Shift-Resolve Parsing: Simple, Unbounded Lookahead, Linear Time*

José Fortes Gálvez Sylvain Schmitz Jacques Farré

Universidad de Las Palmas de Gran Canaria, Spain
jfortes@dis.ulpgc.es

Laboratoire I3S, Université de Nice - Sophia Antipolis & CNRS, France
schmitz@i3s.unice.fr, Jacques.Farre@unice.fr

Abstract

This paper introduces a mechanism for combining unbounded lookahead exploration with linear time complexity in a deterministic parser. The idea is to use a *resolve* parsing action in place of the classical reduce. The construction of shift-resolve parsers is presented as a two-step algorithm, from the grammar to a finite nondeterministic automaton, and from this automaton to the deterministic parser. Grammar classes comparisons are provided.

Key words: Shift-resolve, noncanonical parser, regular lookahead, two-stack pushdown automaton, position graph.

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

Common deterministic parser generators [5] provide a parser developer with two interesting static guarantees: that the input grammar is unambiguous, and that the resulting parser will process its input string in linear time. There is however a major issue with these parser generation algorithms: they cannot provide a deterministic parser for an arbitrary context-free grammar, resulting in the infamous *conflicts* between possible parsing actions. Their inability to deal with parsing decisions that need more than the pre-established k lookahead terminal symbols is to blame for a large part of it.

Two different parsing techniques allow to circumvent this limitation to bounded lookaheads in bottom-up parsers, but to keep the unambiguity guarantee. The first, called regular lookahead parsing, uses a finite state automaton to

*Also in shorter form in Oscar H. Ibarra and Hsu-Chun Yen, editors, *CIAA'06*, volume 4094 of *Lecture Notes in Computer Science*, pages 253–264. doi: 10.1007/11812128_24

explore an unbounded right context [2, 3, 6]. The linear time guarantee is however lost. The second, called noncanonical parsing, explores the right context using the parser itself. The latter can thus perform some reductions in this right context, return to the conflict point, and use a bounded number of the newly reduced symbols to yield a deterministic decision [14, 15, 12]. However, the preset bound on the reduced lookahead length—in practice the bound is $k = 1$—hampers the power of the noncanonical methods.

We want to have our cake and eat it too: we want linear time parsing, ambiguity detection, and no user defined bound on the lookahead length. Shift-resolve parsing is a new combination of the regular and noncanonical strategies that achieves all these properties. To this end, we make the following contributions.

- We propose a new parsing action, *resolve* (Section 2.1), which combines the classical reduction with a pushback, *i.e.* it rewinds the stack down to the point where the reduction should take place. The exact amount of pushback is not fixed, but computed for each reduction as a minimal necessary length.
- By promoting the resolve action as a replacement for the reduce action, our parsers properly integrate noncanonical resolutions in the right context exploration (Section 2.2). One could fear that a quadratic time complexity would stem from this combination. We avoid it by ensuring that the pushback lengths remain bounded.
- We present the construction of shift-resolve parsers as the determinization of a phrase recognizer (Section 4). The algorithm generalizes similar constructions for LR parsers. The choice of the approximations used in order to have a finite recognizer is left open, and we use a lattice of possible approximations (Section 3.2). Hence, our method is highly generic and allows for tradeoffs between descriptional complexity and classes of accepted grammars.

2 Shift-Resolve Parsing

A bottom-up parser operates by reverting the derivations that led from the axiom of the grammar to the input string. Each of these reversions is the *reduction* of a *phrase* α to a nonterminal A, where $A \to \alpha$ is a rule of P. A canonical parser always reduces the leftmost phrase in a given sentential form, called the *handle* of the sentential form, but a noncanonical parser partially ignores this ordering. It is able to reduce a phrase further right from a handle, and to use the additional information provided by the newly reduced nonterminals to infer its parsing decisions. Indeed, a single nonterminal symbol describes a complete context-free language, and, using only a few nonterminals as lookahead, a noncanonical parser has an impressive amount of right context information at its disposal.

2.1 The Approach

We make here the simplifying choice of always using completely reduced lookahead symbols: symbols as they appear in the grammar rule we are exploring,

	\$	a	b	c	S	A	B	C	D
q_0		s_4	s_5		s_1	s_2	s_3		
q_1	r_1'0								
q_2				s_8				s_6	s_7
q_3				s_8				s_9	s_{10}
q_4				s_8				r_5'0	r_5'0
q_5				s_8				r_7'0	r_7'0
q_6		s_{11}		s_8					
q_7				s_8				r_4'0	r_4'0
q_8		r_8'0	r_9'0	s_8				s_{12}	s_{13}
q_9				s_8				r_6'0	r_6'0
q_{10}			s_{14}	s_8					
q_{11}	r_2'0								
q_{12}		r_9'1		s_8				r_8'1	r_8'1
q_{13}			r_8'1	s_8				r_9'1	r_9'1
q_{14}	r_3'0								

Table 1: Shift-resolve parsing table for $\mathcal{G}_1$.

and cannot be reduced without reducing the entire rule.

