LABORATOIRE

INFORMATIQUE, SIGNAUX ET SYSTÈMES
DE SOPHIA ANTIPOLIS
UMR 6070

NONCANONICAL LALR(1) PARSING

Sylvain Schmitz

Projet LANGAGES

Rapport de recherche
ISRN I3S/RR–2005-21–FR

Août 2005

LABORATOIRE I3S: Les Algorithmes / Euclide B – 2000 route des Lucioles – B.P. 121 –
06903 Sophia-Antipolis Cedex, France – Tél. (33) 492 942 701 – Télécopie : (33) 492 942 898
http://www.i3s.unice.fr/I3S/FR/

RÉSUMÉ :

La construction d'analyseurs syntaxiques LALR(1) non canoniques est présentée. La classe des grammaires NLALR(1) contient strictement les classes des grammaires NSLR(1) et LALR(1). Les langages reconnus peuvent être non déterministes, mais les automates générés analysent leur entrée de manière déterministe en temps linéaire, utilisant l'entrée comme seconde pile. La construction est une extension naturelle à la construction LALR(1) canonique.

MOTS CLÉS :

Analyse syntaxique, lookahead LR, non canonique, automates à deux piles, grammaires algébriques

ABSTRACT:

A construction for noncanonical LALR(1) parsers is presented. The class of NLALR(1) grammars is a proper superclass of the NSLR(1) and LALR(1) grammar classes. The recognized languages include some non deterministic languages, but the generated machines parse their input deterministically in linear time using the input as a second stack. The construction is a natural extension to the canonical LALR(1) construction.

KEY WORDS :

Parsing, lookahead LR, noncanonical, two-stacks automata, context-free grammars

Noncanonical LALR(1) Parsing

Sylvain Schmitz
Laboratoire I3S
Université de Nice - Sophia Antipolis, France
schmitz@i3s.unice.fr

Abstract

This paper addresses the longstanding problem of the recognition limitations of classical LALR(1) parser generators by proposing the usage of noncanonical parsers. To this end, we present a definition of noncanonical LALR(1) parsers. The class of grammars accepted by noncanonical LALR(1) parsers is a proper superclass of the NSLR(1) and LALR(1) grammar classes. The recognized languages include some nondeterministic languages. The proposed deterministic parsers retain many of the qualities of canonical LALR(1) parsers: they are unambiguous, easy to construct, and running in linear time. We argue that they could provide the basis for a range of powerful noncanonical parsers.

Categories and Subject Descriptors D.3.1 [*Programming Languages*]: Formal Definitions and Theory—Syntax; D.3.4 [*Programming Languages*]: Processors—Parsing; F.4.2 [*Mathematical Logic and Formal Languages*]: Grammars and Other Rewriting Systems—Parsing

1. Introduction

Testimonies abound on the shortcomings of classical parser generators like YACC [12] and bison [7]. The problem lies in the large expressivity gap between what can be specified using the context-free grammar they are fed with, and what can actually be parsed by the lookahead LR(1) (LALR(1)) automaton they produce. Transforming a grammar until its LALR(1) parser becomes deterministic is arduous, and can obfuscate the attached semantics [13]; moreover, some languages are simply not deterministic.

The expressivity gap does not exist when general parsers [8, 23] are used, since they can parse any string generated by any context-free grammar. But using these, parsing is no longer guaranteed to be performed in linear time. Much worse, choosing this solution is also done at the expense of the detection of ambiguities. This potentially leads to runtime problems if such ambiguities were not expected, for instance with machine languages, supposedly precise and unambiguous.

Alternatively, the expressivity gap is reduced when LALR(k) parsers with $k > 1$ symbols of lookahead [3] are used, or, even better, when LALR parsers allowing an unbounded regular lookahead [2, 17, 9] are used. These parsers accept a wider class of grammars, while not accepting any ambiguous syntax. Such properties coincide with what we aim for in this paper, so we will often use regular lookahead parsers for comparison purposes.

This paper advocates an almost forgotten way of diminishing the expressivity gap: the usage of noncanonical parsers, which are allowed to reduce non leftmost phrases. Like regular lookahead extensions [4], they can parse nondeterministic languages. They have been thoroughly investigated in [19, 20], but the only known practical method is an extension to SLR(1) parsing [21].

If we compare noncanonical parsers with regular lookahead parsers, we see that both are able to use an unbounded right context. Noncanonical parsers achieve this by reducing this unbounded context to a finite sequence of symbols, a sequence that will fit in their lookahead window. Regular lookahead parsers achieve this by using a finite state automaton to explore this unbounded context, thus acting as if they had no limit on the length of their lookahead window. The classes of grammars accepted by both methods are incomparable in general [20]. But, in practice, regular lookahead parsers seem to handle a few more useful cases.

On further comparison, noncanonical parsers could still compete with their alternatives. In contrast with the incomparability on the grammatical level, noncanonical parsers are strictly more powerful than regular lookahead parsers on the generated languages level [20]. And there is a winning argument in favor of noncanonical parsers: they can also increase the size of their lookahead window, possibly to an unbounded length [10]. This point motivates the study of noncanonical LALR(1) parsers, since NSLR(1) parsers are unfit for such extensions: their lookahead computation is not contextual.

We describe a noncanonical extension to LALR(1) parsing. We obtain its practical realization by adapting the efficient and broadly used LALR(1) computation of [5] to the noncanonical case. The additional complexity of generating a NLALR(1) parser instead of a LALR(1) or a NSLR(1) one, as well as the increase of the parser size and the overhead on parsing performances are all quite small. Therefore, the improved parsing power comes at a fairly reasonable price. Moreover, known methods for the computation of k-lookaheads or regular lookaheads for a LALR parser should be easy to adapt to the noncanonical version.

The paper is organized as follows: section 2 is a short introduction to noncanonical parsing, with an informal description, using an example, of noncanonical LALR(1) parsing and parser generation. Section 3 recalls the formal details of the canonical LALR(1) definition, which will be extended for its noncanonical counterpart in section 4. We cast this definition in the framework of [5] to derive a simple construction in section 5. In section 6, noncanonical LALR(1) parsers are compared to other parsers. A concrete example of the use of a noncanonical LALR(1) parser is given in section 7. Proofs are omitted due to space constraints, but they are presented in the appendices of a research report [16], along with the description of some alternative definitions for noncanonical LALR-based parsers.

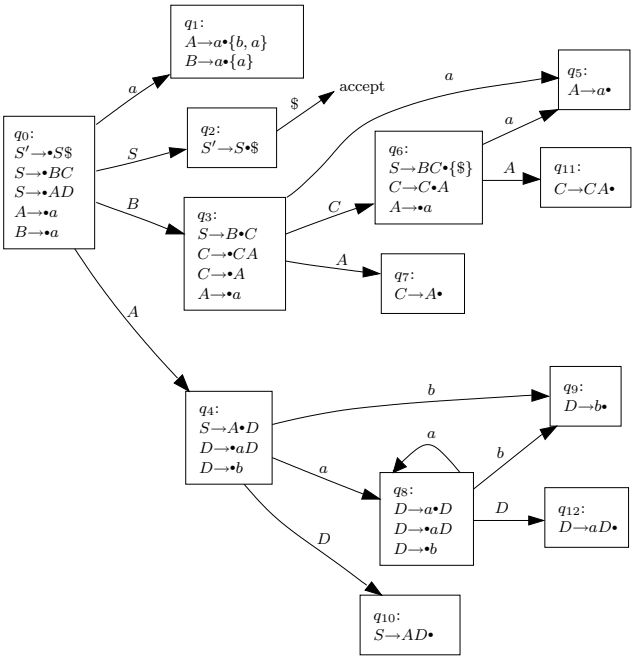

Figure 1. The LALR(1) automaton for $\mathcal{G}_1$.

Terminology and Notations The basic terminology, definitions, and notational conventions used in this paper are mostly consistent with those of [1, 18]. We recall the following:

- (V, P) is a *rewriting system* where finite set V denotes the *vocabulary* and P the set of *productions* or *rules* defined over $V^* \times V^*$; its elements are denoted as $\alpha{\to}\beta$ in generative systems (grammars) or $\beta \vdash \alpha$ in recognitive systems (automata);
- $\Rightarrow$ is the *derivation* relation of generative systems over $V^* \times V^*$, defined for δ and σ in V^* and $\alpha{\to}\beta$ in P by $\delta\alpha\sigma \overset{\alpha\to\beta}{\Rightarrow} \delta\beta\sigma$; we omit the superscript when unnecessary; $\models$ is defined similarly for recognitive systems;
- $\mathcal{G} = \langle N, T, P, S\rangle$ is a context-free grammar (CFG), where N, T and S are the set of *nonterminal* symbols, the set of *terminal* symbols, and the *start* symbol in N; it is a generative rewriting system with $V = N \cup T$ and P restricted to $N \times V^*$;
- the *language* generated by the CFG $\mathcal{G}$ is $\mathcal{L}_\mathcal{G} = \{x \mid S \Rightarrow^* x\}$;
- $\underset{\text{rm}}{\Rightarrow}$ denotes a *rightmost derivation* and $\underset{\text{lm}}{\Rightarrow}$ a *leftmost derivation*, using respectively the rightmost or leftmost applicable rewriting rule;
- our grammars are reduced and *augmented* with a rule $S'{\to}S\$$ where \$ stands for the end of the input, S' for the new start symbol, $T' = T \cup \{\$\}$, $N' = N \cup \{S'\}$ and $V' = T' \cup N'$;
- $A, B, C, \ldots$ denote nonterminals in N; $a, b, c, \ldots$ denote terminals in T; $u, v, w, \ldots$ denote strings in T^*; X, Y, Z denote symbols in V; $\alpha, \beta, \gamma, \ldots$ denote strings in V^*; q denotes a canonical LR(0) states; s denotes a noncanonical state;
- ε is the empty string or empty sequence;
- $|\alpha|$ is the *length* of string α; $|\varepsilon| = 0$;
- $k{:}\alpha =$ is the *prefix* of length k of string α.

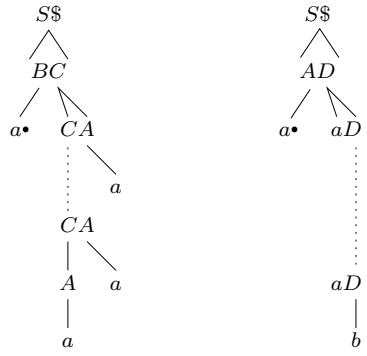

Figure 2. The derivation trees with the dot corresponding to the reduce/reduce conflict position in state q_1.

2. Noncanonical Parsing

A bottom-up parser reverses the derivation steps which lead to the terminal string it parses. The reversal of a derivation $\delta A\sigma{\Rightarrow}\delta\alpha\sigma$ is the reduction of the *phrase* α in the sentential form $\delta\alpha\sigma$ to the nonterminal A. For most bottom-up parsers, including LALR parsers, these derivations are rightmost, and therefore the reduced phrase is the leftmost one, called the *handle* of the sentential form.

To quote [1], noncanonical parsers allow the reduction of phrases which may not be handles. The derivation order can be different from the rightmost one, and adapted to the needs of the parser. A noncanonical parser is able to suspend a reduction decision where its canonical counterpart would not be deterministic, explore the remaining input, perform some reductions, resume to the conflict point and use nonterminals—resulting from the reduction of a possibly unbounded amount of input—in its lookahead window to infer its parsing decisions. We will clarify this behavior with an example.

2.1 Parsing Example

Consider for instance the grammar with rules

$$S{\to}BC\,|\,AD,\ A{\to}a,\ B{\to}a,\ C{\to}CA\,|\,A,\ D{\to}aD\,|\,b, \quad (\mathcal{G}_1)$$

generating the language

$$\mathcal{L}_{\mathcal{G}_1} \ = \ aa^+ \ \mid \ aa^*b,$$

and whose LALR(1) machine is shown in Figure 1.

