Noncanonical LALR(1) Parsing*

Sylvain Schmitz

Laboratoire I3S, Université de Nice - Sophia Antipolis & CNRS, 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, NLALR(1). The class of grammars accepted by NLALR(1) parsers is a proper superclass of the NSLR(1) and LALR(1) grammar classes. Among the recognized languages are some nondeterministic languages. The proposed parsers retain many of the qualities of canonical LALR(1) parsers: they are deterministic, easy to construct, and run in linear time. We argue that they could provide the basis for a range of powerful noncanonical parsers.

Key words: Noncanonical parser, deterministic parser, LALR, two-stack automaton

ACM categories: 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 LALR(1) parser generators like YACC [9]. 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 LALR(1) automaton they produce. Transforming a grammar until its LALR(1) parser becomes deterministic is arduous, and can obfuscate the attached semantics; moreover, some languages are simply not deterministic.

The expressivity gap vanishes when general parsers [6, 15] are preferred. Such a choice is however done at the expense of the detection of ambiguities. While this might seem acceptable for well established languages, for which the scrutiny of many implementors has pinpointed all ambiguous constructs, there always remains a risk of runtime problems if an unexpected ambiguity appears. The avoidance of such problems is clearly a desirable guarantee, thus motivating our option of restricting to some subclass of the unambiguous grammars.

*Also in shorter form in Zhe Dang and Oscar H. Ibarra, editors, *DLT'06*, volume 4036 of *Lecture Notes in Computer Science*, pages 95–107. doi: 10.1007/11779148_10

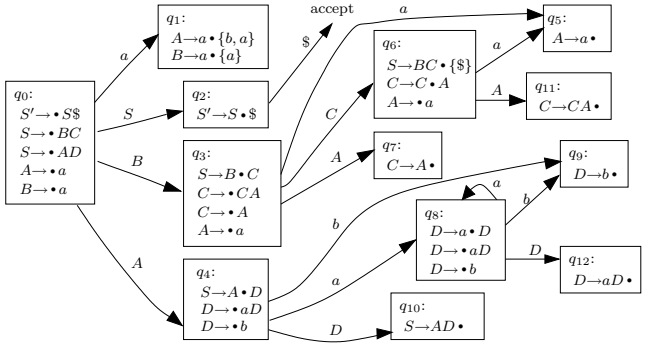
This paper advocates an almost forgotten way of diminishing the expressivity gap: the usage of *noncanonical parsers*. We apply it to LALR(1) parsing by means of a generic construction. Therefore, we also allow immediate application to other LR-based parsing methods.

Noncanonical parsers have been thoroughly investigated on a theoretical level [12]. Surprisingly, there are very few practical noncanonical parsing methods, and their formal study remains largely unexplored. Indeed, the only one of clear practical interest is an extension to SLR(1) parsing [13]. Noncanonical parsers are however a powerful means of reducing the expressivity gap, while still rejecting any ambiguous syntax. In this they can be compared to LALR(k) parsers with $k > 1$ [3], or, to a larger extent, to parsers allowing unbounded regular lookaheads [4, 2, 7]. Like the latter, noncanonical parsers can recognize nondeterministic languages. The classes of grammars accepted by both methods are incomparable in general, but the class of languages accepted by noncanonical parsers is strictly wider than the one accepted by regular lookahead parsers [12]. 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 [8]. This point motivates our study of noncanonical LALR(1) parsers, since NSLR(1) parsers are unfit for such extensions: their lookahead computation is not contextual.

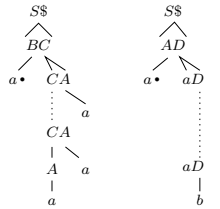
Also in contrast with NSLR(1), our definitions rely on a prefix equivalence relation: we use the LR(0) equivalence so that the resulting parsers are LALR(1), but finer equivalences could just as easily be used. Our specific choice of LALR(1) parsers can be explained by their wide adoption, their practical relevance, and the existence of efficient and broadly used algorithms for their generation [5]. We express our computations in the same framework and obtain a simple and efficient practical construction. 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.