As usual in noncanonical parsing [1], a deterministic two-stack model is used to hold the current sentential form. The parsing (or left) stack corresponds to the traditional LR stack, while the input (or right) stack initially contains the input string. Two operations allow to move symbols from the top of one stack to the top of the other: a *shift* of a symbol from the input stack to the parsing stack, and a *pushback* of a bounded number of symbols the other way around. A *reduction* using rule $A \to \alpha$ removes the topmost $|\alpha|$ symbols from the parsing stack and pushes A on top of the input stack.

We compute, for each reduction, the minimal bounded reduced lookahead length needed to discriminate it from other parsing actions. This lookahead exploration is properly integrated in the parser. Once the parser succeeds in telling which action should have been done, we either keep parsing if it was a shift, or need to reduce at an earlier point. The pushback brings the parser back at this point; we call the combination of a pushback and a reduction a *resolution*.

No cost is paid in terms of computational complexity, since shift-resolve parsers are linear in the length of the input text. A simple proof is that the only re-explored symbols are those pushed back. Since pushback lengths are bounded, and since each reduction gives place to a single pushback, the time linearity is clear if the number of reductions is linear with the input length. This last point stems from the fact that our method detects and rejects cyclic grammars.

2.2 Parsing Example

Let us consider the extended grammar with rules

$$\begin{array}{l} S' \xrightarrow{1} S,\ S \xrightarrow{2} ACa,\ S \xrightarrow{3} BDb,\ A \xrightarrow{4} AD, \\ A \xrightarrow{5} a,\ B \xrightarrow{6} BC,\ B \xrightarrow{7} b,\ C \xrightarrow{8} c,\ D \xrightarrow{9} c. \end{array} \qquad (\mathcal{G}_1)$$

Grammar $\mathcal{G}_1$ can require an unbounded lookahead if we consider approximated parsing methods, like for instance a LR(0) approximation, which provides the basis for most practical parsing methods. A single *inadequate* state with items $C \to c\bullet$ and $D \to c\bullet$ can be reached after reading both prefixes Ac and Bc. After

parsing stack	input stack	actions
q_0	$acca\$$	s_4
q_0aq_4	$cca\$$	s_8
$q_0aq_4cq_8$	$ca\$$	s_8
$q_0aq_4cq_8cq_8$	$a\$$	r_8'0

We have reached the first phrase in $acca\$$ that we can resolve with a completely reduced lookahead. This lookahead is a, and indeed it cannot be reduced any further in the rule $S \to ACa$. The lookahead allows the decision of resolving $C \to c$. The newly reduced nonterminal is pushed on the input stack, as usual in noncanonical parsing.

$q_0aq_4cq_8$	$Ca\$$	s_{12}
$q_0aq_4cq_8Cq_{12}$	$a\$$	r_9'1

We have here a non-null pushback: the resolve action r_9'1, which would have needed an unbounded terminal lookahead, is solved using the stacked C and the lookahead a. The pushback of length 1 emulates a reduced lookahead inspection of length 2.

q_0aq_4	$DCa\$$	r_5'0
q_0	$ADCa\$$	s_2
q_0Aq_2	$DCa\$$	s_7
$q_0Aq_2Dq_7$	$Ca\$$	r_4'0
q_0	$AC\$$	s_2
q_0Aq_2	$Ca\$$	s_6
$q_0Aq_2Cq_6$	$a\$$	s_{11}
$q_0Aq_2Cq_6aq_{11}$	$\$$	r_2'0
q_0	$S\$$	s_1
q_0Sq_1	$\$$	r_1'0, accept

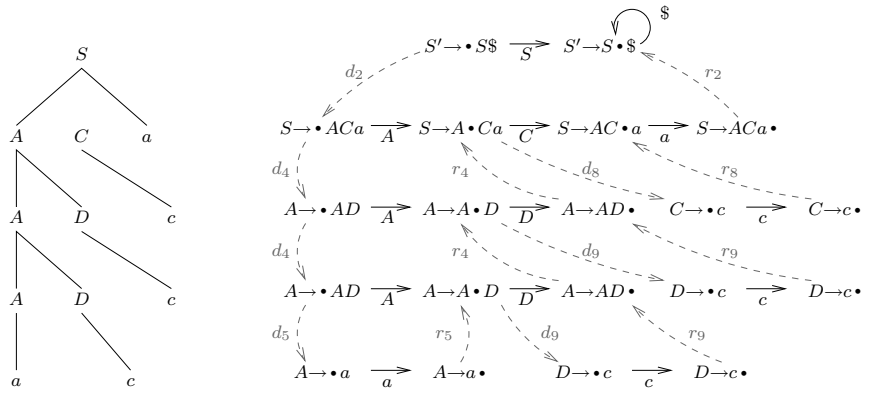
Table 2: The parse of the string $acca$ by the shift-resolve parser for $\mathcal{G}_1$.