The state q_1 in this automaton is *inadequate*: the parser is unable to decide between the reduction according to the rule $A{\to}a$ and the reduction according to the rule $B{\to}a$ when a appears in the lookahead window. This first a of the input is the handle. We see on the derivation trees of Figure 2 that, in order to choose between the two reductions for the handle, the parser has to know whether there is a b symbol at the very end of the input, and choose the reduction to A, or not, and choose the reduction to B.

This need for an unbounded lookahead makes $\mathcal{G}_1$ non-LR(k), for any value of k we might choose. A parser using a regular lookahead would solve the conflict by associating the distinct regular lookaheads a^*b and $a^+\$$ with the reductions to A and B respectively.

However, we notice on the derivation trees of Figure 2 that a single lookahead symbol is sufficient in order to choose the parsing action: if the parser is able to explore the context on the right of this point, and to reduce some other phrases, then, at some point, it will reduce the entire right context to a D or a C. When coming back to the conflict point, it will see a D or a C in the lookahead window.

parsing stack	input stack	actions
q_0	$aaa\$$	shift
q_0q_1	$aa\$$	shift

The automaton reaches the inadequate state q_1 with lookahead a. The decision of reducing to A or B can be restated as the decision of reducing the right context to D or C. In order to be able to perform the latter decision, we shift a and reach a state s_1 where we now expect a^*b and $a^*\$$ as remaining input for the reductions of a to A and B. We are pretty much in the same situation as before: s_1 is also inadequate. But we know that in front of b or \$ a decision can be made:

$q_0q_1s_1$	$a\$$	shift

There is a new conflict between the reduction of a to A and the shift of a to a position $D\rightarrow a\bullet D$. We also shift this a. The expected right contexts are still a^*b and $a^*\$$, so the shift brings us again to s_1:

$q_0q_1s_1s_1$	$\$$	reduce using $A\rightarrow a$

The decision is made in front of \$. We reduce the a represented by s_1 on top of the parsing stack, and push the reduced symbol A *on top* of the input stack:

$q_0q_1s_1$	$A\$$	reduce using $A\rightarrow a$

Using this new lookahead symbol, the parser is able to decide another reduction to A:

q_0q_1	$AA\$$	reduce using $B\rightarrow a$

We are now back in state q_1. Clearly, there is no need to wait until we see a completely reduced symbol C in the lookahead window: A is already a symbol specific to the reduction to B:

q_0	$BAA\$$	shift
q_0q_3	$AA\$$	shift
$q_0q_3q_7$	$A\$$	reduce using $C\rightarrow A$
q_0q_3	$CA\$$	shift
$q_0q_3q_6$	$A\$$	shift
$q_0q_3q_6q_{11}$	$\$$	reduce using $C\rightarrow CA$
q_0q_3	$C\$$	shift
$q_0q_3q_6$	$\$$	reduce using $S\rightarrow BC$
q_0	$S\$$	shift
q_0q_2	$\$$	accept

Table 1. The parse of the string aaa by the NLALR(1) parser for $\mathcal{G}_1$.

The grammar clearly allows an A or a B in front of an a in state q_1 only if this a was itself produced by a D or a C respectively.

Table 1 presents a noncanonical parse for a string in $\mathcal{L}_{\mathcal{G}_1}$ using this principle. The noncanonical machine is not very different from the canonical one, except that

- it uses two stacks instead of one; the additional stack is called the *input stack* and also serves as an input for the parser, whereas the other stack is called the *parsing stack*;
- reductions push the reduced nonterminal on top of the input stack; there is no goto operation *per se*: the nonterminal on top of the input stack either allows a parsing decision which had been delayed, or is simply shifted as in a goto.

We will now see how to transform and extend the canonical LALR(1) parser of Figure 1 to perform these parsing steps.

2.2 Construction Principles

The LALR construction relies heavily on the LR(0) automaton. This automaton provides a nice explanation for LALR lookahead sets: the symbols in the lookahead set for some reduction are the symbols expected next by the LR(0) parser, should it really perform this reduction.

Let us compute the lookahead set for the reduction using $A\rightarrow a$ in state q_1. Should the LR(0) parser decide the reduction to A, it would then pop q_1 from the parsing stack and thus be in state q_0. And since it would have reduced to an A, it would push q_4 on the parsing stack. We read directly on Figure 1 that three symbols are acceptable in q_4: D, a and b. Similarly, the reduction using $B\rightarrow a$ in q_1 has $\{C, A, a\}$ for lookahead set, read directly from state q_3.

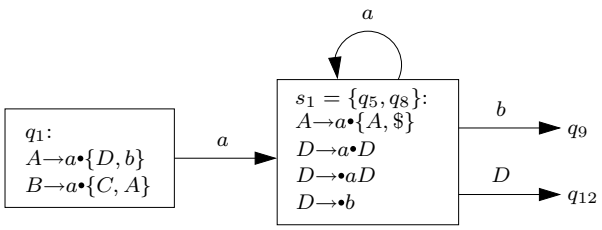

Figure 3. State q_1 extended for noncanonical parsing.

The intersection of the lookahead sets for the reductions of a in q_1 is not empty: symbol a appears in both, which means a conflict. Luckily enough, a is not a *totally reduced symbol*: D and C are reduced symbols, read from kernel items in q_4 and q_3. Symbol a could be reduced, and later we might see a symbol on which we can make a decision instead. As shown in the parsing example, we shift the lookahead symbol a in order to reduce it and solve the conflict later. All the other symbols in the computed lookaheads allow to make a decision, so we leave them in the lookaheads sets, but we remove a from both sets.

Shifting a puts us in a situation equivalent to following transitions on a from both q_3 and q_4, the two states found earlier by simulating the reductions. The noncanonical generation simulates both reductions. We create a noncanonical transition from q_1 on a to a noncanonical state s_1,

$$s_1 = \{q_5, q_8\},$$

which will behave as the union of states q_5 and q_8 accessed on a from q_3 and q_4. State s_1 will thus allow a reduction using $A\rightarrow a$ inherited from q_5, and the shifts of a, b and D inherited from q_8. We therefore need to compute the lookaheads for reduction using $A\rightarrow a$ in q_5 in hope of being able to distinguish it from the shifts. Using again the LR(0) simulation technique, we see on Figure 1 that this reduction would lead us to either q_7, if q_3 was on the stack just under q_5, or to q_{11} if q_6 was on the stack. In both cases, the LR(0) automaton would perform a reduction to C that would lead next to q_6. At this point, the LR(0) automaton expects either the end of file symbol \$, should a reduction to S occur, or an A or an a. The complete lookahead set for the reduction using $A\rightarrow a$ in q_8 is thus $\{A, a, \$\}$.

The new state s_1 is also inadequate: with an a in the lookahead window, we cannot choose between the shift of a and the reduction using $A\rightarrow a$. As before, we create a new transition on a from s_1 to a noncanonical state

$$s'_1 = \{q_5, q_8\}.$$

State q_5 is the state accessed on a from q_6. State q_8 is the state accessed from q_8 if we simulate a shift of symbol a.

State s'_1 is the same as state s_1, and we merge them. The noncanonical computation is now finished. Figure 3 sums up how state q_1 has been transformed and extended. Note that in reality, we just use the set $\{q_5, q_8\}$ in a noncanonical LALR(1) automaton; items represented in Figure 3 are there to ease understanding.

We will soon provide precise definitions for noncanonical LALR(1) parsers, but we first recall some theory of LR(0) and LALR(1) parsing.

3. LALR(1) Parsers

LALR parsers were introduced in [6] as practical parsers for deterministic languages. Rather than building an exponential number of LR(k) states, LALR(k) parsers add lookahead sets to the actions of the small LR(0) parser. These lookahead sets are the unions of all the lookahead sets a LR(k) parser would expect, given the input prefix processed so far.

We recall some important definitions and results on LR(0) and LALR(1) parsers. The reader will find a much more detailed exposition in [18].

3.1 Valid Items and Prefixes

A dotted production $A \rightarrow \alpha \bullet \beta$ of $\mathcal{G}$ is a *valid LR(0) item* for string γ in V'^* if

$$S' \underset{\text{rm}}{\Rightarrow}^* \delta A z \underset{\text{rm}}{\Rightarrow} \delta\alpha\beta z = \gamma\beta z. \tag{1}$$

If such a derivation holds in $\mathcal{G}$, then γ in V'^* is a *valid prefix*.

The set of valid items for a given string γ in V'^* is denoted by Valid(γ). Two strings δ and γ are equivalent if and only if they have the same valid items.

The valid item sets are obtained through the following computations:

$$\text{Kernel}(\varepsilon) = \{S' \rightarrow \bullet S\$\}, \tag{2}$$

$$\text{Kernel}(\gamma X) = \{A \rightarrow \alpha X \bullet \beta \mid A \rightarrow \alpha \bullet X \beta \in \text{Valid}(\gamma)\}, \tag{3}$$

$$\text{Valid}(\gamma) = \text{Kernel}(\gamma) \cup \{B \rightarrow \bullet\omega \mid A \rightarrow \alpha \bullet B\beta \in \text{Valid}(\gamma)\}. \tag{4}$$

3.2 LR(0) States

LR automata are pushdown automata that use equivalence classes on valid prefixes as their stack alphabet Q. We therefore denote explicitly states of a LR parser as $q = [\delta]$, where δ is some valid prefix in q the state reached upon reading this prefix. Each state of form $[\delta X]$ has a unique *accessing symbol* X. For instance, in the automaton of Figure 1, state q_2 is the equivalence class $\{S\}$, while state q_8 is the equivalence class described by the regular language Aa^*a.

A pair $([\delta], X)$ in $Q \times V'$ is a *transition* if and only if δX is a valid prefix. If this is the case, then $[\delta X]$ is the state accessed upon reading δX, thus the notation $[\delta X]$ also implies a transition from $[\delta]$ on X, and $[\delta\alpha]$ a *path* on α. Transition (q, X) is a *terminal transition* if X is in T or a *nonterminal transition* if X is in N.

3.3 LR(0) Automata

Let $M = (Q \cup T \cup \{\$, \|\}, R)$ be a rewriting system where \$ and $\|$ (the end marker and the stack top, respectively) are not found in Q nor in T (the set of states and the input alphabet, respectively). A *configuration* of M is a string of the form

$$[\varepsilon][X_1] \dots [X_1 \dots X_n] \| x\$ \tag{5}$$

where $X_1 \dots X_n$ is a string in V^* and x a string in T^*.

We say that M is a *LR(0) automaton* for grammar $\mathcal{G}$ if its initial configuration is $[\varepsilon]\|w\$$ with w the input string in T^*, its final configuration is $[\varepsilon][S]\|\$$, and if each rewriting rule in R is of one of the forms

- *shift* a in state $[\delta]$

$$[\delta]\|a \vdash_{\text{shift}} [\delta][\delta a]\|, \tag{6}$$

  defined if there is an item of form $A \rightarrow \alpha \bullet a \beta$ in Valid(δ), or

- *reduce* by rule $A \rightarrow X_1 \dots X_n$ of P in state $[\delta X_1 \dots X_n]$

$$[\delta X_1] \dots [\delta X_1 \dots X_n]\| \vdash_{A \rightarrow X_1 \dots X_n} [\delta A]\|, \tag{7}$$

  defined if $A \rightarrow X_1 \dots X_n \bullet$ is in Valid($\delta X_1 \dots X_n$).

A *bottom-up parser* (M, τ) is a transducer based on automaton M with a homomorphism τ from R^* to P^* defined for shift and reduce actions by

$$\tau(\vdash_{\text{shift}}) = \varepsilon, \text{ and} \tag{8}$$

$$\tau(\vdash_{A \rightarrow \alpha}) = A \rightarrow \alpha. \tag{9}$$

3.4 LALR(1) Automata

The *LALR(1) lookahead set* of a reduction $\vdash_{A \rightarrow \alpha}$ in state q is

$$\text{LA}(q, A \rightarrow \alpha) = \{1{:}z \mid S' \underset{\text{rm}}{\Rightarrow}^* \delta A z \text{ and } q = [\delta\alpha]\}. \tag{10}$$

Rewriting system M is a *LALR(1) automaton* for grammar $\mathcal{G}$ if it is a LR(0) automaton with another possible form of rules

- *reduce* by rule $A \rightarrow X_1 \dots X_n$ of P in state $[\delta X_1 \dots X_n]$ with lookahead a

$$[\delta X_1] \dots [\delta X_1 \dots X_n]\|a \vdash_{A \rightarrow X_1 \dots X_n} [\delta A]\|a, \tag{11}$$

  defined if $A \rightarrow X_1 \dots X_n \bullet$ is in Valid($\delta X_1 \dots X_n$) and lookahead a is in LA($[\delta X_1 \dots X_n], A \rightarrow X_1 \dots X_n$).

We have now enough elements to present a formal definition for noncanonical LALR(1) parsers.

4. NLALR(1) Parsers

As seen in section 2.2, there is a number of differences between the LALR(1) definition and the NLALR(1) one. The most visible one is that we accept nonterminals in our lookahead sets. We also want to know which lookahead symbols are totally reduced. Finally, we are adding new states, which are sets of LR(0) states. Therefore, the objects in most of our computations will be LR(0) states.

4.1 Valid Covers

We have recalled in the previous section that LR(0) states can be viewed as collections of valid prefixes. A similar definition for NLALR(1) states would be nice. However, due to the suspended parsing actions, the language of all prefixes accepted by a noncanonical parser is no longer a regular language. This means the parser will only have a regular approximation of the exact parsing stack language. The noncanonical states, being sets of LR(0) states, and thus being sets of equivalence classes on valid prefixes using the LR(0) equivalence, provide this approximation. We therefore define valid covers as valid prefixes covering the parsing stack language.

DEFINITION 1. *String γ is a* valid cover *in $\mathcal{G}$ for string δ if and only if γ is a valid prefix and $\gamma \Rightarrow^* \delta$. We write $\hat{\delta}$ to denote some cover of δ and Cover(δ) to denote the set of all valid covers for string δ.*

An immediate property of valid covers is that, given a prefix δ of some sentential form $\delta\omega$, it is a simple matter of permutating the sequence of productions used in derivation $S' \Rightarrow^* \delta\omega$ to find a valid cover $\hat{\delta}$ such that $S' \underset{\text{rm}}{\Rightarrow}^* \hat{\delta}\sigma$ where $\omega \Rightarrow^* \sigma$. On the other hand, if δ is not a prefix of some sentential form of $\mathcal{G}$, then it does not have any valid cover. Also note that valid covers of valid prefixes will be of special interest: in case of a conflict, we need to find how to keep on exploring the right context.

Remember for instance configuration $q_0 q_1 \| aa\$$ from Table 1. This configuration leads to pushing state $s_1 = \{q_5, q_8\}$, where both valid prefixes $(B|BC)a$ and Aa^*a of q_5 and q_8 are valid covers for the actual parsing stack prefix aa. Thus in s_1 we cover the parsing stack prefix by $(B \mid BC \mid Aa^*)a$.

4.2 Noncanonical Lookaheads

Noncanonical lookaheads are lookaheads symbols in V', as opposed to being taken from T', as done in canonical operation. Adapting the computation of the LALR(1) lookahead sets is simple, but a few points deserve some explanations.

First of all, all noncanonical lookahead symbols have to be *non null*, *i.e.* X is non null if and only if $X \Rightarrow^* ax$. Indeed, null symbols, which only derive the empty string, do not provide any additional right context information—worse, they can hide it, as explained in [19]. If we consider that we always perform a reduction at the earliest parsing stage possible, then they will never appear in a lookahead window.

4.2.1 Totally Reduced Lookaheads

Totally reduced lookaheads form a subset of the noncanonical lookahead set such that none of its elements can be further reduced. A conflict with a totally reduced symbol as lookahead cannot be solved by a noncanonical exploration of the right context, since there is no hope of ever reducing it any further.

We define here totally reduced lookaheads as non null symbols which can follow the right part of the offending rule in a leftmost derivation.

DEFINITION 2. *The set of* totally reduced lookaheads *for a reduction using $A \to \alpha$ in LR(0) state q is defined by*

$$\mathit{RLA}(q, A \to \alpha) = \{X \mid S' \underset{\text{lm}}{\Rightarrow}^* zA\gamma X\omega, \gamma \Rightarrow^* \varepsilon, X \Rightarrow^* ax, \text{ and } q = [\hat{z}\alpha]\}. \tag{12}$$

4.2.2 Derived Lookaheads

The derived lookahead symbols are simply defined by extending (10) to the set of all non null symbols in V.

DEFINITION 3. *The set of* derived lookaheads *for a reduction using $A \to \alpha$ in LR(0) state q is defined by*

$$\mathit{DLA}(q, A \to \alpha) = \{X \mid S' \Rightarrow^* \delta AX\omega, X \Rightarrow^* ax, \text{ and } q = [\hat{\delta}\alpha]\}. \tag{13}$$

We obviously have

$$\mathrm{LA}(q, A \to \alpha) = \mathrm{DLA}(q, A \to \alpha) \cap T'. \tag{14}$$

4.2.3 Conflicting Lookahead Symbols

Last, we need to compute which lookahead symbols would make the state inadequate in canonical parsing. A noncanonical exploration of the right context is required for these symbols. They appear in the derived lookahead sets of several reductions and/or are transition labels. However, the totally reduced lookaheads of a reduction are not part of this lookahead set, for if they are involved in a conflict, then there is no hope of being able to solve it.

DEFINITION 4. Conflicts lookahead set *for a reduction using $A \to \alpha$ in set s of LR(0) states is defined as*

$$\mathit{CLA}(s, A \to \alpha) = \{X \in \mathit{DLA}(q, A \to \alpha) \mid q \in s, ((q, X) \text{ or } \exists p \in s, X \in \mathit{DLA}(p, B \to \beta)) \text{ and } X \notin \mathit{RLA}(q, A \to \alpha)\}. \tag{15}$$

We then define the noncanonical lookahead set *for a reduction using $A \to \alpha$ in set s of LR(0) states as*

$$\mathit{NLA}(s, A \to \alpha) = \Big(\bigcup_{q \in s} \mathit{DLA}(q, A \to \alpha)\Big) - \mathit{CLA}(s, A \to \alpha). \tag{16}$$

We illustrate these definitions by computing the lookahead sets for the reduction using $A \to a$ in state $s_1 = \{q_5, q_8\}$ as in section 2.2:

$$\begin{aligned} &\mathrm{RLA}(q_5, A \to a) = \{A, \$\}, \\ &\mathrm{DLA}(q_5, A \to a) = \{A, a, \$\}, \\ &\mathrm{CLA}(s_1, A \to a) = \{a\}, \text{ and} \\ &\mathrm{NLA}(s_1, A \to a) = \{A, \$\}. \end{aligned}$$

4.3 Noncanonical States

We saw in section 2.2 that states in the NLALR(1) automaton were in fact sets of LR(0) states. We denote by $[\![\delta]\!]$ the noncanonical state accessed upon reading string δ in V'^*.

DEFINITION 5. Noncanonical state $[\![\delta]\!]$ *is the set of LR(0) states defined by*

$$[\![\varepsilon]\!] = \{[\varepsilon]\} \text{ and} \tag{17}$$

$$[\![\delta X]\!] = \{\widehat{[\hat{\gamma}AX]} \mid X \in \mathit{CLA}([\![\delta]\!], A \to \alpha) \text{ and } [\hat{\gamma}\alpha] \in [\![\delta]\!]\} \cup \{[\varphi X] \mid [\varphi] \in [\![\delta]\!]\}. \tag{18}$$

Noncanonical transition *from $[\![\delta]\!]$ to $[\![\delta X]\!]$ on symbol X, denoted by $([\![\delta]\!], X)$, exists if and only if $[\![\delta X]\!] \neq \emptyset$.*

Reduction $([\![\delta]\!], A \to \alpha)$ exists if and only if there exists a reduction $(q, A \to \alpha)$ and q is in $[\![\delta]\!]$.

Note that these definitions remain valid for plain LALR(1) states since in absence of a conflict, a noncanonical state is a singleton set containing the corresponding LR(0) state.

A simple induction on the length of δ shows that the LR(0) states considered in the noncanonical state $[\![\delta]\!]$ provide a valid cover for any accessing string of the noncanonical state. It basically means that the actions decided in a given noncanonical state make sense at least for a cover of the real sentential form prefix that is read.

Note that the approximations done when covering the actual sentential form prefix are made on top of the previous approximations: with each new conflict, we need to find a new set of LR(0) states covering the parsing stack contents. This stacking is made obvious in the above definition when we write $\widehat{\hat{\gamma}AX}$. Among other things, it means that NLALR(1) parsers are not prefix valid, but prefix cover valid. In other words, we cannot guarantee that the input read so far is a valid prefix of some sentential form in the language, but we can guarantee that there exists at least one valid prefix that covers the input read so far.

In this paper, we use the LR(0) automaton to approximate the prefix read so far, but we could use more powerful methods—but it would not really be in the spirit of LALR parsing any longer; see for instance the alternative NLALR(1) definition given in Appendix A.3 of [16]. Avoiding this stacking of approximations one on top of the previous one seems likely to yield an undecidable method; see for instance FSPA(k) parsing in [20].

4.4 NLALR(1) Automata

Here we formalize noncanonical LALR(1) parsing machines. As seen in section 2.1, they are a special case of two-stack pushdown automata (2PDA).

DEFINITION 6. *Let $M = (Q \cup V \cup \{\$, \|\}, R)$ be a rewriting system where $\$$ and $\|$ (the end marker and the stacks separation marker, respectively) are not found in Q nor in T (the parsing stack alphabet and the input stack alphabet, respectively).*

A configuration *of M is a string of the form*

$$[\![\varepsilon]\!][\![X_1]\!] \dots [\![X_1 \dots X_n]\!] \| \omega\$ \tag{19}$$

where $X_1 \dots X_n$ is a string in V^* and ω a string in V^*.

We say that M is a NLALR(1) automaton *for grammar $\mathcal{G}$ if its initial configuration is $[\![\varepsilon]\!] \| w\$$ with w the input string in T^*, its*

final configuration is $[\![\varepsilon]\!][\![S]\!]\|\$$*, and if each rewriting rule in* R *is of the form*

- shift X *in state* $[\![\delta]\!]$

$$[\![\delta]\!]\|X \vdash_{shift} [\![\delta]\!][\![\delta X]\!]\|, \tag{20}$$

defined if there is a transition $([\![\delta]\!], X)$*, or*

- reduce *by rule* $A \to X_1 \dots X_n$ *of* P *in state* $[\![\delta X_1 \dots X_n]\!]$

$$[\![\delta X_1]\!] \dots [\![\delta X_1 \dots X_n]\!]\| \vdash_{A \to X_1 \dots X_n} \|A, \tag{21}$$

defined if $A \to X_1 \dots X_n$ *is a reduction in* $[\![\delta X_1 \dots X_n]\!]$*, or*

- reduce *by rule* $A \to X_1 \dots X_n$ *of* P *in state* $[\![\delta X_1 \dots X_n]\!]$ *with lookahead* X

$$[\![\delta X_1]\!] \dots [\![\delta X_1 \dots X_n]\!]\|X \vdash_{A \to X_1 \dots X_n} \|AX, \tag{22}$$

defined if $A \to X_1 \dots X_n$ *is a reduction in* $[\![\delta X_1 \dots X_n]\!]$ *and lookahead* X *is in NLA*$([\![\delta X_1 \dots X_n]\!], A \to X_1 \dots X_n)$*.*

We see in this definition that NLALR(1) automata are able to backtrack by a limited amount, corresponding to the length of their window, at reduction time only. We know from [19] that noncanonical parsers using a bounded lookahead window operate in linear time; the following theorem precisely shows that the total number of rules involved in the parsing of an input string is linear in respect with the number of reductions performed, which itself is linear in respect with the input string length.

THEOREM 1. *Let* $\mathcal{G}$ *be a grammar and* (M, τ) *its NLALR(1) bottom-up parser. Then* $\mathcal{L}_M \subseteq \mathcal{L}_\mathcal{G}$ *and if* π *is a parse of* w *in* M*, then the number of parsing steps* $|\pi|$ *is related to the number* $|\tau(\pi)|$ *of derivations producing* w *in* $\mathcal{G}$ *and to the length* $|w|$ *of* w *by*

$$|\pi| = 2|\tau(\pi)| + |w|.$$

Proof. Proof is given in Appendix B.1 of [16]. □

Since all the conflict lookahead symbols are removed from the noncanonical lookahead sets NLA, the only possibility for the noncanonical automaton to be nondeterministic would be to have a totally reduced symbol causing a conflict. A context-free grammar $\mathcal{G}$ is NLALR(1) if and only if its NLALR(1) automaton is deterministic, and thus if and only if no totally reduced symbol can cause a conflict.

The following rules illustrate the previous definition on state s_1 of the NLALR(1) automaton for $\mathcal{G}_1$:

$$\begin{array}{llr} s_1\|a & \vdash_{\text{shift}} & s_1 s_1\|, \\ s_1\|b & \vdash_{\text{shift}} & s_1\{q_9\}\|, \\ s_1\|D & \vdash_{\text{shift}} & s_1\{q_{12}\}\|, \\ s_1\|A & \vdash_{A \to a} & \|AA, \text{ and} \\ s_1\|\$ & \vdash_{A \to a} & \|A\$. \end{array}$$

4.5 Practical Construction Steps

We present here a more informal construction, with the main steps leading to the construction of a NLALR(1) parser.

1. Construct the LR(0) automaton for grammar $\mathcal{G}$.
2. Associate a noncanonical state $s = \{q\}$ with each LR(0) state q.
3. Iterate while there exists an inadequate[1] state s:
   (a) if it has not been done before, compute the RLA and DLA lookahead sets for the reductions involved in the conflict; save their values for the reduction and LR(0) state involved;
   (b) compute the CLA and NLA lookahead sets for s;
   (c) set the lookaheads to NLA for the reduction actions in s;
   (d)
   - if the NLA lookahead sets leave the state inadequate, meaning there is a conflict on the totally reduced lookaheads, then report the conflict and use a conflict resolution policy or terminate with an error;
   - if CLA is not empty, create transitions on its symbols and create new states if no fusion occurs. New states get new transition and reduction sets. If these new states result from shift/reduce conflicts, the transitions from s on the conflicting lookahead symbol now lead to the new states.

This process always terminates since there is a bounded number of LR(0) states and thus a bounded number of noncanonical states.

There is room for optimization here. For instance, using the described procedure, some states might have to be discarded at the end of the construction if they cannot be reached from the initial state. Also, different states may have the same sets of parsing actions, which could be avoided by compressing the parsing tables. Note however that the noncanonical lookahead sets RLA and DLA of a reduction in a given LR(0) state are only computed once.

Let us conclude this section with a few words on the size of the generated parsers. Since NLALR(1) states are sets of LR(0) states, we find an exponential function of the size of the LR(0) automaton as an upper bound on the size of the NLALR(1) automaton. This bound seems however pretty irrelevant in practice. The NLALR(1) parser needs to create a new state for each symbol in conflict, which does not happen so often. All the grammars we studied so far created transitions to canonical states very quickly afterwards. In [21], the author gave experimental results for NSLR(1) parsers showing that the increase in size was negligible in practice.

5. Computing the Lookaheads and Covers

Up to this point, we have avoided the question of how to compute the RLA and DLA lookahead sets or the valid covers. These issues are related to the simulation of the LR(0) parser behavior as presented in Section 2.2. This simulation might seem complex at times; it was for instance more lengthy with the reduction using $A \to a$ in q_5 than with the reductions in q_1. It is even more complex when ε-rules are involved.

Several ways of performing this simulation have been designed in the past, and we will follow the LALR construction of [5], since it is arguably the most widespread one, both in education and in actual implementations. We thus expect our noncanonical LALR(1) construction to be simple to understand and to implement. Note that we are in no way restricted to this particular construction, and that our definition of a noncanonical LALR(1) parser could easily be implemented in a different way.

5.1 Computing the LALR(1) Lookahead Sets

The LALR(1) lookahead sets that are defined in Equation (10) can be efficiently computed using the following definitions, where **lookback** is a relation between reductions and nonterminal LR(0) transitions, **includes** and **reads** are relations between nonterminal LR(0) transitions, and DR—standing for *directly reads*—is a function from nonterminal LR(0) transitions to sets of lookahead symbols.

$$([\delta\alpha], A \to \alpha) \textbf{ lookback } ([\delta], A), \tag{23}$$

$$([\delta\beta], A) \textbf{ includes } ([\delta], B) \text{ iff } B \to \beta A \gamma \text{ and } \gamma \Rightarrow^* \varepsilon, \tag{24}$$

$$([\delta], A) \textbf{ reads } ([\delta A], C) \text{ iff } ([\delta A], C) \text{ and } C \Rightarrow^* \varepsilon, \tag{25}$$

$$\text{DR}([\delta], A) = \{a \mid ([\delta A], a)\}. \tag{26}$$

[1] We mean here inadequate in the LR(0) sense, thus no lookaheads need to be computed yet.

Using the above definitions, we can rewrite (10) as

$$\mathrm{LA}(q, A{\to}\alpha) = \bigcup_{(q,A\to\alpha)\ \textbf{lookback}\circ\textbf{includes}^*\circ\textbf{reads}^*(r,C)} \mathrm{DR}(r, C). \tag{27}$$

For instance, considering the reduction using $A{\to}a$ in q_5 as we did in Section 2.2, since

$$(q_5, A{\to}a)\ \textbf{lookback}\ (q_3, A)\ \textbf{includes}\ (q_3, C)\ \textbf{includes}\ (q_0, S)$$

and since

$$\mathrm{DR}(q_0, S) = \{\$\},$$

we find again that the end of file \$ belongs to the LALR(1) lookahead set for this reduction.

Two reasons make this computation for LALR(1) lookahead sets extremely efficient:

1. it can entirely be performed on the LR(0) automaton, and
2. the **includes** and **reads** transitive closures can be done using a fast closure algorithm inspired by [22].

5.2 Computing the NLALR(1) Lookahead Sets

Since we have a very efficient way of computing the canonical LALR(1) lookahead sets, why not try to use it for the noncanonical ones?

THEOREM 2.

$$\begin{aligned} \mathit{RLA}(q, A{\to}\alpha) = \{X \mid\ & X{\Rightarrow}^* ax, \psi{\Rightarrow}^*\varepsilon, \\ & C{\Rightarrow}\rho B{\bullet}\psi X\sigma \in \mathit{Kernel}(\delta\rho B)\ \textit{and} \\ & (q, A{\to}\alpha)\ \textbf{\textit{lookback}}\ \circ\ \textbf{\textit{includes}}^*([\delta\rho], B)\}. \end{aligned}$$

Proof. Proof is given in Appendix B.2 of [16]. □

This theorem is consistent with the description of section 2.2, where we said that C was a totally reduced lookahead for reduction $B{\to}a$ in q_1: $(q_1, B{\to}a)$ **lookback** (q_0, B), and item $S{\to}B{\bullet}C$ is in the kernel of state q_3 accessed by (q_0, B).

THEOREM 3. *Let us extend the* directly reads *function of (26) to*

$$\mathit{DR}([\delta], A) = \{X \mid ([\delta A], X)\ \textit{and}\ X{\Rightarrow}^* ax\},$$

then

$$\mathit{DLA}(q, A{\to}\alpha) = \bigcup_{(q,A\to\alpha)\ \textbf{\textit{lookback}}\circ\textbf{\textit{includes}}^*\circ\textbf{\textit{reads}}^*(r,C)} \mathit{DR}(r, C).$$

Proof. Proof is given in Appendix B.3 of [16]. □

We are still consistent with the description of section 2.2 since, using this new definition of the DR function, $\mathrm{DR}(q_0, B)$ is $\{a, C, A\}$.

5.3 Finding the Valid Covers

To find the valid covers that approximate a sentential form prefix using the LR(0) automaton and to find the LALR lookahead sets wind up being very similar operations. As presented in Section 2.2, both can be viewed as simulating the LR(0) parser behavior. The following theorem formalizes this resemblance.

THEOREM 4. *Let α be a phrase such that $S'{\Rightarrow}^*\delta AX\omega{\Rightarrow}\delta\alpha X\omega$ with $q = [\hat{\delta}\alpha]$, then*

$$\begin{aligned} \mathit{Cover}(\hat{\delta}AX) = \{\gamma CX \mid\ & \\ & (q, A{\to}\alpha)\ \textbf{\textit{lookback}}\ \circ\ \textbf{\textit{includes}}^*\ \circ\ \textbf{\textit{reads}}^*([\gamma], C)\}. \end{aligned}$$

Proof. Proof is given in Appendix B.4 of [16]. □

The valid covers of a reduction can thus be found using relations (23), (24) and (25). This allows us to reuse our lookahead sets computations for the automaton construction itself, as illustrated by the following corollary.

COROLLARY 1. Noncanonical state $[\![\delta]\!]$ *is the set of LR(0) states defined by*

$$\begin{aligned} [\![\varepsilon]\!] =& \{[\varepsilon]\}\ \textit{and} \\ [\![\delta X]\!] =\ & \{[\gamma CX] \mid X \in \mathit{CLA}([\![\delta]\!], A{\to}\alpha), q \in [\![\delta]\!]\ \textit{and} \\ & (q, A{\to}\alpha)\textbf{\textit{lookback}}\circ\textbf{\textit{includes}}^*\circ\textbf{\textit{reads}}^*([\gamma], C)\} \\ & \cup \{[\varphi X] \mid [\varphi] \in [\![\delta]\!]\}. \end{aligned}$$

6. NLALR(1) Grammars and Languages

We compare here the power of NLALR(1) parsing with the power of various other parsing methods.

A *grammar class* is a set of grammars, usually the set for which a deterministic parser can be generated by some construction. We denote by $\mathcal{C}_{\mathrm{NLALR(1)}}$ the class of grammars accepted by a NLALR(1) parser; we will compare this class with other grammar classes.

6.1 Unambiguous Grammars

We have said in Section 4.4 that a grammar was NLALR(1) if and only if no totally reduced symbol could cause a conflict. This statement yields the following theorem.

THEOREM 5. *Every NLALR(1) grammar is unambiguous.*

Proof. As will be seen in the next comparison, NLALR(1) grammars are all FSPA(1) grammars, which are all unambiguous. A direct proof is sketched in Appendix B.5 of [16]. □

The class of NLALR(1) grammars is thus included in the class of all unambiguous grammars. Considering then the unambiguous and non-NLALR(1) grammar with rules

$$S{\to}aSa \,|\, bSb \,|\, \varepsilon, \tag{$\mathcal{G}_2$}$$

we get the proper inclusion

$$\mathcal{C}_{\mathrm{NLALR(1)}} \subset \mathcal{C}_{\mathrm{unambiguous}}. \tag{28}$$

6.2 FSPA(1) Grammars

Grammars that are parsable using a finite-state parsing automaton and k symbols of lookahead window have been introduced in [20]. Informally presented, a FSPA(k) grammar can be parsed noncanonically using a regular approximation of the parsing stack contents and k symbols of lookahead. This parsing method is thus the most general noncanonical parsing method using a bounded lookahead and a finite representation of the parsing stack language. However, the problem of deciding whether a given grammar is FSPA(k), even for a given k, is undecidable in general.

Noncanonical LALR(1) parsing complies with this definition; moreover, grammar

$$S{\to}A \,|\, B,\ A{\to}EEA \,|\, EE,\ B{\to}OOB \,|\, O,\ E{\to}a,\ O{\to}a \quad (\mathcal{G}_3)$$

is FSPA(1) but not NLALR(1)[2], hence the following proper inclusion for grammars without any null nonterminal

$$\mathcal{C}_{\mathrm{NLALR(1)}} \subset \mathcal{C}_{\mathrm{FSPA(1)}}. \tag{29}$$

[2] With this grammar we hit the limits of our LR(0)-based approximations. Better approximations can be designed, such as the one presented in Appendix A.3 of [16].

$<FieldDeclaration> \rightarrow <FieldModifiers><Type><VariableDeclarators>$;
$<FieldModifiers> \rightarrow <FieldModifiers><FieldModifier> \mid \varepsilon$
$<FieldModifier> \rightarrow$ `public` | `protected` | `private` | `final` | `static` | `transient` | `volatile`
$<MethodHeader> \rightarrow <MethodModifiers><ResultType><MethodDeclarator><Throws>$
$<MethodModifiers> \rightarrow <MethodModifiers><MethodModifier> \mid \varepsilon$
$<MethodModifier> \rightarrow$ `public` | `protected` | `private` | `static` | `abstract` | `final` | `native` | `synchronized`
$<ResultType> \rightarrow <Type> \mid$ `void`

Table 2. Fields and methods declarations in the Java specification. $<FieldDeclaration>$ and $<MethodHeader>$ appear in the same context, and both $<VariableDeclarators>$ and $<MethodDeclarator>$ first derive identifiers.

6.3 LALR(1) Grammars

Equality (14) and definition 5 together imply that any LALR(1) grammar has a NLALR(1) automaton with singleton noncanonical states and empty CLA lookahead sets.

Grammar $\mathcal{G}_1$ is an example of a non-LALR(1) and NLALR(1) grammar; thus the proper inclusion

$$\mathcal{C}_{\text{LALR(1)}} \subset \mathcal{C}_{\text{NLALR(1)}}. \tag{30}$$

6.4 NLSR(1) Grammars

Noncanonical SLR(1) parsers [21] are built in a very similar way to NLALR(1) grammars, though they are item-based. Their lookahead computation is however less precise as it uses the noncanonical Follow sets instead of the RLA and DLA lookahead sets. Here are the noncanonical Follow sets as corrected in [15] to exclude null symbols:

$$\text{RFollow}(A) = \{X \mid S' \underset{\text{lm}}{\Rightarrow}^* zA\gamma X\omega, \gamma \Rightarrow^* \varepsilon, X \Rightarrow^* ax\} \tag{31}$$

$$\text{DFollow}(A) = \{X \mid S' \Rightarrow^* \delta AX\omega, X \Rightarrow^* ax\}. \tag{32}$$

Given a LR(0) state q with a possible reduction using $A \rightarrow \alpha$, clearly

$$\text{RLA}(q, A \rightarrow \alpha) \subseteq \text{RFollow}(A), \tag{33}$$

and thus any NSLR(1) grammar is also NLALR(1). Grammar $\mathcal{G}_1$ is an example of a non-NSLR(1) and NLALR(1) grammar; hence the proper inclusion

$$\mathcal{C}_{\text{NSLR(1)}} \subset \mathcal{C}_{\text{NLALR(1)}}. \tag{34}$$

6.5 LL(1) and LR(1) Grammars

On one hand, Grammar $\mathcal{G}_1$ was shown to be non-LR(k) and thus non-LL(k), but is NLALR(1). On the other hand, many classical non-LALR grammars are not NLALR(1) either, since these examples rely on a faulty approximation done in the construction of the underlying LR(0) machine. For instance, the grammar with rules

$$\begin{array}{l} S \rightarrow aA \mid bB,\ A \rightarrow Cc \mid Dd,\ B \rightarrow Cd \mid Dc, \\ C \rightarrow FE,\ D \rightarrow FH,\ E \rightarrow \varepsilon,\ F \rightarrow \varepsilon,\ H \rightarrow \varepsilon \end{array} \qquad (\mathcal{G}_4)$$

is SLL(1) and thus LL(1) and LR(1) but not LALR(k) for any k nor NLALR(1).

Grammar class $\mathcal{C}_{\text{NLALR(1)}}$ is therefore incomparable with both grammar classes $\mathcal{C}_{\text{LL(1)}}$ and $\mathcal{C}_{\text{LR(1)}}$.

6.6 LRR Grammars

LR-Regular parsers [4] are able to use a regular cover for an unbounded lookahead exploration. Like with $\mathcal{C}_{\text{FSPA}(k)}$, the membership in the grammar class is an undecidable problem. However, some practical approximations have been developed [2, 17, 9].

Grammar $\mathcal{G}_4$ being LR(1), it is *a fortiori* LRR. Grammar $\mathcal{G}_5$ with rules shamelessly copied from [20]

$$S \rightarrow AC \mid BD,\ A \rightarrow a,\ B \rightarrow a,\ C \rightarrow bC \mid bD,\ D \rightarrow bDc \mid bc \quad (\mathcal{G}_5)$$

is not LRR but is NLALR(1).

Grammar class $\mathcal{C}_{\text{NLALR(1)}}$ is therefore incomparable with grammar class $\mathcal{C}_{\text{LRR}}$.

6.7 Deterministic Languages

The NLALR(1) language class is the set of all languages for which there exists a NLALR(1) grammar.

All deterministic languages can be parsed using a deterministic SLR(1) machine; since $\mathcal{C}_{\text{SLR(1)}} \subset \mathcal{C}_{\text{LALR(1)}}$, they can also be parsed by a deterministic NLALR(1) machine. This inclusion is proper: [21] abounds with nondeterministic languages which are NSLR(1) and are thus NLALR(1).

7. Real-life Application

Finding practical and interesting grammatical difficulties that are immediately solved by the use of a NLALR(1) parser generator is not so easy. Common examples are very often NLALR(k) for small values of k. On the other hand, they can easily be transformed into NLALR(1), by ways similar to the transformation of NSLR(k) grammars to NSLR(1) given in [21]. In particular, unit productions, usually avoided in LALR(1) grammars for performance reasons, often allow to very simply get a NLALR(1) grammar.

7.1 Problem Description

We address the problem of the Java field and method declarations syntax, described in sections 19.1.2 and 19.1.3 of [11]. The corresponding excerpt of the Java specification grammar is NLALR(2), but can be made NLALR(1); Table 2 presents the original grammar excerpt.

Consider the partial Java input:

```
class Problem { public static int length
```

When the parser sees the terminal symbol `public` in its lookahead window, it cannot tell whether ε should be reduced to a $<FieldModifier>$ as in:

```
public static int length = 1;
```

or to a $<MethodModifier>$ as in:

```
public static int length(String s) {
    return s.length();
}
```

The same happens between `public` and `static`, and after reading `static` and seeing `int` as lookahead symbol. This set of grammar productions is not LR(k) for any fixed k since we could repeat the modifiers indefinitely. A similar problem happens after reading `int` and seeing the identifier `length` as lookahead symbol, where the parser does not know whether to reduce to $<Type>$ or to $<ResultType>$.

7.2 Canonical Solutions

The solution of [11] to this problem was to combine all kinds of modifiers in a single $<Modifier>$ nonterminal, and to check

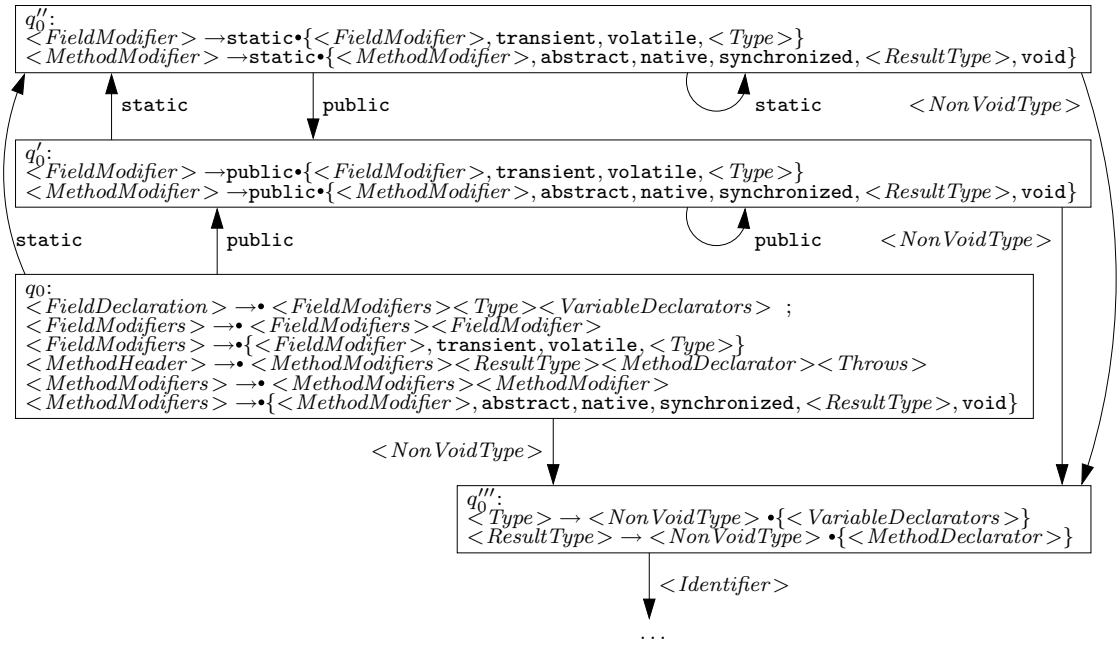

Figure 4. Partial NLALR(1) automaton for the Java fields and methods declaration syntax.

whether they are allowed in their context at a later stage of compiler analysis. The $<MethodHeader>$ production was then rewritten in two rules to separate the case of a `void` return type from a $<Type>$ return type.

Another fairly common solution is to use the predicated LL(k) parser ANTLR [14]: a syntactic predicate is explicitly given by the grammar writer to help the parser distinguish fields from methods, like the following one for method declarations:

```
((MethodModifier)* ResultType MethodDeclarator)?
```

Predicates have two major drawbacks: they can be highly inefficient at parsing time, and they are error-prone since the grammar writer has to manually interfere with the parser.

7.3 Noncanonical Solution

We modified the specification grammar with the simple reassignment of the productions of $<Type>$ to $<NonVoidType>$, the addition of the unit production $<Type> \rightarrow <NonVoidType>$, and the change from $<Type>$ to $<NonVoidType>$ in the derivation of $<ResultType>$:

$$\begin{array}{c} <Type> \rightarrow <NonVoidType> \\ <ResultType> \rightarrow <NonVoidType> \mid \texttt{void} \end{array} \qquad (35)$$

We can generate a NLALR(1) parser for the modified set of grammar productions. Figure 4 presents a small portion of the NLALR(1) automaton. On the partial input presented in section 7.1, it will shift the `public` and `static` tokens, shift `int` and reduce it to a $<NonVoidType>$ and shift it, reduce `length` and parts of what follows to either $<VariableDeclarator>$ or $<MethodDeclarator>$. The parser will come back to this point later, but now it is able to solve all pending conflicts. Nonterminal $<NonVoidType>$can be reduced to $<Type>$ or $<ResultType>$, and `static` followed by `public` to a $<FieldModifier>$ or a $<MethodModifier>$ each. At last, ε and these two nonterminals are reduced to a single $<FieldModifiers>$ or$<MethodModifiers>$.

This real-life example demonstrates the suitability of NLALR(1) parsers for practical purposes. We notice in particular that the noncanonical version of the automaton has only six more states, one for each common modifier and one for $<NonVoidType>$, than its canonical counterpart for the modified grammar snippet. We would of course have preferred a solution with greater lookahead capabilities, avoiding any modification to the original specification grammar. We find consolation in the fact that the grammar transformation is very simple when compared to the usual transformations required in order to get canonical LALR(1) grammars.

8. Conclusion

We have presented a construction for noncanonical LALR(1) parsers. Such parsers have been shown to be practical for some difficult syntax problems. They improve on both noncanonical SLR(1) parsers and canonical LALR(1) parsers, and their generation is only slightly more complex while their size and their performances are comparable.

For practical uses, we feel we would need an unbounded lookahead version of NLALR(1) parsers. Though this would probably be at the expense of linear time parsing in some cases, the freedom given to the grammar writer would probably be worth it.

Finally, we point out that the definition of valid covers might be a good starting point for a formal study of practical noncanonical parsers, whereas [20] studied more theoretical methods.

References

[1] Alfred V. Aho and Jeffrey D. Ullman. *The Theory of Parsing, Translation, and Compiling*, volume I: Parsing of *Series in Automatic Computation*. Prentice Hall, Englewood Cliffs, New Jersey, 1972.

[2] Manuel E. Bermudez and Karl M. Schimpf. Practical arbitrary lookahead LR parsing. *Journal of Computer and System Sciences*, 41:230–250, 1990.

[3] Philippe Charles. *A Practical method for Constructing Efficient LALR(k) Parsers with Automatic Error Recovery*. PhD thesis, New York University, May 1991.

[4] Karel Čulik and Rina Cohen. LR-Regular grammars—an extension of LR(k) grammars. *Journal of Computer and System Sciences*, 7:66–96, 1973.

[5] Frank L. DeRemer and Thomas Pennello. Efficient computation of LALR(1) look-ahead sets. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 4(4):615–649, 1982.

[6] Franklin Lewis DeRemer. *Practical Translators for LR(k) Languages*. PhD thesis, Massachusetts Institute of Technology, Cambridge, Massachusetts, 1969.

[7] Charles Donnely and Richard Stallman. *Bison, The YACC-compatible Parser Generator*. The Free Software Foundation, February 2002.

[8] Jay Earley. An efficient context-free parsing algorithm. *Commun. ACM*, 13(2):94–102, 1970.

[9] Jacques Farré and José Fortes Gálvez. A bounded-connect construction for LR-regular parsers. In Reinhard Wilhelm, editor, *International Conference on Compiler Construction, CC 2001*, volume 2027 of *LNCS*, pages 244–258, Genova, Italy, April 2001. Springer-Verlag.

[10] Jacques Farré and José Fortes Gálvez. Bounded-connect noncanonical discriminating-reverse parsers. *Theoretical Computer Science*, 313(1):73–91, February 2004.

[11] James Gosling, Bill Joy, and Guy Steele. *The Java™ Language Specification.* Addison-Wesley Longman Publishing Co., Inc., first edition, August 1996.

[12] Stephen C. Johnson. YACC — yet another compiler compiler. Computing science technical report 32, AT&T Bell Laboratories, Murray Hill, New Jersey, July 1975.

[13] M. Dennis Mickunas, Ronald L. Lancaster, and Victor B. Schneider. Transforming LR(k) grammars to LR(1), SLR(1), and (1,1) Bounded Right-Context grammars. *Journal of the ACM (JACM)*, 23(3):511–533, 1976.

[14] Terence J. Parr and Russel W. Quong. ANTLR: A predicated-LL(k) parser generator. *Software, Practice and Experience*, 25(7):789–810, 1995.

[15] Daniel J. Salomon and Gordon V. Cormack. Scannerless NSLR(1) parsing of programming languages. In *Proceedings of the SIGPLAN '89 Conference on Programming language design and implementation*, pages 170–178. ACM Press, 1989.

[16] Sylvain Schmitz. Noncanonical LALR(1) parsing. Technical Report I3S/RR-2005-21-FR, Laboratoire I3S, November 2005. Available online at `http://www.i3s.unice.fr/~mh/RR/2005/RR-05.21-S.SCHMITZ.pdf`.

[17] B. Seité. A Yacc extension for LRR grammar parsing. *Theoretical Computer Science*, 52:91–143, 1987.

[18] Seppo Sippu and Eljas Soisalon-Soininen. *Parsing Theory*. EATCS Monographs on Theoretical Computer Science. Springer-Verlag, 1990.

[19] Thomas G. Szymanski. *Generalized Bottom-Up Parsing*. PhD thesis, Department of Computer Science, Cornell University, Ithaca, New York, May 1973.

[20] Thomas G. Szymanski and John H. Williams. Noncanonical extensions of bottom-up parsing techniques. *SIAM Journal of Computing*, 5(2):231–250, June 1976.

[21] Kuo-Chung Tai. Noncanonical SLR(1) grammars. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 1(2):295–320, 1979.

[22] Robert E. Tarjan. Depth first search and linear graph algorithms. *SIAM Journal on Computing*, 1(2):146–160, 1972.

[23] Masaru Tomita. *Efficient Parsing for Natural Language*. Kluwer Academic Publishers, Boston, 1986.

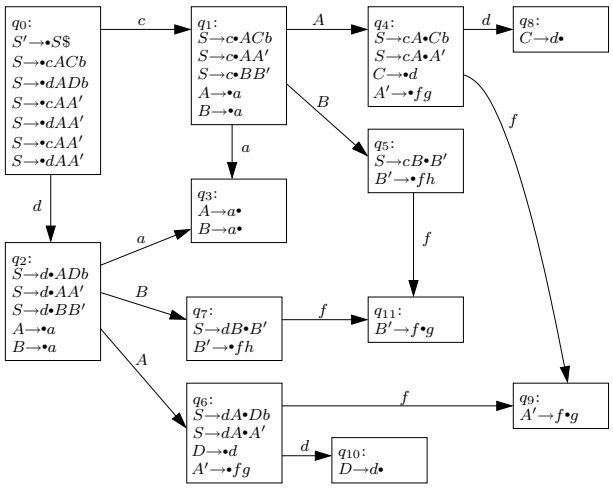

Figure 5. Partial LR(0) automaton for $\mathcal{G}_6$.

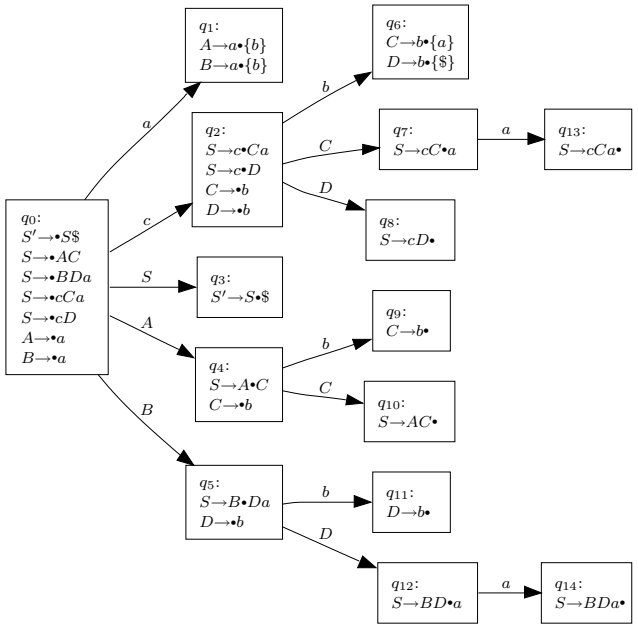

Figure 6. LALR(1) automaton for $\mathcal{G}_7$.

A. Alternative Definitions

We present here a few variants that also allow the construction of noncanonical LALR-based parsers.

A.1 Leftmost LALR(1) Alternative

If instead of deciding a reduction as early as possible we waited until we saw a totally reduced symbol in the lookahead window, then we would have defined a variant called leftmost LALR(1) parsing by analogy with leftmost SLR(1) parsing from [21].

In order to generate a leftmost LALR(1) (LLALR(1)) parser, we have to change the definition with

$$\mathrm{CLA}(s, A{\to}\alpha) = \{X \in \mathrm{DLA}(q, A{\to}\alpha) \mid q \in s \text{ and } X \notin \mathrm{RLA}(q, A{\to}\alpha)\} \text{ and} \quad (36)$$

$$\mathrm{NLA}(s, A{\to}\alpha) = \bigcup_{q\in s} \mathrm{RLA}(q, A{\to}\alpha) \quad (37)$$

instead of (15) and (16).

In [21], LSLR(1) parsers were shown to be strictly weaker than NSLR(1) parsers. The same is true for LLALR(1) parsers compared to NLALR(1) parsers. The delayed resolution can cause a conflict. Grammar $\mathcal{G}_6$ with rules

$$S{\to}cACb \mid dADb \mid cAA' \mid dAA' \mid cBB' \mid dBB',\quad A{\to}a,\ B{\to}a,\ C{\to}d,\ D{\to}d,\ A'{\to}fg,\ B'{\to}fh \quad (\mathcal{G}_6)$$

illustrates this case. Figure 5 presents a portion of the LR(0) automaton for $\mathcal{G}_6$.

The noncanonical lookaheads for the reductions in state q_3 are

$$\begin{aligned}\mathrm{RLA}(q_3, A{\to}a) &= \{C, D, A'\},\\ \mathrm{DLA}(q_3, A{\to}a) &= \{C, D, d, A', f\},\\ \mathrm{RLA}(q_3, B{\to}a) &= \{B'\} \text{ and}\\ \mathrm{DLA}(q_3, B{\to}a) &= \{B', f\}.\end{aligned}$$

In NLALR(1) parsing, noncanonical state $\{q_5\}$ has a transition on f to noncanonical state $\{q_9, q_{11}\}$, where the conflict is then solved by shifting g or h and reducing to A' or B'.

In LLALR(1) parsing, there is another transition on d from $\{q_5\}$ to $\{q_8, q_{10}\}$. This new state has a reduce/reduce conflict where both $\mathrm{RLA}(q_8, C{\to}d)$ and $\mathrm{RLA}(q_{10}, D{\to}d)$ contain symbol b. Grammar $\mathcal{G}_6$ is not LLALR(1).

A.2 Item-based Alternative

In the item-based variant, noncanonical states are collections of valid LR(0) items instead of sets of LR(0) states. This allows to merge states with identical Kernel item sets as defined in (2–3), and to produce more compact automata. There is however an impact on the power of the parsing method.

Consider grammar with rules

$$S{\to}AC \mid BDa \mid cCa \mid cD,\ A{\to}a,\ B{\to}a,\ C{\to}b,\ D{\to}b. \quad (\mathcal{G}_7)$$

Figure 6 shows the LALR(1) automaton for $\mathcal{G}_7$.

Using the state-based NLALR(1) definition, we just add a transition on b from noncanonical state $\{q_1\}$ to noncanonical state $\{q_9, q_{11}\}$ where the reductions of b to C and D are associated with lookahead symbols \$ and a respectiveley.

Using the item-based definition, the state reached by a transition on b from state $\{A{\to}a\bullet, B{\to}a\bullet\} = \mathrm{Valid}(a)$ is state $\{C{\to}b\bullet, D{\to}b\bullet\} = \mathrm{Valid}(cb)$ where both symbols a and \$ are possible lookaheads for both reductions using $C{\to}b$ and $D{\to}b$. These symbols are totally reduced, and the generated parser for $\mathcal{G}_7$ is nondeterministic.

A solution for combining the strength of the state-based approach with the smaller automata of the item-based variant could be to consider the lookahead sets associated with each reduction item. We would then merge states with the same noncanonical LALR(1) item sets instead of the same LR(0) item sets. Since this would be comparable to a state merge once the construction is over, compression techniques for parsing tables seem a better choice.

A.3 Transition-based Alternative

The two previous modifications we presented were weaker than the regular definition. The transition-based definition is stronger, but the generated noncanonical parsers are not isomorphic to LR(0) parsers on the conflict-free portions of the parser.

In the transition-based alternative, noncanonical states are sets of LR(0) transitions. All computations would then be based on LR(0) transitions instead of LR(0) states. We express this alterna-

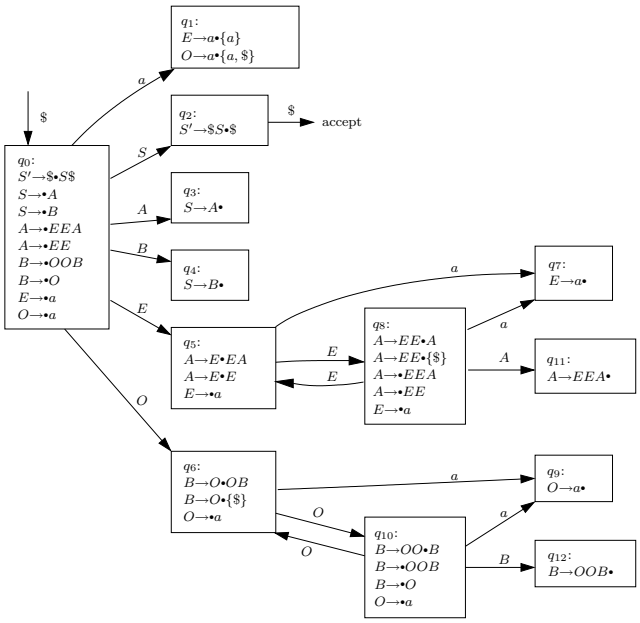

Figure 7. The LALR(1) automaton for $\mathcal{G}_3$.

tive in terms of the relations of Section 5.1 because they involve LR(0) transitions.

First, we would augment our grammars with rule $S' \to \$S\$$ in order to have an incoming transition to the initial LR(0) state $[\$]$.

Then, we would redefine the **lookback** relation. For a reduction using $A \to \alpha$, instead of looking back from q to any transition $([\delta], A)$ such that there is a path accessing q on $\delta\alpha$, we provide a transition (p, Y) accessing q and want to find transitions $([\delta], A)$ with a path starting in $[\delta]$ and accessing q on $\delta\alpha$ and ending by transition (p, Y).

$$((p, Y), A \to \alpha)\,\textbf{lookback}\,(q, A) \text{ iff } p = [\varphi], q = [\delta] \text{ and } \delta\alpha = \varphi Y, \tag{38}$$

where $([\varphi Y], A \to \alpha)$ is a reduction using $A \to \alpha$ in state $[\varphi Y]$. All noncanonical lookaheads computations would then be modified accordingly.

At last, we would redefine noncanonical states as sets of LR(0) transitions defined by

$$\begin{aligned}
[\![\$]\!] &= \{([\varepsilon], \$)\} \text{ and} \\
[\![\delta X]\!] &= \{([\gamma C], X) \mid X \in \text{CLA}([\![\delta]\!], A \to \alpha), (p, Y) \in [\![\delta]\!] \text{ and} \\
&\qquad ((p, Y), A \to \alpha)\,\textbf{lookback} \circ \textbf{includes}^* \circ \textbf{reads}^*([\gamma], C)\} \\
&\quad \cup \{([\varphi Y], X) \mid ([\varphi], Y) \in [\![\delta]\!]\}.
\end{aligned} \tag{39}$$

Let us illustrate these definitions on an example. Remember Grammar $\mathcal{G}_3$ from page 7; its LALR(1) automaton is presented in Figure 7.

Using the state-based NLALR(1) construction, we would create a transition on symbol a from state $\{q_1\}$ to state $\{q_7, q_9\}$, where totally reduced symbol \$ would appear in the lookahead window of both reductions using $E \to a$ and $O \to a$.

Using the transition-based construction, noncanonical state $\{(q_0, a)\}$ corresponding to the LR(0) state q_1 would have a transition on symbol a to $\{(q_5, a), (q_6, a)\}$ where \$ appears only in the lookahead window of reduction using $E \to a$. Noncanonical state $\{(q_5, a), (q_6, a)\}$ would also have a transition on a to $\{(q_8, a), (q_{10}, a)\}$ where \$ appears only in the lookahead window of reduction using $O \to a$. At last, state $\{(q_8, a), (q_{10}, a)\}$ would have a transition on a back to $\{(q_5, a), (q_6, a)\}$. Figure 8 recapitulates these computations.

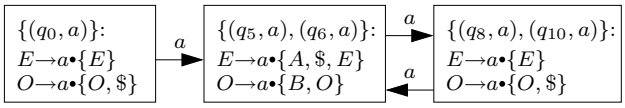

Figure 8. State q_1 of $\mathcal{G}_3$ transformed and extended for noncanonical parsing.

But we also notice that LR(0) state q_5 would be split into noncanonical states $[\![\$E]\!] = \{(q_0, E)\}$ and $[\![\$EE(EE)^*E]\!] = \{(q_8, E)\}$. This construction is not really LALR in spirit. In fact, it could be generalized by having n-tuples of LR(0) transitions as noncanonical states, improving the precision of the lookahead computations at the expense of an increased size for the resulting automaton.

B. Omitted Proofs

B.1 Proof of Theorem 1

THEOREM 1. *Let $\mathcal{G}$ be a grammar and (M, τ) its NLALR(1) bottom-up parser. Then $\mathcal{L}_M \subseteq \mathcal{L}_\mathcal{G}$ and if π is a parse of w in M, then the number of parsing steps $|\pi|$ is related to the number $|\tau(\pi)|$ of derivations producing w in $\mathcal{G}$ and to the length $|w|$ of w by*

$$|\pi| = 2|\tau(\pi)| + |w|.$$

We start by proving the following lemma inspired by Lemma 5.17 from [18].

LEMMA 1. *Let $\mathcal{G}$ be a CFG and (M, τ) its NLALR(1) parser. Further, let $X_1 \dots X_n$, $Y_1 \dots Y_m$, α and β be strings in V^* and π an action string in R^* such that*

$$[\![\varepsilon]\!][\![X_1]\!] \dots [\![X_1 \dots X_n]\!] \| \alpha\$ \models_\pi [\![\varepsilon]\!][\![Y_1]\!] \dots [\![Y_1 \dots Y_m]\!] \| \beta\$. \tag{40}$$

Then,

$$\begin{aligned}
&|\pi| = 2|\tau(\pi)| + |\alpha| - |\beta| \text{ and} \\
&Y_1 \dots Y_m \beta \overset{\tau(\pi)^R}{\Rightarrow} X_1 \dots X_n \alpha \text{ in } \mathcal{G}.
\end{aligned} \tag{41}$$

Proof. The proof is by induction on the length of action string π.

If $\pi = \varepsilon$, then (41) clearly holds.

Let us prove the induction step. We assume that $\pi = r\pi'$ where r is a single reduce or a single shift action, and where the lemma holds for π'.

If r is a reduce action, then for some number p with $1 \le p \le n$ and rule $A \to X_p \dots X_n$,

$$\begin{aligned}
&\models_{A \to X_p \dots X_n} && [\![\varepsilon]\!][\![X_1]\!] \dots [\![X_1 \dots X_n]\!] \| \alpha\$ \\
&\models_{\pi'} && [\![\varepsilon]\!][\![X_1]\!] \dots [\![X_1 \dots X_{p-1}]\!] \| A\alpha\$ \\
& && [\![\varepsilon]\!][\![Y_1]\!] \dots [\![Y_1 \dots Y_m]\!] \| \beta\$ \text{ in } M.
\end{aligned}$$

By induction hypothesis,

$$\begin{aligned}
&|\pi'| = |\tau(\pi')| + |A\alpha| - |\beta| \text{ and} \\
&Y_1 \dots Y_m \beta \overset{\tau(\pi')^R}{\Rightarrow} X_1 \dots X_{p-1} A\alpha \text{ in } \mathcal{G}.
\end{aligned}$$

Thus

$$\begin{aligned}
&|\pi| = |r| + |\pi'| = |r| + |A| + 2|\tau(\pi')| + |\alpha| - |\beta| \text{ and} \\
&Y_1 \dots Y_m \beta \overset{\tau(\pi')^R}{\Rightarrow} X_1 \dots X_{p-1} A\alpha \overset{A \to X_p \dots X_n}{\Rightarrow} X_1 \dots X_n \alpha \text{ in } \mathcal{G},
\end{aligned}$$

i.e. (41) holds.

If r is a shift action, then we rewrite α as $X\omega$ and

$$\begin{array}{ll} & [\![\varepsilon]\!][\![X_1]\!]\dots[\![X_1\dots X_n]\!]\|X\omega\$ \\ \models_{\overline{\text{shift}}} & [\![\varepsilon]\!][\![X_1]\!]\dots[\![X_1\dots X_n]\!][\![X_1\dots X_nX]\!]\|\omega\$ \\ \models_{\overline{\pi'}} & [\![\varepsilon]\!][\![Y_1]\!]\dots[\![Y_1\dots Y_m]\!]\|\beta\$ \text{ in } M. \end{array}$$

By induction hypothesis,

$$|\pi'| = |\tau(\pi')| + |\omega| - |\beta| \text{ and}$$
$$Y_1\dots Y_m\beta \overset{\tau(\pi')^R}{\Rightarrow} X_1\dots X_nX\omega \text{ in } \mathcal{G}.$$

Thus

$$|\pi| = |r| + |\pi'| = 2|\tau(\pi')| + |X| + |\omega| - |\beta| \text{ and}$$
$$Y_1\dots Y_m\beta \overset{\tau(\pi')^R}{\Rightarrow} X_1\dots X_nX\omega = X_1\dots X_n\alpha \text{ in } \mathcal{G},$$

where $|\tau(r)| = 0$ and $|X| + |\omega| = |\alpha|$, thus (41) holds. □

The proof of Theorem 1 is then immediate when choosing w for α, ε for $X_1\dots X_n$ and β and S for $Y_1\dots Y_m$ in Lemma 1.

B.2 Proof of Theorem 2

THEOREM 2.

$$\begin{array}{l} \mathit{RLA}(q, A{\to}\alpha) = \{X \mid X{\Rightarrow}^*ax, \psi{\Rightarrow}^*\varepsilon, \\ \qquad\qquad C{\Rightarrow}\rho B{\bullet}\psi X\sigma \in \mathit{Kernel}(\delta\rho B) \text{ and} \\ \qquad (q, A{\to}\alpha)\ \textbf{\textit{lookback}}\ \circ\ \textbf{\textit{includes}}^*([\delta\rho], B)\}. \end{array}$$

Theorem 2 is an immediate application of Lemma 2, and provides a computation for the totally reduced lookaheads. We quote first a technical lemma given in [18].

LEMMA 7.14 of [18]. *Let $\mathcal{G} = \langle V, T, P, S\rangle$ be a grammar. Further, let A be a nonterminal, X and Y symbols in V, γ and ψ strings in V^*, y a string in T^*, and π a rule string in P^* such that*

$$A \overset{\pi}{\underset{\text{rm}}{\Rightarrow}} \gamma X\psi Y y \text{ and } \psi{\Rightarrow}^*\varepsilon. \tag{42}$$

Then there are symbols X' and Y' in V, a rule $r' = B{\to}\alpha X'\psi' Y'\beta$ in P, and strings γ', α', β' in V^* and y' in T^* such that

$$\begin{array}{l} A \overset{\pi'}{\underset{\text{rm}}{\Rightarrow}} \gamma' By' \overset{r'}{\underset{\text{rm}}{\Rightarrow}} \gamma'\alpha X'\psi' Y'\beta y',\ \gamma'\alpha\alpha' = \gamma, \\ X'{\underset{\text{rm}}{\Rightarrow}}^*\alpha' X,\ \psi'{\Rightarrow}^*\varepsilon, \text{ and } Y'{\underset{\text{lm}}{\Rightarrow}}^*Y\beta', \end{array} \tag{43}$$

where $\pi' r'$ is a prefix of π.

LEMMA 2. *Non null symbol X belongs to $\mathit{RLA}(q, A{\to}\alpha)$ if and only if there is a rule $C{\to}\rho B\psi X\sigma$ with $\psi{\Rightarrow}^*\varepsilon$ and state $[\delta\rho]$ such that*

$$\begin{array}{l} (q, A{\to}\alpha)\ \textbf{\textit{lookback}}\ \circ\ \textbf{\textit{includes}}^*([\delta\rho], B) \text{ and} \\ C{\to}\rho{\bullet}B\psi X\sigma \in \mathit{Valid}(\delta\rho). \end{array}$$

Proof. This lemma is very similar to Lemma 7.15 from [18].

We first assume that X belongs to $\text{RLA}(q, A{\to}\alpha)$. Then

$$S'{\underset{\text{lm}}{\Rightarrow}}^* zA\gamma X\omega, \gamma{\Rightarrow}^*\varepsilon, X{\Rightarrow}^*ax \text{ and } q = [\hat{z}\alpha];$$

we transform this derivation into a rightmost derivation:

$$S'{\underset{\text{rm}}{\Rightarrow}}^*\hat{z}A\gamma Xy{\underset{\text{rm}}{\Rightarrow}}^*\hat{z}Aaxy \text{ where } \omega{\Rightarrow}^*y.$$

Then, we can apply Lemma 7.14 from [18], hence there exists rule $C{\to}\rho B\psi X'\sigma$ and strings δ, β, φ and u such that

$$S'{\underset{\text{rm}}{\Rightarrow}}^*\delta Cu{\underset{\text{rm}}{\Rightarrow}}\delta\rho B\psi X'\sigma u, \delta\rho\beta = \hat{z},$$
$$B{\underset{\text{rm}}{\Rightarrow}}^*\beta A, \psi{\Rightarrow}^*\varepsilon \text{ and } X'{\underset{\text{lm}}{\Rightarrow}}^*X\varphi.$$

Remember that X appeared in the leftmost derivation $S'{\underset{\text{lm}}{\Rightarrow}}^* zA\gamma X\omega$, where it could not be derived by X' in a leftmost order, thus

$$X' = X \text{ and } \varphi = \varepsilon.$$

Thus $C{\to}\rho{\bullet}B\psi X\sigma \in \text{Valid}(\delta\rho)$ and

$$([\hat{z}], A)\ \textbf{includes}^*([\delta\rho], B), \text{ where}$$
$$([\hat{z}\alpha], A{\to}\alpha)\ \textbf{lookback}\ ([\hat{z}], A),$$

from which we deduce the result.

Conversely, since $C{\to}\rho{\bullet}B\psi X\sigma \in \text{Valid}(\delta\rho)$,

$$S'{\underset{\text{rm}}{\Rightarrow}}^*\delta\rho B\psi X\sigma u,$$

and since $([\hat{z}\alpha], A{\to}\alpha)\ \textbf{lookback}\ \circ\ \textbf{includes}^*([\delta\rho], B)$,

$$B{\Rightarrow}^*\beta A\gamma \text{ with } \gamma{\Rightarrow}^*\varepsilon.$$

Therefore, converting to leftmost derivations,

$$\begin{array}{l} S'{\underset{\text{lm}}{\Rightarrow}}^* yB\psi X\sigma\omega \text{ where } \delta\rho{\Rightarrow}^*y \text{ and } \omega{\Rightarrow}^*u, \\ \quad{\underset{\text{lm}}{\Rightarrow}}^* zA\gamma\psi X\sigma\omega \text{ where } \gamma\psi{\Rightarrow}^*\varepsilon \text{ and } \delta\rho\beta{\Rightarrow}^*z. \end{array}$$

Hence non null symbol X complies with the conditions of $\text{RLA}([\hat{z}\alpha], A{\to}\alpha)$. □

B.3 Proof of Theorem 3

THEOREM 3. *Let us extend the* directly reads *function of (26) to*

$$\mathit{DR}([\delta], A) = \{X \mid ([\delta A], X) \text{ and } X{\Rightarrow}^*ax\},$$

then

$$\mathit{DLA}(q, A{\to}\alpha) = \bigcup_{(q, A\to\alpha)\ \textbf{\textit{lookback}}\ \circ\ \textbf{\textit{includes}}^*\circ\ \textbf{\textit{reads}}^*(r,C)} \mathit{DR}(r, C).$$

Proof. Let non null symbol $X{\Rightarrow}^*ax$ be in $\text{DLA}(q, A{\to}\alpha)$ with $q = [\hat{\delta}\alpha]$. Let us show it is in the doirectly read set of some related transition. We have

$$S'{\Rightarrow}^*\delta AX\omega, \tag{44}$$

thus

$$S'{\underset{\text{rm}}{\Rightarrow}}^*\hat{\delta}AXy \text{ with } \omega{\Rightarrow}^*y. \tag{45}$$

Then, there exist a rule $C{\to}\beta BX\sigma$ and strings γ and y' such that

$$S'{\underset{\text{rm}}{\Rightarrow}}^*\gamma Cy'{\underset{\text{rm}}{\Rightarrow}}\gamma\beta BX\sigma y'{\underset{\text{rm}}{\Rightarrow}}^*\gamma\beta BXy{\underset{\text{rm}}{\Rightarrow}}^*\gamma\beta Baxy{\underset{\text{rm}}{\Rightarrow}}^*\hat{\delta}Aaxy. \tag{46}$$

Therefore, $X \in \text{DR}([\gamma\beta], B)$ where $\gamma\beta B$ is a valid cover for $\hat{\delta}A$. By Theorem 4, we get the desired result.

Conversely, if X is in $\text{DR}([\gamma], C)$ where γC is a valid cover for $\hat{\delta}A$, then

$$S'{\underset{\text{rm}}{\Rightarrow}}^*\gamma CXz{\Rightarrow}^*\hat{\delta}AXz{\Rightarrow}^*\delta AXz, \tag{47}$$

and thus X is in $\text{DLA}([\hat{\delta}\alpha], A{\to}\alpha)$. □

We could have proven this theorem using Lemma 2 and the following lemma inspired by lemma 7.13 of [18]:

LEMMA 3. *Let*

$$\mathit{DFirst}(\alpha) = \{X \mid \alpha{\Rightarrow}^*X\omega{\Rightarrow}^*ax\omega\},$$

then X is in $\mathit{DR}(r, C)$ with $([\gamma], A)$ $\textbf{\textit{reads}}^*(r, C)$ if and only if $\mathit{Valid}(\gamma)$ contains an item $B{\to}\beta{\bullet}A\alpha$ and X is in $\mathit{DFirst}(\alpha)$.

The noncanonical lookaheads could then be derived from the totally reduced lookaheads, making the computation of the **reads*** relation unnecessary. But we need to compute **reads***in order to get the valid covers for the noncanonical transitions. We might as well use this information for the computation of the noncanonical lookaheads, and avoid the computation of the DFirst sets.

B.4 Proof of Theorem 4

THEOREM 4. *Let α be a phrase such that $S'\Rightarrow^*\delta AX\omega\Rightarrow\delta\alpha X\omega$ with $q = [\hat{\delta}\alpha]$, then*

$$\mathit{Cover}(\hat{\delta}AX) = \{\gamma CX \mid (q, A\rightarrow\alpha)\ \textbf{\textit{lookback}} \circ \textbf{\textit{includes}}^* \circ \textbf{\textit{reads}}^*([\gamma], C)\}.$$

The proof is immediate using the following lemma.

LEMMA 4. *Let δA be a valid prefix; then*

$$\mathit{Cover}(\delta A) = \{\gamma C \mid ([\delta], A)\ \textbf{\textit{includes}}^* \circ \textbf{\textit{reads}}^*([\gamma], C)\}.$$

Proof. Let $([\delta], A)$ **includes**$^*\circ$ **reads**$^*([\gamma], C)$, and let us show that γC is a valid cover for δA.

1. if $([\varphi\beta], A)$ **includes** $([\varphi], B)$, then φB is a valid cover for $\varphi\beta A$, since $\varphi B\Rightarrow\varphi\beta A\gamma\Rightarrow\varphi\beta A$, and
2. if $([\varphi], B)$ **reads** $([\varphi B], C)$, then φBC is a valid cover for φB since $\varphi BC\Rightarrow\varphi B$.

Conversely, let δA be a valid prefix and γC a valid cover for δA, and let us show that $([\delta], A)$ **includes**$^*\circ$ **reads**$^*([\gamma], C)$. Since δA is a valid prefix and γC is a valid cover for δA,

$$S'\underset{\text{rm}}{\Rightarrow}{}^*\gamma Cz\underset{\text{rm}}{\Rightarrow}{}^*\delta Az.$$

Suppose $C\Rightarrow^*\varepsilon$. Then $\gamma C\underset{\text{rm}}{\Rightarrow}\gamma\underset{\text{rm}}{\Rightarrow}{}^*\delta A$, thus γ ends with a nonterminal and can be rewritten as $\gamma' C'$, and $([\gamma'], C')$ **reads** $([\gamma], C)$ in this case. This can be iterated over any nullable suffix of γC.

What remains to be proven is that if $S'\underset{\text{rm}}{\Rightarrow}{}^*\gamma Cz\underset{\text{rm}}{\Rightarrow}{}^*\delta Az$ with $C\not\Rightarrow^*\varepsilon$, then $([\delta], A)$ **includes**$^*([\gamma], C)$. But in this case $C\underset{\text{rm}}{\Rightarrow}{}^*\beta A$ with $\delta = \gamma\beta$, and by Lemma 7.9 of [18], $([\delta], A)$ **includes**$^*([\gamma], C)$. □

B.5 Proof of Theorem 5

THEOREM 5. *Every NLALR(1) grammar is unambiguous.*

Proof. Suppose there is an ambiguous NLALR(1) grammar $\mathcal{G}$. Since $\mathcal{G}$ is ambiguous, then the following derivations hold in $\mathcal{G}$:

$$S'\underset{\text{lm}}{\Rightarrow}{}^*xA\omega\underset{\text{lm}}{\Rightarrow}x\alpha\omega\underset{\text{lm}}{\Rightarrow}{}^*z\omega \text{ and}$$
$$S'\underset{\text{lm}}{\Rightarrow}{}^*yB\sigma\underset{\text{lm}}{\Rightarrow}y\beta\rho\sigma\underset{\text{lm}}{\Rightarrow}{}^*z\rho\sigma \text{ with}$$
$$\omega = \rho\sigma.$$

Supposing further that no other conflict on totally reduced symbols arose on a shorter prefix, there is a string ψ with $\psi\Rightarrow^*z$ such that in noncanonical state $[\![\psi]\!]$, we find the LR(0) states $[\hat{x}\alpha]$ and $[\hat{y}\beta]$.

String ω cannot be empty since it ends with \$. Let $\omega = \gamma X\varphi$ with $\gamma\Rightarrow^*\varepsilon$ and $X\Rightarrow^*au$; by definition of a totally reduced lookahead,

$$X \in \text{RLA}([\hat{x}\alpha], A\rightarrow\alpha).$$

Moreover, if $\rho = \varepsilon$, then

$$X \in \text{RLA}([\hat{y}\beta], B\rightarrow\beta),$$

and $[\hat{y}\beta X]$ is a valid prefix if $\rho \neq \varepsilon$.

Hence, totally reduced symbol X causes a conflict in $[\![\psi]\!]$, and $\mathcal{G}$ cannot be NLALR(1). □