The paper is organized as follows: Section 2 briefly introduces noncanonical parsing; Section 3 recalls the formal details of the canonical LALR(1) definition, which will be extended for its noncanonical counterpart in Section 4. We refer the interested reader to a separate research report [10] for a complete study, including grammar classes comparisons, alternative definitions for noncanonical LALR-based parsers, a concrete example of application, and omitted proofs.

Notation The basic terminology, definitions, and notational conventions used in this paper are classical [1, 11]. Our context-free grammars are reduced and augmented to $\mathcal{G}' = \langle N', T', P', S' \rangle = \langle N \cup \{S'\}, T \cup \{\$\}, P \cup \{S' \to S\$\}, S' \rangle$. As usual, $A, B, C, \ldots$ 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'^*; ε is the empty string or empty sequence; $k : \alpha$ is the prefix of length k of string α. Rightmost derivations are denoted by $\underset{\text{rm}}{\Rightarrow}$, whereas leftmost derivations are denoted by $\underset{\text{lm}}{\Rightarrow}$.



(a) LALR(1) automaton



(b) Derivation trees

Figure 1: The conflict position in state q_1 for $\mathcal{G}_1$.

2 Noncanonical Parsing

A bottom-up parser reverses the derivation steps which lead to the terminal string it parses. For most bottom-up parsers, including LALR ones, these derivations are rightmost, and therefore the reduced phrase is the leftmost one, called the *handle* of the sentential form.

Noncanonical parsers allow the reduction of phrases which may not be handles [1]. 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.

2.1 Parsing Example

Consider for instance grammar $\mathcal{G}_1$ with rules $S{\to}BC \,|\, AD$, $A{\to}a$, $B{\to}a$, $C{\to}CA \,|\, A$, $D{\to}aD \,|\, b$, generating the language $\mathcal{L}_{\mathcal{G}_1} = aa^+ \mid aa^*b$.

The state q_1 in the automaton of Figure 1a is *inadequate*: the parser is unable to decide between reductions $A{\to}a$ and $B{\to}a$ when the lookahead is a. We see on the derivation trees of Figure 1b that, in order to choose between the two reductions, the parser has to know if there is a b at the very end of the input. This need for an unbounded lookahead makes $\mathcal{G}_1$ non-LR. 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 that a single lookahead symbol (D or C) is enough: if the parser is able to explore the context on the right of the conflict, and to

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

The inadequate state q_1 is reached 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 perform the latter decision, we shift a and reach a state s_1 where we now expect a^*b and $a^*\$$. 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 $A{\to}a$ and the shift of a to a position $D{\to}a \bullet D$. 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 $A{\to}a$

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 $A{\to}a$

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

q_0q_1	$AA\$$	reduce using $B{\to}a$

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 $C{\to}A$
q_0q_3	$CA\$$	shift
$q_0q_3q_6$	$A\$$	shift
$q_0q_3q_6q_{11}$	$\$$	reduce using $C{\to}CA$
q_0q_3	$C\$$	shift
$q_0q_3q_6$	$\$$	reduce using $S{\to}BC$
q_0	$S\$$	shift, and then accept

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

reduce some other phrases, then, it will reduce this 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.

Table 1 presents a noncanonical parse for a string in $\mathcal{L}_{\mathcal{G}_1}$. The noncanonical machine is not very different from the canonical one, except that it uses two stacks. The additional stack, the *input stack*, contains the (possibly reduced) right context, whereas the other stack is the classical *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.

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

2.2 Construction Principles

The LALR(1) construction relies heavily on the LR(0) automaton. This automaton provides a nice explanation for LALR lookahead sets: the symbols

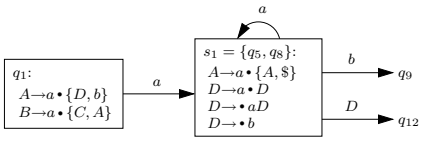


Figure 2: State q_1 extended for noncanonical parsing.

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

The intersection of the lookahead sets for the reductions in q_1 is not empty: 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. The conflicting lookahead symbol a could be reduced, and later we might see a symbol on which we can make a decision instead. Thus, 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 the same situation we would have been in if we had followed the transitions on a from both q_3 and q_4, since the noncanonical generation simulates both reductions in q_1. We create a noncanonical transition from q_1 on a to a noncanonical state $s_1 = \{q_5, q_8\}$, which will behave as the union of states q_5 and q_8. State s_1 will thus allow a reduction using $A \to a$ 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 $A \to a$ in q_5. Using again the LR(0) simulation technique, we see on Figure 1a that this reduction would lead us to either q_7 or to q_{11}. 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 $A \to a$ 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 $A \to a$. 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 2 sums up how state q_1 has been transformed and extended. Note that we just use the set $\{q_5, q_8\}$ in a noncanonical LALR(1) automaton; items represented in Figure 2 are only there to ease understanding.

3 LALR(1) Parsers

LALR parsers were introduced 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. We briefly recall

some important definitions and results on LR(0) and LALR(1) parsers.

Valid Items and Prefixes A dotted production $A \to \alpha \bullet \beta$ 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 $\text{Valid}(\gamma)$. 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:

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

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

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

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. For instance, in the automaton of Figure 1a, 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[1] a transition from $[\delta]$ on X, and $[\delta\alpha]$ a *path* on α.

LALR(1) Automata The *LALR(1) lookahead set* of a reduction using $A \to \alpha$ in state q is

$$\text{LA}(q, A \to \alpha) = \{1{:}z \mid S' \underset{\text{rm}}{\Rightarrow}^* \delta A z \text{ and } q = [\delta\alpha]\}. \tag{5}$$

4 NLALR(1) Parsers

There is a number of differences between the LALR(1) and NLALR(1) definitions. 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 (*i.e.*, sets of equivalence classes on valid prefixes), provide this approximation. We therefore define valid covers as valid prefixes covering the parsing stack language.

[1] We always assume when writing $[\delta X]$ that $\text{Valid}(\delta X)$ is not the empty set.

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(L) to denote the set of all valid covers for the set of strings L.*

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 symbols in V'. Adapting the computation of the LALR(1) lookahead sets is simple, but a few points deserve some explanations.

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

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 of a reduction 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 $A \to \alpha$ in LR(0) state q is defined by*

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

Derived Lookaheads The derived lookahead symbols are simply defined by extending (5) to the set of all non null symbols in V.

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

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

We obviously have that

$$\mathrm{LA}(q, A \to \alpha) = \mathrm{DLA}(q, A \to \alpha) \cap T'. \tag{6}$$

Conflicting Lookahead Symbols Last, we need to compute which lookahead symbols would make the state inadequate. 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*

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

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

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

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: $\mathrm{RLA}(q_5, A{\to}a) = \{A, \$\}$, $\mathrm{DLA}(q_5, A{\to}a) = \{A, a, \$\}$, $\mathrm{CLA}(s_1, A{\to}a) = \{a\}$ and $\mathrm{NLA}(s_1, A{\to}a) = \{A, \$\}$.

4.3 Noncanonical States

We said at the beginning of this section 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*

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

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.

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 $\widehat{\hat{\gamma} A X}$. It means that NLALR(1) parsers are not prefix valid, but prefix cover valid.

Throughout this paper, we use the LR(0) automaton to approximate the prefix read so far. We could use more powerful methods—but it would not really be in the spirit of LALR parsing any longer; see [10] for alternative methods.

4.4 NLALR(1) Automata

Here we formalize noncanonical LALR(1) parsing machines. They are a special case of two-stack pushdown automata (2PDA). As said before, the additional

stack serves as an input for the parser, and reductions push the reduced nonterminal on top of this stack. This behavior of reductions excepted, the definition of a NLALR(1) automaton is similar to the LALR(1) one.

Definition 6 *Let $M = (Q \cup V \cup \{\$, \|\}, R)$ be a rewriting system. A* configuration *of M is a string of the form*

$$[\![\varepsilon]\!][\![X_1]\!] \dots [\![X_1 \dots X_n]\!] \| \omega\$$$

where $X_1 \dots X_n$ and ω are strings in V^*. We say that M is a NLALR(1) automaton *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* $[\![\delta]\!]$*, defined if there is a transition* $([\![\delta]\!], X)$

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

- *or* reduce *by rule* $A \to X_1 \dots X_n$ *of* P *in state* $[\![\delta X_1 \dots X_n]\!]$ *with lookahead* X*, defined if* $A \to X_1 \dots X_n$ *is a reduction in* $[\![\delta X_1 \dots X_n]\!]$ *and lookahead* X *is in* $NLA([\![\delta X_1 \dots X_n]\!], A \to X_1 \dots X_n)$

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

The following rules illustrate Definition 6 on state s_1 of the NLALR(1) automaton for $\mathcal{G}_1$: $s_1\|a \vdash_{\text{shift}} s_1 s_1\|$, $s_1\|b \vdash_{\text{shift}} s_1\{q_9\}\|$, $s_1\|D \vdash_{\text{shift}} s_1\{q_{12}\}\|$, $s_1\|A \vdash_{A \to a} \|AA$ and $s_1\|\$ \vdash_{A \to a} \|A\$$.

According to Definition 6, NLALR(1) automata are able to backtrack by a limited amount, corresponding to the length of their window, at reduction time only. We know that noncanonical parsers using a bounded lookahead window operate in linear time [12]; 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 with the input string length. This theorem uses an output effect τ which outputs the rules used for each reduction performed by M; we then call (M, τ) a NLALR(1) parser.

Theorem 1 *Let $\mathcal{G}$ be a grammar and (M, τ) its NLALR(1) parser. 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*

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

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 its NLALR(1) automaton is deterministic, and thus if no totally reduced symbol can cause a conflict.

4.5 Computing the Lookaheads and Covers

The LALR(1) lookahead sets that are defined in Equation (5) can be expressed using the following definitions [5], 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.

$$([\delta\alpha], A{\to}\alpha)\ \textbf{lookback}\ ([\delta], A), \tag{7}$$
$$([\delta\beta], A)\ \textbf{includes}\ ([\delta], B)\ \text{iff}\ B{\to}\beta A\gamma\ \text{and}\ \gamma{\Rightarrow}^*\varepsilon, \tag{8}$$
$$([\delta], A)\ \textbf{reads}\ ([\delta A], C)\ \text{iff}\ ([\delta A], C)\ \text{and}\ C{\Rightarrow}^*\varepsilon, \tag{9}$$
$$\mathrm{DR}([\delta], A) = \{a \mid ([\delta A], a)\}. \tag{10}$$

Using the above definitions, we can rewrite Equation (5) as

$$\mathrm{LA}(q, A{\to}\alpha) = \bigcup_{(q,A\to\alpha)\ \textbf{lookback}\circ\textbf{includes}^*\circ\textbf{reads}^*(r,C)} \mathrm{DR}(r, C). \tag{11}$$

This computation for LALR(1) lookahead sets is highly efficient. It can entirely be performed on the LR(0) automaton, and the union can be interleaved with a fast transitive closure algorithm [14] on the **includes** and **reads** relations.

Since we have a very efficient and widely adopted computation for the canonical LALR(1) lookahead sets, why not try to use it for the noncanonical ones?

Theorem 2

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

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: item $S{\to}B\bullet C$ is in the kernel of state q_3 accessed by (q_0, B), and $(q_1, B{\to}a)$ **lookback** (q_0, B).

Theorem 3 *Let us extend the* directly reads *function of (10) to*

$$\begin{aligned} DR([\delta], A) &= \{X \mid ([\delta A], X)\ \textit{and}\ X{\Rightarrow}^*ax\};\ \textit{then}\\ DLA(q, A{\to}\alpha) &= \bigcup_{(q,A\to\alpha)\ \textbf{\textit{lookback}}\circ\textbf{\textit{includes}}^*\circ\textbf{\textit{reads}}^*(r,C)} DR(r, C). \end{aligned}$$

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 $\{a, C, A\}$.

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. This allows us to reuse our relational computations for the automaton construction itself, as illustrated by the following theorem.