reading Ac, the lookahead for the reduction to C is a, while the one for the reduction to D is c^+a. After reading Bc, the lookaheads are c^+b and b respectively. Thus, if we use a LR(0) approximation, we need an unbounded terminal lookahead length in order to choose between the reduction to C or D, when seeing the last input symbol a or b after a sequence c^+.

Grammar $\mathcal{G}_1$ is not LALR(1). If we try to use more advanced parsers, $\mathcal{G}_1$ is not NSLR(1) [15]—it is NSLR(2)—, and the time complexity of XLR(∞) parsing [2] —LR-Regular using a LR(0) approximation—according to $\mathcal{G}_1$ is quadratic.

Table 1 contains the parse table for shift-resolve parsing according to $\mathcal{G}_1$. The table is quite similar to a LR(1) table, with the additional pushback length information, but describes a parser with much more lookahead information. States are denoted by q_i; shift entries are denoted as s_i where i is the new state of the parser; resolve entries are denoted as r_i'j where i is the number of the rule for the reduction and j the pushback length. The reduction according to rule $S' \xrightarrow{1} S$ indicates that the input is successfully parsed. Table 2 details the parsing steps on the valid input $acca$. Symbols are interleaved with states in the parsing stack in order to ease the reading, and are not actually used.

The originality of shift-resolve parsing resides in that Table 1 is not the result of a very precise computation; in fact, we used the worst approximation we tolerate. Still, the parsing time is linear and no preset lookahead length was



(a) Derivation tree (b) Position graph

Figure 1: Representing the derivation of string *accca* in $\mathcal{G}_1$.

necessary.

3 Grammatical Representation

The shift-resolve parsing table presented in Table 1 is the result of a two-steps process: the first step builds a finite nondeterministic automaton from the grammar, and the second generates the deterministic shift-resolve parser from it.

3.1 Position Graph

We consider here a graph representation of a context-free grammar. This graph can be seen as the set of all left to right walks in all possible derivation trees for the grammar. The nodes of this graph are *positions* to the immediate left or immediate right of a derivation tree node. The vertices tell which other positions are reachable. We label each position with a dotted rule giving its local context.

For instance, with $\mathcal{G}_1$, any tree node ν_A with symbol A can nondeterministically derive a node ν_a with symbol a from $A{\rightarrow}a$, or two nodes ν'_A and ν_D with symbols A and D from $A{\rightarrow}AD$. Following this idea, we find that the local context of ν'_A provides us with more information than the mere symbol A: the symbol A in question is in front of a dot in the position $A{\rightarrow}\bullet AD$.

If we make this local context explicit in the labels of the positions, then the relations between these positions become visible. These transitions are of three types: symbol transitions $\xrightarrow{X}$, and a two kinds of ε-transitions: derivation transitions $\xrightarrow{d_i}$ and reduction transitions $\xrightarrow{r_i}$ where i is a rule number. Figure 1 presents the portion deriving *accca* of the position graph for $\mathcal{G}_1$, along with the traditional derivation tree representation. We emulate an infinite number of end of file markers with a looping transition $\xrightarrow{\$}$.

We introduce the *bracketed grammar* $b(\mathcal{G})$ of a context-free grammar $\mathcal{G}$ as the grammar with rules $A \xrightarrow{i} d_i \alpha r_i$ whenever $A \xrightarrow{i} \alpha$ is a rule of $\mathcal{G}$. We also define a homomorphism h that removes all the d_i and r_i symbols from a string. An immediate consequence is that $\mathcal{L}(\mathcal{G}) = h(\mathcal{L}(b(\mathcal{G})))$.

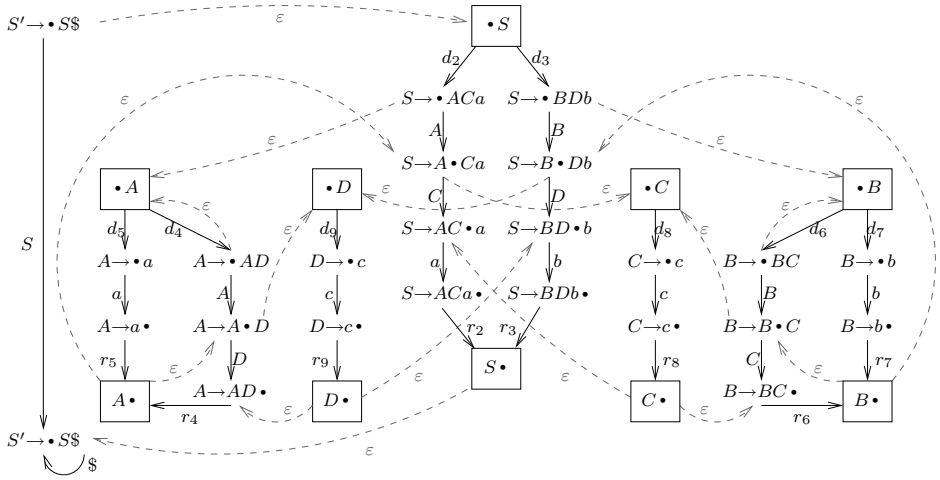


Figure 2: Nondeterministic automaton for Grammar $\mathcal{G}_1$ using κ_0.

In order to uniquely identify a single position in the position graph, we define *valid positions* for a grammar $\mathcal{G}$ as triples $\delta d_i[A \xrightarrow{i} \alpha \bullet \alpha']r_i y$ such that

$$S \underset{\text{rm}}{\Rightarrow}^* \delta A y \underset{\text{rm}}{\Rightarrow} \delta d_i \alpha \alpha' r_i y \text{ in } b(\mathcal{G}). \tag{1}$$

For instance, the position labeled by $C \to \bullet c$ in Figure 1b is identified by the expression $d_2 A d_8[C \xrightarrow{8} \bullet c]r_8 a r_2$.

Definition 1 *The* position graph $\mathcal{I} = \langle \mathcal{N}, \longmapsto \rangle$ *of grammar* $\mathcal{G}$ *associates the (potentially infinite) set* $\mathcal{N}$ *of valid positions for* $\mathcal{G}$ *with the labeled relation* $\longmapsto$ *defined by*

$$\delta[A \xrightarrow{i} \alpha \bullet X \alpha']y \overset{X}{\longmapsto} \delta[A \xrightarrow{i} \alpha X \bullet \alpha']y, \tag{2}$$

$$\delta[A \xrightarrow{i} \alpha \bullet B \alpha']y \overset{d_j}{\longmapsto} \delta \alpha d_j[B \xrightarrow{j} \bullet \beta]r_j u y \qquad \text{if } \alpha' \Rightarrow^* u \text{ in } b(\mathcal{G}), \text{ and} \tag{3}$$

$$\delta \alpha d_j[B \xrightarrow{j} \beta \bullet]r_j u y \overset{r_j}{\longmapsto} \delta[A \xrightarrow{i} \alpha B \bullet \alpha']y \qquad \text{if } \alpha' \Rightarrow^* u \text{ in } b(\mathcal{G}). \tag{4}$$

3.2 Position Equivalences

We are eager to put explicit labels on our positions because we intend to collapse the position graph into a finite graph. The equivalence relations defined to this end will preserve the local context, and thus use the position labels.

Definition 2 *The* collapsed position graph $\Gamma_\kappa = \langle [\mathcal{N}]_\kappa, \longmapsto_\kappa \rangle$ *of a* position graph $\mathcal{I} = \langle \mathcal{N}, \longmapsto \rangle$ *associates* $[\mathcal{N}]_\kappa$ *the finite set of equivalence classes* $[p]_\kappa$ *over* $\mathcal{N}$ *modulo* κ *with the labeled relation* $\longmapsto_\kappa$ *defined by*

$$[p]_\kappa \overset{\chi}{\longmapsto}_\kappa [q]_\kappa \text{ iff } \exists p' \in [p]_\kappa, q' \in [q]_\kappa, p' \overset{\chi}{\longmapsto} q'. \tag{5}$$

Simple Equivalence Relation Figure 2 is not a Rorschach test but the collapsed position graph Γ_{κ_0} for $\mathcal{G}_1$ using a simple equivalence relation κ_0 between positions.

Definition 3 *Two positions are* simply equivalent *if and only if they have the same dotted rule as label, i.e.*

$$\delta[A{\to}\alpha\bullet\alpha']y \;\kappa_0\; \gamma[B{\to}\beta\bullet\beta']z \text{ iff } A{\to}\alpha\bullet\alpha' = B{\to}\beta\bullet\beta'. \tag{6}$$

While very basic, this equivalence relation is fine enough to yield a working shift-resolve parser for $\mathcal{G}_1$. It is the simplest equivalence relation we will use for shift-resolve parsing.

Base Positions The addition of *base positions* to $\mathcal{N}$ as in Figure 2 is straightforward. We identify them as $\delta[\bullet A]y$ or $\delta[A\bullet]y$ whenever $S\underset{\text{rm}}{\Rightarrow}{}^*\delta Ay$ in $\mathcal{G}$; