::`", and seeing an identifier "`B`" in its lookahead window, has a nondeterministic choice between

- *reducing* "`A::`" to a single N, in the hope that "`B`" will be the identifier qualified by "`A::`", as in "`A::B<C::D>`", and
- *shifting* the identifier, in the hope that "`B`" will be a specifier of the identifier actually qualified, for instance "`E`" in "`A::B<C::D>::E`".

An informed decision requires an exploration of the specifier starting with "`B`" in search of a double colon symbol. The need for unbounded lookahead occurs if "`B`" is the start of an arbitrarily long template identifier: this grammar is not LR(k) for any finite k.

Note that the double colon token might also appear inside a template argument. Considering that the conflict could also arise there, as after reading "`A<B::`" in "`A<B::C<D::E>::F>::G`", we see that it can be arduous to know

whether a "::" symbol is significant for the resolution of the conflict or not. In fact, this is an example of a conflict that *cannot* be solved by using regular lookahead as proposed in (Boullier, 1984; Bermudez and Schimpf, 1990; Farré and Fortes Gálvez, 2001), because keeping track of the nesting level of well-balanced brackets is beyond the power of regular languages.[1]

2.2 Uniform ML

Observe that, in our extract of the C++ grammar, if we were to *postpone* the choice between the two possible actions and attempt to parse an N in full, then the issue would disappear. The mechanism Leermakers (1992) employs for delaying parsing decisions is to extend a nonterminal with additional terminal and nonterminal symbols from its right context, thus delaying reduction to that nonterminal until the moment when these additional symbols have been parsed in full. This also involves introducing a new end-of-file terminal "#".

In Appendix A, we recall the ML(k, m) parser construction of (Leermakers, 1992; Bertsch and Nederhof, 2007). The automaton obtained by applying this construction on the C++ grammar is too large to be rendered on a single page. Appendix A therefore provides a simpler example. In what follows we present an alternative characterization of ML parsing on the basis of a grammar transformation.

Uniform ML as a Transformation. Although Leermakers does not present his technique in these terms, the intuition of extending nonterminals with right context can be realized by a grammar transformation that introduces nonterminals of the form $[A\delta]$ in $N' = N \cdot V^{\leq k}$, which combine a nonterminal A with its immediate right context δ.

This results for $k = 1$ and our C++ example into an LALR(1) grammar with rules:

$$
\begin{gathered}
[I\#] \to [U\#] \mid [Q\#], \qquad [I\texttt{>}] \to [U\texttt{>}] \mid [Q\texttt{>}], \\
[U\#] \to i\,\# \mid [T\#], \quad [U\texttt{>}] \to i\,\texttt{>} \mid [T\texttt{>}], \quad [U\,\texttt{::}] \to i\,\texttt{::} \mid [T\,\texttt{::}], \quad [U] \to i \mid [T], \\
[Q\#] \to [NU]\,\#, \qquad [Q\texttt{>}] \to [NU]\,\texttt{>}, \\
[NU] \to [U\,\texttt{::}]\,[NU] \mid [U\,\texttt{::}]\,[U], \\
[T\#] \to i\,\texttt{<}\,[I\texttt{>}]\,\#, \quad [T\texttt{>}] \to i\,\texttt{<}\,[I\texttt{>}]\,\texttt{>}, \quad [T\,\texttt{::}] \to i\,\texttt{<}\,[I\texttt{>}]\,\texttt{::}, \quad [T] \to i\,\texttt{<}\,[I\texttt{>}].
\end{gathered}
$$

The new grammar demonstrates that our initial grammar for C++ identifier expressions is ML(1, 1): it requires contexts of length $k = 1$, and lookahead of length $m = 1$.

Combing Function. Formally, the nonterminals in N' are used in the course of the application of the *uniform k-combing function* comb_k from V^* to ($N' \uplus$

[1] We can amend the rules of N to use left-recursion and solve the conflict: $N \to N\,U\,\texttt{::} \mid U\,\texttt{::}$. This correction was made by the Standards Committee in 2003 (see `http://www.open-std.org/jtc1/sc22/wg21/docs/cwg_defects.html#125`). The correction was not motivated by this conflict but by an ambiguity issue, and the fact that the change eliminated the conflict seems to have been a fortunate coincidence. The C++ grammar of the Elsa parser (McPeak and Necula, 2004) employs right recursion.

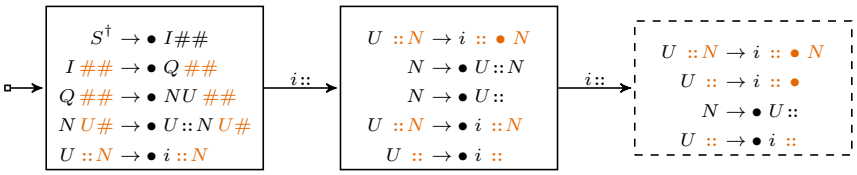

Figure 1: Parts of the uniform ML(2, 0) parser for C++ identifier expressions.

$\Sigma)^*$, defined recursively as:

$$\mathsf{comb}_k(\alpha) = \begin{cases} [A\delta] \cdot \mathsf{comb}_k(\alpha') & \text{if } \alpha = A\delta\alpha', A \in N, \text{ and either } |\delta| = k, \\ & \quad \text{or } |\delta| \leq k \text{ and } \alpha' = \varepsilon \\ a \cdot \mathsf{comb}_k(\alpha') & \text{if } \alpha = a\alpha' \text{ and } a \in \Sigma \\ \varepsilon & \text{otherwise, which is if } \alpha = \varepsilon\ . \end{cases}$$

For instance, $\mathsf{comb}_1(ABcDeF) = [AB]c[De][F]$.

The right parts of the rules of $[A\delta]$ are then of the form $\mathsf{comb}_k(\alpha\delta)$ if $A \to \alpha$ was a rule of the original grammar, effectively delaying the reduction of α to A until after δ has been parsed.

Definition 1 (Uniform combing)**.** Let $\mathcal{G} = \langle N, \Sigma, P, S\rangle$ be a CFG. Its *uniform k-combing* is the CFG $\langle N \cdot V^{\leq k}, \Sigma, \{[A\delta] \to \mathsf{comb}_k(\alpha\delta) \mid \delta \in V^{\leq k} \text{ and } A \to \alpha \in P\}, [S]\rangle$.

Equivalence of the Two Views. Of course we should prove that the two views on ML parsing are equivalent:

Theorem 1. *A grammar is ML(k, m) if and only if the uniform k-combing of its k-extension is LR(m).*

Proof Idea. One can verify that the LR(m) construction on the k-combing of the k-extension of $\mathcal{G}$ and the ML(k, m) construction of Bertsch and Nederhof (2007) (recalled in Appendix A) for the same $\mathcal{G}$ are identical. □

2.3 Selective ML

An issue spotted by Bertsch and Nederhof (2007) is that the classes of ML(k, m) grammars and ML($k+1$, m) grammars are not comparable: adding further delays can introduce new LR(m) conflicts in the ML($k+1$, m)-transformed grammar.

For instance, the uniform 2-combing of our grammar for C++ identifier expressions is not LR(m) for any m: Figure 1 shows the path to a conflict similar to that of the original grammar, which is therefore not uniform ML(2, m). Selective ML aims to find the appropriate delay, i.e. the appropriate amount of right context, for each item in the parser, in order to avoid such situations.

Oscillating Behaviour. Bertsch and Nederhof also show that an oscillating behaviour can occur, for instance with the grammar

$$S \to SdA \mid c,\ A \to a \mid ab \qquad (\mathcal{G}_{\text{odd}})$$

being ML(k, 0) only for odd values of k, and the grammar

$$S \to SAd \mid c,\ A \to a \mid ab \qquad (\mathcal{G}_{\text{even}})$$

being ML(k, 0) only for even values of $k > 0$, from which we can build a union grammar

$$S \to SdA \mid SAd \mid c,\ A \to a \mid ab \qquad (\mathcal{G}_2)$$

which is not ML(k, 0) for any k.

Observe however that, if we use different context lengths for the different rules of S in $\mathcal{G}_2$, i.e. if we *select* the different delays, we can still obtain an LR(0) grammar $\mathcal{G}'_2$ with rules

$$\begin{aligned}
[S^\dagger] &\to [S\#]\#,\\
[S\#] &\to [Sd][A\#] \mid [SAd]\# \mid c\#,\\
[Sd] &\to [Sd][Ad] \mid [SAd]d \mid cd,\\
[SAd] &\to [Sd][AAd] \mid [SAd][Ad] \mid c[Ad],\\
[A\#] &\to a\# \mid ab\#,\\
[Ad] &\to ad \mid abd,\\
[AAd] &\to a[Ad] \mid ab[Ad]
\end{aligned} \qquad (\mathcal{G}'_2)$$

As we will see, this means that $\mathcal{G}_2$ is selective ML with a delay of at most 2, denoted selML(2, 0). This example shows that selective ML(k, m) is not just about finding a minimal global $k' \leq k$ such that the grammar is uniform ML(k', m). Because the amount of delay is optimized depending on the context, selective ML captures a larger class of grammars.

3 Selective Delays Through Grammar Transformation

We define selML(k, m) through a grammar transformation akin to that of Definition 1, but which employs a *combing relation* instead of the uniform k-combing function. We first introduce these relations (Section 3.1) before defining the selML(k, m) grammar class and establishing its relationships with various classes of grammars in Section 3.2 (more comparisons with related work can be found in Section 6).

3.1 Combing Relations

In the following definitions, we let $\mathcal{G} = \langle N, \Sigma, P, S\rangle$ be a context-free grammar. Combing relations are defined through the application of a particular inverse homomorphism throughout the rules of the grammar.

Definition 2 (Selective Combing)**.** Grammar $\mathcal{G}' = \langle N', \Sigma, P', S'\rangle$ is a *selective combing* of $\mathcal{G}$, denoted $\mathcal{G}$ comb $\mathcal{G}'$, if there exists a homomorphism μ from V'^* to V^* such that

1. $\mu(S') = S$,
2. $\forall a \in \Sigma, \mu(a) = a$,

3. $\mu(N') \subseteq N \cdot V^*$, and
4. $\{A' \to \mu(\alpha') \mid A' \to \alpha' \in P'\} = \{A' \to \alpha\delta \mid A' \in N', \mu(A') = A\delta, \text{ and } A \to \alpha \in P\}$.

It is a *selective k-combing* if furthermore $\mu(N') \subseteq N \cdot V^{\leq k}$.

We denote the elements of N' by $[A\delta]_i$, such that $\mu([A\delta]_i) = A\delta$, with an i subscript in $\mathbb{N}$ to differentiate nonterminals that share the same image by μ.

Note that, if $\mathcal{G}\,\mathsf{comb}\,\mathcal{G}'$, then there exists some k such that $\mathcal{G}'$ is a selective k-combing of $\mathcal{G}$, because $\mu(N')$ is a finite subset of $N \cdot V^*$. Another observation is that comb is transitive, and thus we can bypass any intermediate transformation by using the composition of the μ's. In fact, comb is also reflexive (using the identity on N for μ), and is thus a quasi order.

Grammar Cover. It is easy to see that a grammar and all its μ-combings are language equivalent. In fact, we can be more specific, and show that any μ-combing $\mathcal{G}' = \langle N', \Sigma, P', [S]_0\rangle$ of $\mathcal{G} = \langle N, \Sigma, P, S\rangle$ defines a *right-to-x cover* of $\mathcal{G}$ (see Nijholt (1980)), i.e. there exists a homomorphism h from P'^* to P^* such that

1. for all w in $L(\mathcal{G}')$ and right parses π' of w in $\mathcal{G}'$, $h(\pi')$ is a parse of w in $\mathcal{G}$, and
2. for all w in $L(\mathcal{G})$ there is a parse π of w in $\mathcal{G}$, such that there exists a right parse π' of w in $\mathcal{G}'$ with $h(\pi') = \pi$.

Indeed, defining h by

$$h([A\delta]_i \to \alpha) = A \to \mu(\alpha) \cdot \delta^{-1} \tag{1}$$

fits the requirements of a right-to-x cover.

Tree Mapping. Nevertheless, the right-to-x cover characterization is still somewhat unsatisfying, precisely because the exact derivation order x remains unknown. We solve this issue by providing a tree transformation that maps any derivation tree of $\mathcal{G}'$ to a derivation tree of $\mathcal{G}$. Besides allowing us to prove the language equivalence of $\mathcal{G}$ and $\mathcal{G}'$ (see Corollary 1), this transformation also allows us to map any parse tree of $\mathcal{G}'$—the grammar we use for parsing—to its corresponding parse tree of $\mathcal{G}$—the grammar we were interested in in the first place.

We express this transformation as a rewrite system over the set of *unranked forests* $\mathcal{F}(N \cup N' \cup \Sigma)$ over the set of symbols $N \cup N' \cup \Sigma$, defined by the abstract syntax

$$t ::= X(f) \qquad \text{(trees)}$$
$$f ::= \varepsilon \mid f \cdot t \qquad \text{(forests)}$$

where "X" ranges over $N \cup N' \cup \Sigma$ and "$\cdot$" denotes concatenation. Using unranked forests, our tree transformation has a very simple definition, using a rewrite system $\to_R$ with one rule per nonterminal $[AX_1 \cdots X_r]_i$ in N':

$$[AX_1 \cdots X_r]_i(x_0 \cdot X_1(x_1) \cdots X_r(x_r)) \to_R A(x_0) \cdot X_1(x_1) \cdots X_r(x_r) \tag{2}$$

with variables $x_0, x_1, \ldots, x_r$ ranging over $\mathcal{F}(N \cup N' \cup \Sigma)$. Clearly, $\to_R$ is noetherian and confluent, and we can consider the mapping that associates to a derivation tree t in $\mathcal{G}'$ its normal form $t{\downarrow_R}$ (see Appendix B.1 for details):

Proposition 1. *Let $\mathcal{G}$ be a CFG and $\mathcal{G}'$ a combing of $\mathcal{G}$.*

1. *If t' is a derivation tree of $\mathcal{G}'$, then $t'{\downarrow_R}$ is a derivation tree of $\mathcal{G}$.*
2. *If t is a derivation tree of $\mathcal{G}$, then there exists a derivation tree t' of $\mathcal{G}'$ such that $t = t'{\downarrow_R}$.*

Since $\to_R$ preserves tree yields, we obtain the language equivalence of $\mathcal{G}$ and $\mathcal{G}'$ as a direct corollary of Proposition 1:

Corollary 1 (Combings Preserve Languages)**.** *Let $\mathcal{G}$ be a k-extended CFG and $\mathcal{G}'$ a combing of $\mathcal{G}$. Then $L(\mathcal{G}) = L(\mathcal{G}')$.*

3.2 Selective ML Grammars

We define selML(k, m) grammars by analogy with the characterization proved in Theorem 1:

Definition 3 (Selective ML)**.** A grammar is *selML(k, m)* if there exists a selective k-combing of its k-extension that is LR(m).

Basic Properties. We now investigate the class of selML(k, m) grammars. As a first comparison, we observe that the uniform k-combing of a grammar is by definition a selective k-combing (by setting μ as the identity on $N \cdot V^{\leq k}$), hence the following lemma:

Lemma 1. *If a grammar is ML(k, m) for some k and m, then it is selML(k, m).*

As shown by $\mathcal{G}_2$, this grammar class inclusion is strict.

A second, more interesting comparison holds between selML(0, m) and LR(m). That a LR(m) grammar is selML(0, m) is immediate since comb is reflexive; the converse is not obvious at all, because a 0-combing can involve "duplicated" nonterminals, but holds nevertheless (see Appendix B.2 for details).

Lemma 2. *A reduced grammar is selML(0, m) if and only if it is LR(m).*

Recall that a context-free language can be generated by some LR(1) grammar if and only if it is deterministic (Knuth, 1965), thus selML languages also characterize deterministic languages:

Corollary 2 (Selective ML Languages)**.** *A context-free language has a selML grammar if and only if it is deterministic.*

Proof. Given a selML(k, m) grammar $\mathcal{G}$, we obtain an LR(m) grammar $\mathcal{G}'$ with a deterministic language, and equivalent to $\mathcal{G}$ by Corollary 1. Conversely, given a deterministic language, there exists an LR(1) grammar for it, which is also selML(0,1) by Lemma 2. □

Monotonicity. We should also mention that, unlike uniform ML, increasing k allows strictly more grammars to be captured by selML(k, m). Indeed, if a grammar is a selective k-combing of some grammar $\mathcal{G}$, then it is also a $k+1$-combing using the same μ (with an extra # endmarker), and remains LR(m).

Proposition 2. *If a grammar is selML(k, m) for some k and m, then it is selML(k', m') for all $k' \geq k$ and $m' \geq m$.*

Strictness can be witnessed thanks to the grammar family $(\mathcal{G}_3^k)_{k\geq 0}$ defined by

$$S \to Ac^kA' \mid Bc^kB',\ A \to cA \mid d,\ B \to cB \mid d,\ A' \to cA' \mid a,\ B' \to cB' \mid b \quad (\mathcal{G}_3^k)$$

where each $\mathcal{G}_3^k$ is selML($k+1$, 0), but not selML(k, m) for any m.

Ambiguity. As a further consequence of Proposition 1, we see that no ambiguous grammar can be selML(k, m) for any k and m.

Proposition 3. *If a grammar is selML(k, m) for some k and m, then it is unambiguous.*

Proof. Assume the opposite: an ambiguous grammar $\mathcal{G}$ has a selective k-combing $\mathcal{G}'$ that is LR(m). Being ambiguous, $\mathcal{G}$ has two different derivation trees t_1 and t_2 with the same yield w. As t_1 and t_2 are in normal form for $\to_R$, the sets of derivation trees of $\mathcal{G}'$ that rewrite into t_1 and t_2 are disjoint, and using Proposition 1 we can pick two different derivation trees t'_1 and t'_2 with $t_1 = t'_1{\downarrow_R}$ and $t_2 = t'_2{\downarrow_R}$. As $\to_R$ preserves tree yields, both t'_1 and t'_2 share the same yield w, which shows that $\mathcal{G}'$ is also ambiguous, in contradiction with $\mathcal{G}'$ being LR(m) and thus unambiguous. □

Again, this grammar class inclusion is strict, because the following unambiguous grammar for even palindromes is not selML(k, m) for any k or m, since its language is not deterministic:

$$S \to aSa \mid bSb \mid \varepsilon \quad (\mathcal{G}_4)$$

Undecidability. Let us first refine the connection between selML and LR in the case of linear grammars: recall that a CFG is *linear* if the right-hand side of each one of its productions contains at most one nonterminal symbol. A consequence is that right contexts in linear CFGs are exclusively composed of terminal symbols. In such a case, the selML(k, m) and LR($k+m$) conditions coincide (see Appendix B.2 for details):

Lemma 3. *Let $\mathcal{G}$ be a reduced linear grammar, and k and m two natural integers. Then $\mathcal{G}$ is selML(k, m) if and only if it is LR($k+m$).*

Note that in the non-linear case, the classes of selML(k, m) and LR($k+m$) grammars are incomparable. Nevertheless, we obtain as a consequence of Lemma 3:

Theorem 2. *It is undecidable whether an arbitrary (linear) context-free grammar is selML(k, m) for some k and m, even if we fix either k or m.*

Proof. Knuth (1965) has proven that it is undecidable whether an arbitrary linear context-free grammar is LR(n) for some n. □

4 Parser Construction

This section discusses how to directly construct an LR-type parser for a given grammar and fixed k and m values. The algorithm is *incremental*, in that it attempts to use as little right context as possible: this is interesting for efficiency reasons (much as incremental lookaheads in (Ancona et al., 1991; Parr and Quong, 1996)), and actually needed since more context does not necessarily lead to determinism (recall Section 2.3). The class of grammars for which the algorithm terminates successfully (i.e. results in a deterministic parser, without ever reaching a failure state) coincides with the class of selML(k, m) grammars (see propositions 4 and 5). An extended example of the construction will be given in Section 4.2.

4.1 Algorithm

Algorithm 1 presents the construction of an automaton from the k-extension of a grammar. We will call this the selML(k, m) automaton. In the final stages of the construction, the automaton will resemble an LR(m) automaton for a selective k-combing. Before that, states are initially constructed without right context. Right contexts are extended only where required to solve conflicts.

Items and States. The items manipulated by the algorithm are of form $([A\delta] \to \alpha \bullet \alpha', L)$, where $L \subseteq \Sigma^{\leq m}$ is a set of terminal lookahead strings, and where α and α' might contain nonterminals of the form $[B\beta]$, where $B \in N$ and $\beta \in V^{\leq k}$. Such nonterminals may later become nonterminals in the selective k-combing of the input grammar. To avoid notational clutter, we assume in what follows that B and $[B]$ are represented in the same way, or equivalently, that an occurrence of B in a right-hand side is implicitly converted to $[B]$ wherever necessary.

States are represented as sets of items. Each such set q is associated with three more sets of items. The first is its closure $\mathsf{close}(q)$. The second is $\mathsf{conflict}(q)$, which is the set of closure items that lead to a shift/reduce or reduce/reduce conflict with another item, either immediately in q or in a state reachable from q by a sequence of transitions. A conflict item signals that the closure step that predicted the corresponding rule, in the form of a non-kernel item, must be reapplied, but now from a nonterminal $[B\beta]$ with longer right context β. Lastly, the set $\mathsf{deprecate}(q)$ contains items that are to be ignored for the purpose of computing the Goto function.

Item Closure. The sets $\mathsf{close}(q)$, $\mathsf{conflict}(q)$ and $\mathsf{deprecate}(q)$ are initially computed from the kernel q alone. However, subsequent visits to states reachable from q may lead to new items being added to $\mathsf{conflict}(q)$ and then to $\mathsf{close}(q)$ and $\mathsf{deprecate}(q)$. How items in these three sets are derived from one another for given q is presented as the deduction system in Figure 2.

The *closure* step is performed as in conventional LR parsing, except that right context is copied to the right-hand side of a predicted rule. The *conflict detection* step introduces a conflict item, after a shift/reduce or reduce/reduce conflict appears among the derived items in the closure. Conflict items solicit additional right context, which

Algorithm 1: Construction of the selML(k, m) automaton for the k-extension of $\mathcal{G} = \langle N, \Sigma, P, S \rangle$, followed by construction of a selective k-combing.

States $\leftarrow \emptyset$
Transitions $\leftarrow \emptyset$
Agenda $\leftarrow \emptyset$
$q_{\text{init}} = \{(S^\dagger \rightarrow \bullet\, S\#^k, \{\varepsilon\})\}$
NewState(q_{init})
while Agenda $\neq \emptyset$ **do**
    $q \leftarrow$ pop(Agenda)
    remove (q, X, q') from Transitions for any X and q'
    apply Figure 2 to add new elements to the three sets associated with q
    for all $([A\delta] \rightarrow \alpha X \bullet \beta, L) \in$ conflict(q) **do**
        for all q' such that $(q', X, q) \in$ Transitions **do**
            AddConflict($([A\delta] \rightarrow \alpha \bullet X\mu(\beta), L), q'$)
        end for
    end for
    if there are no $([A\delta] \rightarrow \alpha X \bullet \beta, L) \in$ conflict(q) **then**
        $q_{max} \leftarrow$ close(q) $\setminus$ deprecate(q)
        for all X such that there is $([A\delta] \rightarrow \alpha \bullet X\beta, L) \in q_{max}$ **do**
            $q' \leftarrow$ Goto(q_{max}, X)
            if $q' \notin$ States **then**
                NewState(q')
            else
                for all $([A'\delta'] \rightarrow \alpha' X \bullet \beta', L) \in$ conflict(q') **do**
                    AddConflict($([A'\delta'] \rightarrow \alpha' \bullet X\mu(\beta'), L), q$)
                end for
            end if
            Transitions $\leftarrow$ Transitions $\cup \{(q, X, q')\}$
        end for
    end if
end while
construct a selective k-combing as explained in the running text

function NewState(q)
    close(q) $\leftarrow q$
    conflict(q) $\leftarrow \emptyset$
    deprecate(q) $\leftarrow \emptyset$
    States $\leftarrow$ States $\cup \{q\}$
    Agenda $\leftarrow$ Agenda $\cup \{q\}$
end function

function AddConflict($([A\delta] \rightarrow \alpha \bullet X\beta, L), q$)
    if $([A\delta] \rightarrow \alpha \bullet X\beta, L) \notin$ conflict(q) **then**
        conflict(q) $\leftarrow$ conflict(q) $\cup \{([A\delta] \rightarrow \alpha \bullet X\beta, L)\}$
        Agenda $\leftarrow$ Agenda $\cup \{q\}$
    end if
end function

$$\frac{([A\delta] \to \alpha \bullet [B\beta_1]\beta_2, L) \in \mathsf{close}(q)}{([B\beta_1] \to \bullet\, \gamma\beta_1, L') \in \mathsf{close}(q)} \left\{ \begin{array}{l} B \to \gamma \in P, \\ L' = \mathsf{First}_m(\beta_2 L) \end{array} \right. \quad \text{(closure)}$$

$$\frac{\begin{array}{c}([A_1\delta_1] \to \alpha_1 \bullet \beta_1, L_1) \in \mathsf{close}(q) \\ ([A_2\delta_2] \to \alpha_2 \bullet, L_2) \in \mathsf{close}(q)\end{array}}{([A_2\delta_2] \to \alpha_2 \bullet, L_2) \in \mathsf{conflict}(q)} \left\{ \begin{array}{l} (A_1\delta_1, \alpha_1, \beta_1) \neq (A_2\delta_2, \alpha_2, \varepsilon), \\ \mathsf{First}_m(\mu(\beta_1)L_1) \cap L_2 \neq \emptyset \end{array} \right. \quad \text{(conflict detection)}$$

$$\frac{\begin{array}{c}([A\delta] \to \alpha \bullet [B\beta], L) \in \mathsf{close}(q) \\ ([B\beta] \to \bullet\, \gamma, L) \in \mathsf{conflict}(q)\end{array}}{([A\delta] \to \alpha \bullet [B\beta], L) \in \mathsf{conflict}(q)} \quad \text{(conflict propagation)}$$

$$\frac{\begin{array}{c}([A\delta] \to \alpha \bullet [B\beta_1]X\beta_2, L) \in \mathsf{close}(q) \\ ([B\beta_1] \to \bullet\, \gamma, L') \in \mathsf{conflict}(q)\end{array}}{([A\delta] \to \alpha \bullet [B\beta_1 X]\beta_2, L) \in \mathsf{close}(q)} \left\{ \begin{array}{l} |\beta_1| < k, \\ L' = \mathsf{First}_m(X\beta_2 L), \end{array} \right. \quad \text{(extension)}$$

$$\frac{([B\beta] \to \bullet\, \gamma, L) \in \mathsf{conflict}(q)}{\bot} \left\{ \; |\beta| = k \right. \quad \text{(failure)}$$

$$\frac{([A\delta] \to \alpha \bullet [B\beta_1 X]\beta_2, L) \in \mathsf{close}(q)}{([A\delta] \to \alpha \bullet [B\beta_1]X\beta_2, L) \in \mathsf{deprecate}(q)} \quad \text{(deprecation)}$$

$$\frac{([A\delta] \to \alpha \bullet [B\beta_1 X]\beta_2, L) \in \mathsf{close}(q)}{([B\beta_1] \to \bullet\, \gamma', L') \in \mathsf{deprecate}(q)} \left\{ \begin{array}{l} B \to \gamma \in P, \mu(\gamma') = \gamma\beta_1, \\ L' = \mathsf{First}_m(X\beta_2 L), \end{array} \right. \quad \text{(deprecate closure)}$$

Figure 2: Closure of set q with local resolution of conflicts.

- may be available locally in the current state, as in step *extension*, where nonterminal $[B\beta_1]$ is extended to incorporate the following symbol X—we assume μ here is a generic "uncombing" homomorphism, turning a single nonterminal $[B\beta_1]$ into a string $B\beta_1 \in N \cdot V^{\leq k}$—, or
- if no more right context is available at the closure item from which a conflict item was derived, then the closure item itself becomes a conflict item, by step *conflict propagation*—propagation of conflicts across states is realized by Algorithm 1 and will be discussed further below—, or
- if there is ever a need for right context exceeding length k, then the grammar cannot be selML(k, m) and the algorithm terminates reporting failure by step *failure*.

Step *deprecation* expresses that an item with shorter right context is to be ignored for the purpose of computing the Goto function. The Goto function will be discussed further below. Similarly, step *deprecate closure* expresses that all items predicted from the item with shorter right context are to be ignored.

Main Algorithm. Initially, the agenda contains only the initial state, which is added in line 5. Line 7 of the algorithm removes an arbitrary element from the agenda and assigns it to variable q. At that point, either $\mathsf{close}(q) = q$ and $\mathsf{conflict}(q) = \mathsf{deprecate}(q) = \emptyset$ if q was not considered by line 7 before, or elements may have been added to $\mathsf{conflict}(q)$ since the last such consideration, which also requires updating of $\mathsf{close}(q)$ and $\mathsf{deprecate}(q)$, by Figure 2. By a change of the latter two sets, also the outgoing transitions may change. To

keep the presentation simple, we assume that all outgoing transitions are first removed (on line 8) and then recomputed. From line 10, conflicting items are propagated to states immediately preceding the current state, by one transition. Such a preceding state is then put on the agenda so that it will be revisited later.

Outgoing transitions are (re-)computed from line 15 onward. This is only done if no conflicting items had to be propagated to preceding states. Such conflict items would imply that q itself will not be reachable from the initial state in the final automaton, and in that case there would be no benefit in constructing outgoing transitions from q.

For the purpose of applying the Goto function, we are only interested in the closure items that have maximal right context, as all items with shorter context were found to lead to conflicts. This is the reason why we take the set difference $q_{max} = \mathsf{close}(q) \setminus \mathsf{deprecate}(q)$. The Goto function is defined much as usual:

$$\mathsf{Goto}(q_{max}, X) = \{([A\delta] \to \alpha X \bullet \beta, L) \mid ([A\delta] \to \alpha \bullet X\beta, L) \in q_{max}\} \,. \quad (3)$$

The loop from line 22 is very similar to that from line 10. In both cases, conflicting items are propagated from a state q_2 to a state q_1 along a transition (q_1, X, q_2). The difference lies in whether q_1 or q_2 is the currently popped element q in line 7. The propagation must be allowed to happen in both ways, as it cannot be guaranteed that no new transitions are found leading to states at which conflicts have previously been processed.

Combing Construction. After the agenda in Algorithm 1 becomes empty, only those states reachable from the initial state q_{init} via transitions in $\mathsf{Transitions}$ are relevant, and the remaining ones can be removed from States. From the reachable states, we can then construct a selective k-combing, with start symbol $S^\dagger$, as follows.

For each $q_n \in \mathsf{States}$ and $([A\delta] \to X_1 \cdots X_n \bullet, L) \in \mathsf{close}(q_n) \setminus \mathsf{deprecate}(q_n)$, some $n \geq 0$, find each choice of:

- $q_0, \ldots, q_{n-1}$,
- $\beta_0, \ldots, \beta_n$, with $\beta_n = \varepsilon$,

such that for $0 \leq j < n$,

- $(q_j, X_{j+1}, q_{j+1}) \in \mathsf{Transitions}$,
- $([A\delta] \to X_1 \cdots X_j \bullet X_{j+1}\beta_{j+1}, L) \in \mathsf{close}(q_j) \setminus \mathsf{deprecate}(q_j)$, and
- $\beta_j = \mu(X_{j+1})\beta_{j+1}$.

It can be easily seen that β_0 must be of the form $\alpha\delta$, for some rule $A \to \alpha$. For each choice of the above, now create a rule $Y_0 \to Y_1 \cdots Y_n$, where Y_0 stands for the triple $(q_0, A\delta, L)$, and for $1 \leq j \leq n$:

- if X_j is a terminal then $Y_j = X_j$, and
- if X_j is of the form $[B_j\gamma_j]$ then Y_j stands for the triple $(q_{j-1}, B_j\gamma_j, L_j)$, where $L_j = \mathsf{First}_m(\beta_j L)$.

We assume here that $\mu(Y_0) = A\delta$ and $\mu(Y_j) = B_j\gamma_j$ for $1 \leq j \leq n$.

4.2 Example

Example 2. Let us apply Algorithm 1 to the construction of a selML(2, 0) parser for $\mathcal{G}_{\text{odd}}$. The initial state is $q_{\text{init}} = \{S^\dagger \to \bullet S\#\#\}$ (there is no lookahead set since we set $m = 0$) and produces through the rules of Figure 2

$$\mathsf{close}(q_{\text{init}}) = \{S^\dagger \to \bullet S\#\#, S \to \bullet SdA, S \to \bullet c\} \,. \tag{4}$$

Fast-forwarding a little, the construction eventually reaches state $q_{Sd} = \{S \to Sd \bullet A\}$ with

$$\mathsf{close}(q_{Sd}) = \{S \to Sd \bullet A, A \to \bullet a, A \to \bullet ab\} \,, \tag{5}$$

which in turn reaches state $q_{Sda} = \{A \to a\bullet, A \to a \bullet b\}$ with

$$\mathsf{close}(q_{Sda}) = q_{Sda} \,, \tag{6}$$

$$\mathsf{conflict}(q_{Sda}) = \{A \to a\bullet\} \,. \tag{7}$$

As this item is marked as a conflict item, line 10 of Algorithm 1 sets

$$\mathsf{conflict}(q_{Sd}) = \{A \to \bullet a\} \,, \tag{8}$$

and puts q_{Sd} back in the agenda. Then, the *conflict propagation* rule is fired to set

$$\mathsf{conflict}(q_{Sd}) = \{A \to \bullet a, S \to Sd \bullet A\} \,, \tag{9}$$

and by successive backward propagation steps we get

$$\mathsf{conflict}(q_{\text{init}}) = \{S \to \bullet SdA\} \,. \tag{10}$$

The *extension* rule then yields

$$\mathsf{close}(q_{\text{init}}) = \{S^\dagger \to \bullet S\#\#, S \to \bullet SdA, S \to \bullet c, S^\dagger \to \bullet [S\#]\#, S \to \bullet [Sd]A\} \,, \tag{11}$$

which is closed to obtain

$$\begin{aligned}\mathsf{close}(q_{\text{init}}) = \{&S^\dagger \to \bullet S\#\#, S \to \bullet SdA, S \to \bullet c, S^\dagger \to \bullet [S\#]\#, S \to \bullet [Sd]A,\\ &[S\#] \to \bullet SdA\#, [S\#] \to \bullet c\#, [Sd] \to \bullet SdAd, [Sd] \to \bullet cd\} \,,\end{aligned} \tag{12}$$

and we can apply again the *extension* rule with the conflicting item $S \to \bullet SdA$:

$$\begin{aligned}\mathsf{close}(q_{\text{init}}) = \{&S^\dagger \to \bullet S\#\#, S \to \bullet SdA, S \to \bullet c, S^\dagger \to \bullet [S\#]\#, S \to \bullet [Sd]A,\\ &[S\#] \to \bullet SdA\#, [S\#] \to \bullet c\#, [Sd] \to \bullet SdAd, [Sd] \to \bullet cd,\\ &[S\#] \to \bullet [Sd]A\#, [Sd] \to \bullet [Sd]Ad\} \,.\end{aligned} \tag{13}$$

The *deprecate* and *deprecate closure* rules then yield

$$\begin{aligned}\mathsf{deprecate}(q) = \{&S^\dagger \to \bullet S\#\#, S \to \bullet SdA, S \to \bullet c, [S\#] \to \bullet SdA\#,\\ &[Sd] \to \bullet SdAd, \ldots\} \,.\end{aligned} \tag{14}$$

We leave the following steps to the reader; the resulting parser is displayed in Figure 3 (showing only items in $\mathsf{close}(q) \setminus \mathsf{deprecate}(q)$ in states).

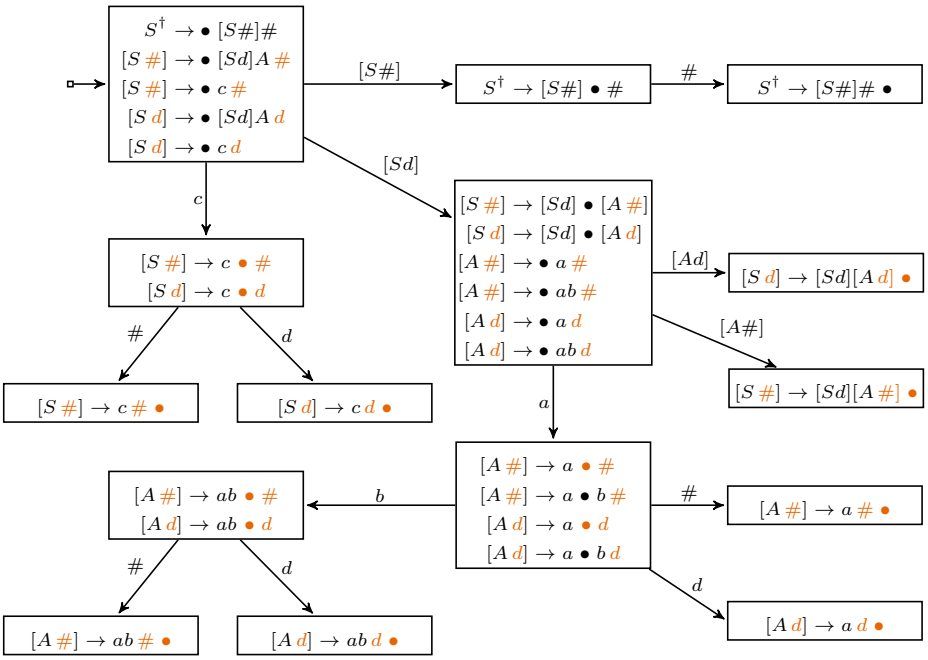

Figure 3: The selML(2, 0) parser for $\mathcal{G}_{\text{odd}}$.

4.3 Correctness

First observe that Algorithm 1 always terminates: the number of possible sets q, along with the growing sets $\mathsf{close}(q)$, $\mathsf{conflict}(q)$ and $\mathsf{deprecate}(q)$, is bounded.

Termination by the *failure* step of Figure 2 occurs only when we know that the resulting parser cannot be deterministic; conversely, successful termination means that a deterministic parser has been constructed. One could easily modify the construction to keep running in case of failure and output a nondeterministic parser instead, for instance to use a generalized LR parsing algorithm on the obtained parser.

The correctness of the construction follows from propositions 4 and 5 (see Appendix C for details).

Proposition 4. *If Algorithm 1 terminates successfully, then the constructed grammar is a selective k-combing. Furthermore, this combing is LR(m).*

Proof Idea. The structure of the selML(k, m) automaton and the item sets ensure that the constructed grammar satisfies all the requirements of a selective k-combing. Had this been non-LR(m), then there would have been further steps or failure. □

Proposition 5. *If the grammar is selML(k, m), then the algorithm terminates successfully.*

Proof Idea. The selML(k, m) automaton under construction reflects minimum right context for nonterminal occurrences in any selective k-combing with the LR(m) property. Furthermore, failure would imply that right context of length exceeding k is needed. □

As a consequence, we can refine the statement of Theorem 2 with

Corollary 3. *It is decidable whether an arbitrary context-free grammar is selML(k, m), for given k and m.*

5 Experimental Results

We have implemented a proof of concept of Algorithm 1, which can be downloaded from `http://www.cs.st-andrews.ac.uk/~mjn/code/mlparsing/`. Its purpose is not to build actual parsers for programming languages, but merely to check the feasibility of the approach and compare selML with uniform ML and more classical parsers. with our test suite and the

Grammar Collection. We investigated a set of *small* grammars that exhibit well-identified syntactic difficulties, to see whether they are treated correctly by a given parsing technique, or lie beyond its grasp. This set of grammars was compiled by Basten (2008) and extended in (Schmitz, 2010), containing mostly grammars for programming languages from the parsing literature and the comp.compilers archive, but also a few RNA grammars used by the bioinformatics community (Reeder et al., 2005).

Conflicts. As expected, we identified a few grammars that were not LALR(1) but were selML(k, m) for small values of k and m—one may bear in mind that any progress in this respect seems worthwhile for a grammar developer struggling with conflicts.

Example 3 (Tiger)**.** One such example is an excerpt from the Tiger syntax found at `http://compilers.iecc.com/comparch/article/98-05-030`. The grammar describes assignment expressions E, which are typically of the form "$L := E$" for L an lvalue.

$$E \to L \mid L := E \mid i[E] \text{ of } E \qquad\qquad L \to i \mid L[E] \mid L.i$$

The grammar is not LR(m) for any m, but is ML(3, 1) and ML(2, 2): a conflict arises between inputs of the form "$i[E]$ of E" and "$i[E] := E$", where the initial i should be kept as such and the parser should shift in the first case, and reduce to L in the second case. An ML(3, 1) or ML(2, 2) parser scans up to the "of" or ":=" token that resolves this conflict, across the infinite language generated by E.

Example 4 (Typed Pascal Declarations)**.** Another example is a version of Pascal identifier declarations with type checking performed at the syntax level, which was proposed by Tai (1979). The grammar is LR(2) and ML(1,0):

$$\begin{aligned} D &\to \text{var } IL\, IT\, ; \mid \text{var } RL\, RT\, ; \\ IL &\to i\, ,\, IL \mid i & IT &\to :\ \text{integer} \\ RL &\to i\, ,\, RL \mid i & RT &\to :\ \text{real} \end{aligned}$$

On an input like "var i , i , i : real ;", a conflict arises between the reductions of the last identifier i to either an integer list IL or a real list RL with ":" as terminal lookahead. By delaying these reductions, one can identify either an integer type IT or a real type RT.

Non-Monotonicity. We found that non-monotonic behaviour with uniform ML parsers occurs more often than expected. Here is one example in addition to the C++ example given in Section 2.1; more could be found in particular with the RNA grammars of Reeder et al. (2005).

Example 5 (Pascal Compound Statements)**.** The following is an excerpt from ISO Pascal and defines compound statements C in terms of ";"-separated lists of statements S:

$$C \to \text{begin } L \text{ end} \qquad L \to L \ ; \ S \mid S \qquad S \to \varepsilon \mid C$$

This is an LALR(1) and ML(1, 0) grammar, but it is not ML(2, 0): the nonterminal $[L; S]$ has a rule $[L; S] \to [L; S]; [S]$, giving rise to a nonterminal $[S]$ with rules $[S] \to \varepsilon \mid [C]$ and a shift/reduce conflict—in fact, this argument shows more generally that the grammar is not ML(k, 0) for even k.

Parser Size. Because selML parsers introduce new context symbols only when required, they can be smaller than the corresponding LR or uniform ML parsers, which carry full lookahead lengths in their items—this issue has been investigated for instance by Ancona et al. (1991) and Parr and Quong (1996) for LR and LL parsers. Our results are inconclusive as to the difference of parser size (in terms of numbers of states) between selML and LR. However, selML parsers tend to be considerably smaller than uniform ML parsers. For instance, the ML(1, 1) parser for the C++ grammar fragment of Section 2.1 has 50 states, while its selML(1, 1) parser only features 28 states.

In fact, we can make the argument more formal: consider the family of grammars $(G_4^j)_{j>0}$, each with rules:

$$\begin{array}{llll} S \to A \mid D, & A \to a \mid Ab \mid Ac, & D \to EF^{j-1}F \mid E'F^{j-1}F', & \\ F \to a \mid bF, & F' \to f \mid bF', & E \to e, \quad E' \to e\,. & \end{array} \tag{G_4^j}$$

The uniform ML(j, 0) parser for G_4^j has exponentially many states in j, caused by the rules $[Aw] \to aw$ for all w in $\{b, c\}^j$, while the selective ML(j, 0) parser has only a linear number of states, as there is no need for delays in that part of the grammar.

6 Related Work

Grammar Transformations and Coverings. The idea of using grammar transformations to obtain LR(1) or even simpler grammars has been thoroughly investigated in the framework of *grammar covers* (Nijholt, 1980). Among the most notable applications, Mickunas et al. (1976) provide transformations from LR(k) grammars into much simpler classes such as *simple LR(1)* or *(1,1)-bounded right context*; Soisalon-Soininen and Ukkonen (1979) transform *predictive LR(k)* grammars into LL(k) ones by generalizing the notion of left-corner parsing. Such techniques were often limited however to *right-to-right* or *left-to-right* covers, whereas our transformation is not confined to such a strict framework.

Parsing with Delays. A different notion of delayed reductions was already suggested by Knuth (1965) and later formalized by Szymanski and Williams (1976) as *LR(k, t) parsing*, where one of the t leftmost phrases in any rightmost derivation can be reduced using a lookahead of k symbols. The difference between the two notions of delay can be witnessed with linear grammars, which are LR(k, t) if and only if they are LR(k)—because there is always at most one phrase in a derivation—but selML(k, m) if and only if they are LR($k+m$)—as shown in Lemma 3.

Like selML languages, and unlike more powerful noncanonical classes, the class of LR(k, t) grammars characterizes deterministic context-free languages. The associated parsing algorithm is quite different however from that of selML parsing: it uses the two-stacks model of noncanonical parsing, where reduced nonterminals are pushed back at the beginning of the input to serve as lookahead in reductions deeper in the stack. Comparatively, selML parsing uses the conventional LR parsing tables with a single stack.

Selectivity. Several parser construction methods attempt to use as little "information" as possible before committing to a parsing action: Ancona et al. (1991) and Parr and Quong (1996) try to use as little lookahead as possible in LR(k) or LL(k) parsing, Demers (1977) generalizes left-corner parsing to delay decisions possibly as late as an LR parser, and Fortes Gálvez et al. (2006) propose a noncanonical parsing algorithm that explores as little right context as possible.

7 Concluding Remarks

Selective ML parsing offers an original balance between

- enlarging the class of admissible grammars, compared to LR parsing, while
- remaining a deterministic parsing technique, with linear-time parsing and exclusion of ambiguities,
- having a simple description as a grammar transformation, and
- allowing the concrete construction of LR parse tables.

This last point is also of interest to practitioners who have embraced general, nondeterministic parsing techniques (Kats et al., 2010): unlike noncanonical or regular-lookahead extensions, selML parsers can be used for nondeterministic parsing exactly like LR parsers. Having fewer conflicts than conventional LR parsers, they will resort less often to nondeterminism, and might be more efficient.

References

Ancona, M., Dodero, G., Gianuzzi, V., and Morgavi, M., 1991. Efficient construction of LR(k) states and tables. *ACM Transactions on Programming Languages and Systems*, 13(1):150–178. doi:10.1145/114005.102809.

Basten, H.J.S., 2008. The usability of ambiguity detection methods for context-free grammars. In Vinju, J. and Johnstone, A., editors, *LDTA'08*, *8th Workshop on Language Descriptions, Tools and Applications*, volume 238(5) of *Electronic Notes in Theoretical Computer Science*, pages 35–46. Elsevier. doi:10.1016/j.entcs.2009.09.039.

Bermudez, M.E. and Schimpf, K.M., 1990. Practical arbitrary lookahead LR parsing. *Journal of Computer and System Sciences*, 41(2):230–250. doi:10.1016/0022-0000(90)90037-L.

Bertsch, E. and Nederhof, M.J., 2007. Some observations on LR-like parsing with delayed reduction. *Information Processing Letters*, 104(6):195–199. doi:10.1016/j.ipl.2007.07.003.

Boullier, P., 1984. *Contribution à la construction automatique d'analyseurs lexicographiques et syntaxiques*. Thèse d'État, Université d'Orléans.

Čulik, K. and Cohen, R., 1973. LR-Regular grammars—an extension of LR(k) grammars. *Journal of Computer and System Sciences*, 7(1):66–96. doi:10.1016/S0022-0000(73)80050-9.

Demers, A.J., 1977. Generalized left corner parsing. In *POPL'77*, *4th Annual Symposium on Principles of Programming Languages*, pages 170–182. ACM. doi:10.1145/512950.512966.

Farré, J. and Fortes Gálvez, J., 2001. A bounded-connect construction for LR-Regular parsers. In Wilhelm, R., editor, *CC 2001*, *10th International Conference on Compiler Construction*, volume 2027 of *LNCS*, pages 244–258. Springer. doi:10.1007/3-540-45306-7_17.

Fortes Gálvez, J., Schmitz, S., and Farré, J., 2006. Shift-resolve parsing: Simple, linear time, unbounded lookahead. In Ibarra, O.H. and Yen, H.C., editors, *CIAA'06*, *11th International Conference on Implementation and Application of Automata*, volume 4094 of *LNCS*, pages 253–264. Springer. doi:10.1007/11812128_24.

Gosling, J., Joy, B., and Steele, G., 1996. *The Java™ Language Specification*. Addison-Wesley, first edition. ISBN 0-201-63451-1.

ISO, 1998. *ISO/IEC 14882:1998: Programming Languages — C++*. International Organization for Standardization, Geneva, Switzerland.

Kats, L.C., Visser, E., and Wachsmuth, G., 2010. Pure and declarative syntax definition: Paradise lost and regained. In *OOPSLA 2010*, *ACM international conference on Object oriented programming systems languages and applications*, pages 918–932. ACM. doi:10.1145/1869459.1869535.

Knuth, D.E., 1965. On the translation of languages from left to right. *Inform. and Cont.*, 8(6):607–639. doi:10.1016/S0019-9958(65)90426-2.

Leermakers, R., 1992. Recursive ascent parsing: from Earley to Marcus. *Theoretical Computer Science*, 104(2):299–312. doi:10.1016/0304-3975(92)90127-2.

Malloy, B.A., Power, J.F., and Waldron, J.T., 2002. Applying software engineering techniques to parser design: the development of a C# parser. In *SAICSIT'02*, pages 75–82. SAICSIT.

Marcus, M.P., 1980. *A Theory of Syntactic Recognition for Natural Language*. Series in Artificial Intelligence. MIT Press. ISBN 0-262-13149-8.

McPeak, S. and Necula, G.C., 2004. Elkhound: A fast, practical GLR parser generator. In Duesterwald, E., editor, *CC 2004*, *13th International Conference on Compiler Construction*, volume 2985 of *LNCS*, pages 73–88. Springer. doi:10.1007/b95956.

Mickunas, M.D., Lancaster, R.L., and Schneider, V.B., 1976. Transforming LR(k) grammars to LR(1), SLR(1), and (1,1) Bounded Right-Context grammars. *Journal of the ACM*, 23(3):511–533. doi:10.1145/321958.321972.

Nijholt, A., 1980. *Context-free grammars: Covers, normal forms, and parsing*, vol-

ume 93 of *LNCS*. Springer. doi:10.1007/3-540-10245-0.

Parr, T.J. and Quong, R.W., 1996. LL and LR translators need $k > 1$ lookahead. *ACM Sigplan Not.*, 31(2):27–34. doi:10.1145/226060.226066.

Reeder, J., Steffen, P., and Giegerich, R., 2005. Effective ambiguity checking in biosequence analysis. *BMC Bioinformatics*, 6:153. doi:10.1186/1471-2105-6-153.

Schmitz, S., 2010. An experimental ambiguity detection tool. *Science of Computer Programming*, 75(1–2):71–84. doi:10.1016/j.scico.2009.07.002.

Sippu, S. and Soisalon-Soininen, E., 1990. *Parsing Theory, Vol. II: LR(k) and LL(k) Parsing*, volume 20 of *EATCS Monographs on Theoretical Computer Science*. Springer. ISBN 3-540-51732-4.

Soisalon-Soininen, E. and Ukkonen, E., 1979. A method for transforming grammars into LL(k) form. *Acta Informatica*, 12(4):339–369. doi:10.1007/BF00268320.

Szymanski, T.G. and Williams, J.H., 1976. Noncanonical extensions of bottom-up parsing techniques. *SIAM Journal on Computing*, 5(2):231–250. doi:10.1137/0205019.

Tai, K.C., 1979. Noncanonical SLR(1) grammars. *ACM Transactions on Programming Languages and Systems*, 1(2):295–320. doi:10.1145/357073.357083.

The following appendices contain material that will not be part of the final paper, if accepted.

A Uniform ML Parsers

We recall here the uniform ML(k, m) parser construction of Bertsch and Nederhof (2007). The states are defined as sets of (k, m)-items $[A\delta \to \beta \bullet \gamma, x]$, where $A \to \alpha$ is a rule in P, $|\delta| \leq k$, $\alpha\delta = \beta\gamma$, and $x \in \Sigma^m$. Define a set of items $\mathsf{close}(q)$, where q is a set of items, as the smallest solution of

$$\mathsf{close}(q) = q \cup \{[B(k:\gamma) \to \bullet\, \beta(k:\gamma), y] \mid [\delta \to \alpha \bullet B\gamma, x] \in \mathsf{close}(q), \\ B \to \beta \in P, y \in \mathsf{First}_m((\gamma:k)x)\}$$

where $\gamma : k$ is the suffix of γ such that $\gamma = (k:\gamma)(\gamma:k)$ and $\mathsf{First}_m(\alpha) = \{m : w \in \Sigma^{\leq m} \mid \alpha \Rightarrow^* w\}$ for any α in V^*. The reductions allowed in state q by the parser are defined as

$$\mathsf{Reductions}(q) = \{[\delta \to \alpha, x] \mid [\delta \to \alpha \bullet, x] \in q\}$$

a transition by the following Goto function

$$\mathsf{Goto}(q, \beta) = \mathsf{close}(\{[\delta \to \alpha a \bullet \gamma, x] \mid [\delta \to \alpha \bullet a\gamma, x] \in q \text{ and } a \in \Sigma\} \\ \cup \{[\delta \to \alpha B\beta \bullet \gamma, x] \mid [\delta \to \alpha \bullet B\beta\gamma, x] \in q, \beta = k:\beta\gamma \text{ and } B \in N\})$$

and the initial state q_0 as

$$q_0 = \mathsf{close}(\{[S^\dagger \to \bullet\, S\#^k, \#^m]\}) \ .$$

Figure 4 provides an example of the Marcus-Leermakers construction on $\mathcal{G}_{\text{odd}}$. Note the conflict in the dashed state.

B Grammar Transformation

B.1 Combing Relations

Remark 1 (Ranked vs. Unranked Trees). We contemplated for a while expressing the transformation performed by $\to_R$ in a ranked setting, using multi bottom-up tree transducers, but the apparatus felt quite heavy. In fact, one could also define combings through macro tree transductions of a very restricted form, but the list of restrictions (linear, non-erasing, order preserving, Σ-preserving, local, etc.) would be daunting.

Lemma 4. *The pair $(\mathcal{G}', h)$ is a right-to-x cover for $\mathcal{G}$.*

Proof. Immediate consequence of Proposition 1 and the fact that $\to_R$ preserves tree yields. □

Lemma 5. *The rewrite relation $\to_R$ is noetherian and confluent.*

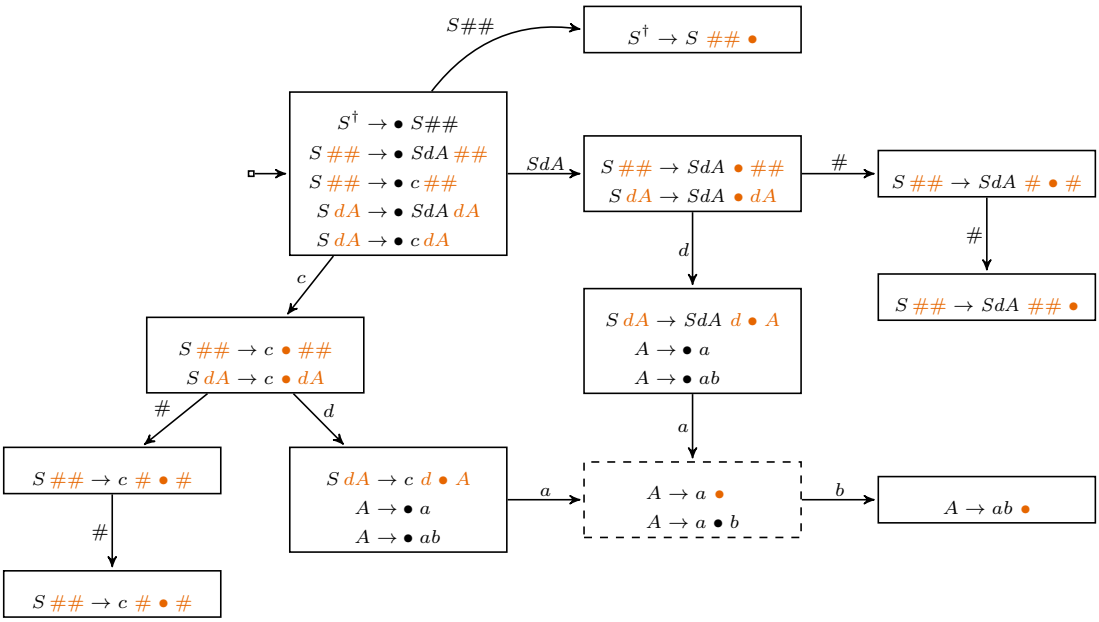

Figure 4: The uniform ML(2, 0) parser for $\mathcal{G}_{\text{odd}}$.

Proof. The fact that $\to_R$ is noetherian is immediate, since the number of symbols of N' that appear as node labels decreases strictly at each application of $\to_R$, and since the trees we consider are finite.

Let us prove that $\to_R$ is *locally confluent*: for all x, y, and z in $\mathcal{F}(N \cup N' \cup \Sigma)$, $x \to_R y$ and $x \to_R z$ imply that there exists $x' \in \mathcal{F}(N \cup N' \cup \Sigma)$ such that $y \to_R^* x'$ and $z \to_R^* x'$. Indeed, either $y = z$ and we can take $x' = y$, or the two rewrites from x have targeted two nonterminals $[AX_1 \dots X_q]_i$ and $[CY_1 \dots Y_r]_j$ in N' at two different nodes of x. Observe that, in the latter case, the children of $[CY_1 \dots Y_r]_j$ labeled by $Y_1, \dots, Y_r$ in N have not been modified by the rewrite at $[AX_1 \dots X_q]_i$, and we can thus still rewrite at $[CY_1 \dots Y_r]_j$ in y, and symmetrically in z. Thus we can obtain a single tree x' after a single rewrite step from either y or z.

We conclude by Newman's Lemma that $\to_R$ is globally confluent. □

Proposition 6. *Let $\mathcal{G}$ be a CFG and $\mathcal{G}'$ a combing of $\mathcal{G}$.*

1. *If t' is a derivation tree of $\mathcal{G}'$, then $t'{\downarrow_R}$ is a derivation tree of $\mathcal{G}$.*
2. *If t is a derivation tree of $\mathcal{G}$, then there exists a derivation tree t' of $\mathcal{G}'$ such that $t = t'{\downarrow_R}$.*

Proof of 1. First observe that, since $\to_R$ is confluent, any strategy for ordering rewrites will lead to the same normal form $t' \downarrow_R$. Here we use a bottom-up strategy, starting from the leaves (of form $X(\varepsilon)$) of t' up to its root $S'(f)$.

We generalize the notion of derivation trees to allow any root label and not just the axiom S of the grammar. Let us prove by induction over f that

Claim 4. If $X(f)$ is a derivation tree of $\mathcal{G}'$, then $X(f){\downarrow_R}$ is a sequence $X_0(f_0)X_1(f_1)\cdots X_r(f_r)$ of derivation trees of $\mathcal{G}$ with $\mu(X) = X_0X_1\cdots X_r$.

Note that the claim implies the desired result, since a derivation tree $t' = S'(f)$ of $\mathcal{G}'$ is then associated to a derivation tree $t'{\downarrow_R} = \mu(S')(f_0) = S(f_0)$ of $\mathcal{G}$ (recall condition 1).

By condition 2, the case where X is a terminal symbol in Σ is immediate, and we consider the one where $X = [AX_1 \cdots X_r]_i$ is in N'; we want to show that $[AX_1 \cdots X_r]_i(f){\downarrow_R}$ is a sequence $A(f_0)X_1(f_1) \cdots X_r(f_r)$ of derivation trees of $\mathcal{G}$.

By definition of a derivation tree, the forest f matches the right hand side of some production $[AX_1 \cdots X_r]_i \to \alpha'$ of P', which by condition 4, verifies $\mu(\alpha') = \alpha X_1 \cdots X_r$ for some $A \to \alpha$ in P. Write $\alpha = Y_1 \cdots Y_m$; by induction hypothesis on f and the fact that $(f \cdot f'){\downarrow_R} = (f{\downarrow_R}) \cdot (f'{\downarrow_R})$, we deduce that

$$f{\downarrow_R} = Y_1(f'_1) \cdots Y_m(f'_m) \cdot X_1(f_1) \cdots X_r(f_r) \tag{15}$$

is a sequence of derivation trees of $\mathcal{G}$. Then

$$[AX_1 \cdots X_r]_i(f){\downarrow_R} = A(Y_1(f'_1) \cdots Y_m(f'_m)) \cdot X_1(f_1) \cdots X_r(f_r) \tag{16}$$

is indeed a sequence of derivation trees of $\mathcal{G}$. □

Proof of 2. The idea is to revert the computation exhibited for the proof of item 1 to build a tree t':

Claim 5. If $f = X_0(f_0)X_1(f_1) \cdots X_r(f_r)$ is a sequence of derivation trees of $\mathcal{G}$ with $\mu(X) = X_0X_1 \cdots X_r$ for some X in N', then there exists a sequence f' s.t. $X(f')$ is a derivation tree of $\mathcal{G}'$ verifying $X(f'){\downarrow_R} = f$.

Item 2 then follows from condition 1, since a derivation tree t of $\mathcal{G}$ is a sequence $S(f)$ with $\mu(S') = S$, for which Claim 5 yields the existence of a derivation tree $t' = S(f')$ with $t'{\downarrow_R} = t$.

We prove the claim by induction on f. Again, the case where X_0 is a terminal in Σ is trivial, since then $r = 0$ and $f_0 = \varepsilon$, and setting $X = X_0$ fits the statement of Claim 5 by condition 2.

Let us therefore consider the case where X_0 is a nonterminal in N. Then f_0 has to match the right hand side of some production $X_0 \to \alpha$ in P, and by condition 4 there exists a corresponding production $X \to \alpha'$ in P' s.t. $\mu(\alpha') = \alpha X_1 \cdots X_r$. If we write $\alpha' = Y_1 \cdots Y_m$, then the sequence $f_0 \cdot X_1(f_1) \cdots X_r(f_r)$ can be seen as a concatenation of factors $Z_{i,1}(f'_{i,1}) \cdots Z_{i,r_i}(f'_{i,r_i})$ with $\mu(Y_i) = Z_{i,1} \cdots Z_{i,r_i}$. Applying the induction hypothesis on each factor yields the existence of derivation trees $Y_i(f''_i)$ of $\mathcal{G}'$ verifying $Y_i(f''_i){\downarrow_R} = Z_{i,1}(f'_{i,1}) \cdots Z_{i,r_i}(f'_{i,r_i})$, from which we deduce

$$[X_0X_1 \cdots X_r]_i(Y_1(f''_1) \cdots Y_m(f''_m)){\downarrow_R} = X_0(f_0)X_1(f_1) \cdots X_r(f_r) \tag{17}$$

as desired. □

B.2 Selective ML Grammars

Lemma 6. *A reduced grammar is selML(0, m) if and only if it is LR(m).*

Proof. First observe that, if a grammar is LR(m), then it is also selML(0, m) if we take $N' = N$ and the identity for μ.

Conversely, let us prove that if $\mathcal{G}$ is not LR(m), then it is not selML(0, m). Consider for this a 0-combing $\mathcal{G}'$ of the 0-extension of $\mathcal{G}$, and a conflict in the m-extension of $\mathcal{G}$:

$$\begin{aligned} &S^\dagger \Rightarrow^*_{\mathrm{rm}} \delta_1 Au \Rightarrow_{\mathrm{rm}} \delta_1\alpha u = \gamma u && \delta_1 \neq \delta_2 \text{ or } A \neq B \text{ or } \alpha \neq \beta \\ &S^\dagger \Rightarrow^*_{\mathrm{rm}} \delta_2 Bv \Rightarrow_{\mathrm{rm}} \delta_2\beta wv = \gamma wv && m : u = m : wv \end{aligned}$$

Because $\mathcal{G}$ is reduced, there exists a derivation

$$\gamma \Rightarrow^* w' \tag{18}$$

for some w'. We obtain in the m-extension of $\mathcal{G}'$

$$\begin{aligned} &S^\dagger \Rightarrow^*_{\mathrm{rm}} \delta'_1 A'u \Rightarrow_{\mathrm{rm}} \delta'_1\alpha' u = \gamma'_1 u && \delta'_1 \neq \delta'_2 \text{ or } A' \neq B' \text{ or } \alpha' \neq \beta' \\ &S^\dagger \Rightarrow^*_{\mathrm{rm}} \delta'_2 B'v \Rightarrow_{\mathrm{rm}} \delta'_2\beta' v = \gamma'_2 wv && m : u = m : wv \end{aligned}$$

where

$$\mu(A') = A \quad \mu(B') = B \quad \mu(\delta'_1) = \delta_1 \quad \mu(\delta'_2) = \delta_2 \quad \mu(\gamma'_1) = \mu(\gamma'_2) = \gamma\ .$$

Observe that this is an LR(m) conflict situation in $\mathcal{G}'$ if $\gamma'_1 = \gamma'_2$; assuming the contrary, we can write $\gamma'_1 = \gamma' C'_1 \gamma''_1$ and $\gamma'_2 = \gamma' C'_2 \gamma''_2$ where γ' is the longest common prefix of γ'_1 and γ'_2, and $\mu(C'_1) = \mu(C'_2) = C$. Pick the production $C \to \rho$ used in (18); we get the existence of two derivations in $\mathcal{G}'$

$$\begin{aligned} &S^\dagger \Rightarrow^*_{\mathrm{rm}} \gamma' C'_1 \gamma''_1 u \Rightarrow^*_{\mathrm{rm}} \gamma' C'_1 w'' u \Rightarrow_{\mathrm{rm}} \gamma' \rho'_1 w'' u && \mu(\rho'_1) = \rho \\ &S^\dagger \Rightarrow^*_{\mathrm{rm}} \gamma' C'_2 \gamma''_2 wv \Rightarrow^*_{\mathrm{rm}} \gamma' C'_2 w'' wv \Rightarrow_{\mathrm{rm}} \gamma' \rho'_2 w'' wv && \mu(\rho'_2) = \rho \end{aligned}$$

where w'' is the corresponding suffix of w' from (18). Because $m : w''u = m : w''wv$, this is an LR(m) conflict in $\mathcal{G}'$ if $\rho'_1 = \rho'_2$. Otherwise, we can again pick the longest common prefix of ρ'_1 and ρ'_2 and keep unfolding the derivation in (18); as the latter is finite, this process eventually terminates and allows to exhibit a conflict. □

Lemma 7. *Let $\mathcal{G}$ be a reduced linear grammar, and k and m two natural integers. Then $\mathcal{G}$ is selML(k, m) if and only if it is LR($k + m$).*

Proof of LR($k + m$) implies selML(k, m). Assume $\mathcal{G}$ is not selML(k, m), and consider the following particular k-combing $\mathcal{G}' = \langle N', \Sigma, P', [S^\dagger]\rangle$ of the k-extension of $\mathcal{G}$:

$$N' = \{[Au] \mid A \in N \text{ and } u \in \mathsf{Follow}_k(A)\}$$

where $\mathsf{Follow}_k(A) = \{k : w \in \Sigma^{\leq k} \mid S \Rightarrow^*_{\mathrm{rm}} \delta Aw\}$ and $\mu([Au]) = Au$ as usual, and

$$\begin{aligned} P' = &\{[Au] \to w[Bv]v' \mid A \to wBw' \in P, vv' = w'u, |v| = k\} \\ &\cup \{[Au] \to wu \mid A \to w \in P\}\ . \end{aligned}$$

Then $\mathcal{G}'$ is not LR(m), thus there is a conflict in its m-extension

$$S^\dagger \Rightarrow^* w_1[Au_1]v_1 \Rightarrow w_1\alpha' v_1 = \gamma' v_1 \qquad w_1 \neq w_2 \text{ or } [Au_1] \neq [Bu_2] \text{ or } \alpha' \neq \beta'$$
$$S^\dagger \Rightarrow^* w_2[Bu_2]v_2 \Rightarrow w_2\beta' v_2 = \gamma' v' v_2 \qquad m : v_1 = m : v' v_2$$

where $|u_1| = |u_2| = k$. But then, we have in the $(m+k)$-extension of $\mathcal{G}$

$$\begin{aligned} S^\dagger &\Rightarrow^* w_1 A u_1 v_1 \Rightarrow w_1 \alpha u_1 v_1 = \gamma u_1 v_1 && \mu(\alpha') = \alpha u_1 \\ S^\dagger &\Rightarrow^* w_2 B u_2 v_2 \Rightarrow w_2 \beta u_2 v_2 = \gamma v u_2 v_2 && \mu(\beta') = \beta u_2 \\ \mu(\gamma') &= \gamma u_1 = \gamma v'' \end{aligned}$$

such that $v''v' = vu_2$. As $k+m : u_1v_1 = k+m : v''v'v_2 = k+m : vu_2v_2$, the only problematic case in order to exhibit an LR($k+m$) conflict is that of $w_1 = w_2$, $\alpha = \beta$, and $A = B$, but $u_1 \neq u_2$. Observe however that α' then needs to be a prefix of β', as $\gamma'\alpha' = w_1\alpha' = w_2\alpha'$ and $\gamma' v' = w_2\beta'$. As we are working with a k-extended grammar, $|u_1| = |u_2| = k$, hence $|\mu(\alpha')| = |\mu(\beta')|$, hence $|\alpha'| = |\beta'|$ (as they need to both contain a nonterminal symbol or not contain a nonterminal symbol, and $|\mu(C)| = k+1$ for all $C \neq S^\dagger$). Finally, $\alpha' = \beta'$, which is incompatible with $A = B$ and $u_1 \neq u_2$. □

Proof of selML(k, m) implies LR(k + m). Assume $\mathcal{G}$ is not LR($k+m$); then we have a conflict in its $(k+m)$-extension:

$$S^\dagger \Rightarrow^* u_1 A v_1 \Rightarrow u_1 u_1' X v_1' v_1 = \gamma v_1 \qquad u_1 \neq u_2 \text{ or } A \neq B \text{ or } u_1' X v_1' \neq u_2' Y w v_2'$$
$$S^\dagger \Rightarrow^* u_2 B v_2 \Rightarrow u_2 u_2' Y w v_2' v_2 = \gamma v_2' v_2 \quad k+m : v_1 = k+m : v_2' v_2$$

Note that, if $X \in N$ or $Y \in N$, then $X = Y$.

Consider now any k-combing $\mathcal{G}'$ of the k-extension of $\mathcal{G}$; the following situation occurs in the m-extension of $\mathcal{G}'$:

$$\begin{aligned} S^\dagger &\Rightarrow^* u_1[Aw_1]_i v_1'' \Rightarrow u_1 \alpha v_1'' && v_1 = w_1 v_1'' && \mu(\alpha) = u_1' X v_1' w_1 \\ S^\dagger &\Rightarrow^* u_2[Bw_2]_j v_2'' \Rightarrow u_2 \beta v_2'' && v_2 = w_2 v_2'' && \mu(\beta) = u_2' Y w v_2' w_2 \end{aligned}$$

If X or Y is in Σ, then both are, and $u_1\alpha v_1'' = \gamma w_1 v_1''$ and $u_2\beta v_2'' = \gamma v_2' w_2 v_2''$ allow to exhibit an LR(m) conflict in $\mathcal{G}'$: since $|w_1| \leq k$, we can rewrite $\gamma v_2' w_2 v_2''$ as $\gamma w_2' w_2''$ such that $\gamma w_2' = \gamma w_1$, and we have $m : v_1'' = m : w_2''$ and $u_1 \neq u_2$ or $[Aw_1]_i \neq [Bw_2]_j$ or $\alpha \neq \beta$.

Otherwise, $X = Y$ but the combing might associate two different nonterminals X' and Y' of N' to them (or if they are the same nonterminal we immediately obtain an LR(m) conflict as in the terminal case). However, applying the same reasoning as in the proof of Lemma 2, because $\mathcal{G}$ is reduced, we will eventually reach a case where a conflict arises. □

C Parser Construction

Proposition 7. *If Algorithm 1 terminates successfully (i.e. without the failure step being triggered), then the constructed grammar is a selective k-combing. Furthermore, this combing is LR(m).*

Proof. In order to prove that the grammar is a selective combing it suffices to show that:

- $S^\dagger$ is a nonterminal in the grammar, and
- for every nonterminal $[A\delta]_i$ in the grammar and each rule $A \to \alpha \in P \cup \{S^\dagger \to S\#^k\}$, there is exactly one rule $[A\delta]_i \to \beta$ with $\mu(\beta) = \alpha\delta$.

A central observation is that if the set $\mathsf{close}(q_0) \setminus \mathsf{deprecate}(q_0)$ of a reachable state q_0 contains an item $([B\delta'] \to \gamma_1 \bullet [A\delta]\gamma_2, L')$, and $A \to \alpha \in P$, then there is precisely one choice of:

- $q_1, \ldots, q_n$,
- $X_1, \ldots, X_n$,
- $\beta_0, \ldots, \beta_n$, with $\beta_0 = \alpha\delta$ and $\beta_n = \varepsilon$,

such that for $0 \leq j < n$,

- $(q_j, X_{j+1}, q_{j+1}) \in \mathsf{Transitions}$,
- $([A\delta] \to X_1 \cdots X_j \bullet X_{j+1}\beta_{j+1}, L) \in \mathsf{close}(q_j) \setminus \mathsf{deprecate}(q_j)$, and
- $\beta_j = \mu(X_{j+1})\beta_{j+1}$,

and $([A\delta] \to X_1 \cdots X_n \bullet, L) \in \mathsf{close}(q_n) \setminus \mathsf{deprecate}(q_n)$, where $L = \mathsf{First}_m(\gamma_2 L')$. This follows from the rules in Figure 2 and the definition of the Goto function. In particular, where a closure item with longer right context is added, the corresponding item with shorter right context is deprecated.

In terms of the selective combing, the above implies that for every nonterminal $[A\delta]_i$ and rule $A \to \alpha$ (excepting $S^\dagger \to S\#^k$ for now) there must be exactly one rule $[A\delta]_i \to \beta$ with $\mu(\beta) = \alpha\delta$. The special case of $S^\dagger \to S\#^k$ can be treated similarly.

That the grammar is a selective k-combing follows from the fact that the inference rules in Figure 2 do not allow right context longer than k.

In order to prove that the combing is LR(m), we first show that for each state of the LR(m) automaton for the combing there is a corresponding state q of the selML(k, m) automaton. Where the LR(m) state has an item $(Y_0 \to Y_1 \cdots Y_j \bullet Y_{j+1} \cdots Y_n, x)$, where $j < n$, the selML(k, m) state has an item $([A\delta] \to X_1 \cdots X_j \bullet X_{j+1}\beta_{j+1}, L) \in \mathsf{close}(q) \setminus \mathsf{deprecate}(q)$, where $\mu(Y_0) = A\delta$, $\mu(Y_i) = \mu(X_i)$ for $1 \leq i \leq j+1$, $\mu(Y_{j+2} \cdots Y_n) = \beta_{j+1}$ and $x \in L$. Where the LR(m) state has an item $(Y_0 \to Y_1 \cdots Y_n \bullet, x)$, the selML($k$, m) state has an item $([A\delta] \to X_1 \cdots X_n \bullet, L) \in \mathsf{close}(q_n) \setminus \mathsf{deprecate}(q_n)$, where $\mu(Y_0) = A\delta$, $\mu(Y_i) = \mu(X_i)$ for $1 \leq i \leq n$ and $x \in L$. This can be proven by induction, starting at the initial state.

Next, it can be shown that if there were a conflict in an LR(m) state, then a similar conflict had arisen in a corresponding selML(k, m) state, and this had triggered further restructuring of the automaton before termination of the algorithm, leading to a contradiction. □

Proposition 8. *If the grammar is selML(k, m), then the algorithm terminates successfully.*

Proof. We rely on the following invariant of Algorithm 1. For each state q reachable through n transitions (q_0, X_1, q_1), …, $(q_{n-1}, X_n, q_n) \in \mathsf{Transitions}$ with $q_0 = q_{\text{init}}$ and $q_n = q$, and for each item $([A\delta] \to \alpha \bullet [B\beta_1 X]\beta_2, L) \in$

$\mathsf{close}(q)$, any combing for the k-extension of the input grammar that is LR(m) cannot allow:

$$[S^\dagger] \Rightarrow^*_{\mathrm{rm}} \gamma[B\beta_1]_i u \tag{19}$$

where $\mu(\gamma) = \mu(X_1 \cdots X_n)$ and $m : u \in \mathsf{First}_m(X\beta_2 L)$. In other words, if the selML(k, m) automaton has constructed a right context $\beta_1 X$ for nonterminal B given left context $\mu(X_1 \cdots X_n)$ and lookahead in $\mathsf{First}_m(X\beta_2 L)$, then no combing with corresponding left context and lookahead can use shorter right context β_1 without causing a conflict in the LR(m) parser.

As long as no right context has been extended (step 'extension' in Figure 2), the above invariant holds trivially. That the invariant is preserved by steps extending right context follows from the observation that a conflict detected in the selML(k, m) automaton implies a corresponding conflict in the LR(m) automaton.

In more detail, the 'conflict detection' step from Figure 2 is triggered by the presence of:

- $([A_1\delta_1] \to \alpha_1 \bullet \beta_1, L_1) \in \mathsf{close}(q)$
- $([A_2\delta_2] \to \alpha_2 \bullet, L_2) \in \mathsf{close}(q)$

with $(A_1\delta_1, \alpha_1, \beta_1) \neq (A_2\delta_2, \alpha_2, \varepsilon)$ and $\mathsf{First}_m(\mu(\beta_1)L_1) \cap L_2 \neq \emptyset$. Let q be reachable through n transitions (q_0, X_1, q_1), $\ldots$, $(q_{n-1}, X_n, q_n) \in \mathsf{Transitions}$ with $q_0 = q_{\mathrm{init}}$ and $q_n = q$. In a combing with right contexts of similar length, we would have:

$$[S^\dagger] \Rightarrow^*_{\mathrm{rm}} \gamma_1[A_1\delta_1]_{i_1} u \Rightarrow_{\mathrm{rm}} \gamma_1\alpha_1\beta_1 u = \gamma_3\beta_1 u$$
$$[S^\dagger] \Rightarrow^*_{\mathrm{rm}} \gamma_2[A_2\delta_2]_{i_2} v \Rightarrow_{\mathrm{rm}} \gamma_2\alpha_2 v = \gamma_3 v$$

where $\beta_1 \Rightarrow^*_{\mathrm{rm}} w$ for some w with $m : wu = m : v \in \mathsf{First}_m(\mu(\beta_1)L_1) \cap L_2$ and $\mu(\gamma_3) = \mu(X_1 \cdots X_n)$. (Recall that we use subscripts such as i_1 and i_2 to differentiate nonterminals of a combing that share the same image by μ, as motivated in Section 3.) This implies a LR(m) conflict. Note that if one constructs a combing with extended right context for the occurrence of A_1 only, such that we have:

$$[S^\dagger] \Rightarrow^*_{\mathrm{rm}} \gamma'_1[A_1\delta_1\delta_3]_{i'_1} u' \Rightarrow_{\mathrm{rm}} \gamma'_1\alpha'_1\beta'_1 u' = \gamma'_3\beta'_1 u'$$
$$[S^\dagger] \Rightarrow^*_{\mathrm{rm}} \gamma'_2[A_2\delta_2]_{i'_2} v \Rightarrow_{\mathrm{rm}} \gamma'_2\alpha'_2 v = \gamma'_3 v$$

with $\mu(\gamma'_1) = \mu(\gamma_1)$, $\mu(\gamma'_2) = \mu(\gamma_2)$, and $m : u \in \mathsf{First}_m(\delta_3 u')$, then the conflict is still present. Therefore, any combing that is LR(m) must at least have extended right context for A_2, given left context and lookahead as above. Extension of right context δ_2, by at least one more symbol X to become $\delta_2 X$, is achieved by the interaction of the inference rules (in particular steps 'extension' and 'conflict propagation') and the propagation of conflict items across states.

It follows that if an item $([B\beta] \to \bullet\, \gamma, L) \in \mathsf{conflict}(q)$ occurs with $|\beta| = k$, then there is a LR(m) conflict that cannot be resolved with a k-combing, and the grammar cannot be selML(k, m). □