Theorem 4 *Noncanonical state* $[\![\delta]\!]$ *is the set of LR(0) states defined by*

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

4.6 Practical Construction Steps

We present here a more informal construction, with the main steps leading to the construction of a NLALR(1) parser, given the LR(0) automaton.

1. Associate a noncanonical state $s = \{q\}$ with each LR(0) state q.
2. Iterate while there exists an inadequate[2] 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 a totally reduced lookahead, 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 computed from the LR(0) states they contain. 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.

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 generator needs to create a new state for each lookahead causing a conflict, which does not happen so often. All the grammars we studied created transitions to canonical states very quickly afterwards. Experimental results with NSLR(1) parsers show that the increase in size is negligible in practice [13].

5 Conclusion

We have presented a construction for noncanonical LALR(1) parsers. Such parsers are 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 parsers. Though the cost to pay might be a quadratic parsing time in the worst case, the freedom offered to the grammar writer would probably be

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

worth it. The ability to specify finer equivalence relations instead of the LR(0) one would prove its usefulness in this setting where precision becomes critical.

In complement to previous theoretical work on noncanonical parsing [12], it would be interesting to formally study practical noncanonical parsers. To this end, we expect the concept of valid covers modulo an equivalence relation to be a good starting point.

Acknowledgements The author is highly grateful to Jacques Farré and Ana Almeida Matos for their invaluable help in the preparation of this paper.

References

[1] Alfred V. Aho and Jeffrey D. Ullman. *The Theory of Parsing, Translation, and Compiling. Volume I: Parsing.* Series in Automatic Computation. Prentice Hall, 1972. ISBN 0-13-914556-7. URL `http://portal.acm.org/citation.cfm?id=SERIES11430.578789`.

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

[3] Philippe Charles. *A Practical method for Constructing Efficient LALR(k) Parsers with Automatic Error Recovery.* PhD thesis, New York University, May 1991. URL `http://jikes.sourceforge.net/documents/thesis.pdf`.

[4] Karel Čulik and Rina Cohen. LR-Regular grammars—an extension of LR(k) grammars. *Journal of Computer and System Sciences*, 7(1):66–96, 1973. ISSN 0022-0000. doi: 10.1016/S0022-0000(73)80050-9.

[5] Frank DeRemer and Thomas Pennello. Efficient computation of LALR(1) look-ahead sets. *ACM Transactions on Programming Languages and Systems*, 4(4):615–649, 1982. ISSN 0164-0925. doi: 10.1145/69622.357187.

[6] Jay Earley. An efficient context-free parsing algorithm. *Communications of the ACM*, 13(2):94–102, 1970. ISSN 0001-0782. doi: 10.1145/362007.362035.

[7] Jacques Farré and José Fortes Gálvez. A bounded-connect construction for LR-Regular parsers. In Reinhard Wilhelm, editor, *CC'01*, volume 2027 of *Lecture Notes in Computer Science*, pages 244–258. Springer, 2001. URL `http://springerlink.com/content/e3e8g77kxevkyjfd`.

[8] Jacques Farré and José Fortes Gálvez. Bounded-connect noncanonical discriminating-reverse parsers. *Theoretical Computer Science*, 313(1):73–91, 2004. ISSN 0304-3975. doi: 10.1016/j.tcs.2003.10.006.

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

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

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

[12] Thomas G. Szymanski and John H. Williams. Noncanonical extensions of bottom-up parsing techniques. *SIAM Journal on Computing*, 5(2):231–250, 1976. ISSN 0097-5397. doi: 10.1137/0205019.

[13] Kuo-Chung Tai. Noncanonical SLR(1) grammars. *ACM Transactions on Programming Languages and Systems*, 1(2):295–320, 1979. ISSN 0164-0925. doi: 10.1145/357073.357083.

[14] Robert E. Tarjan. Depth first search and linear graph algorithms. *SIAM Journal on Computing*, 1(2):146–160, 1972. ISSN 0097-5397. doi: 10.1137/0201010.

[15] Masaru Tomita. *Efficient Parsing for Natural Language*. Kluwer Academic Publishers, 1986. ISBN 0-89838-202-5.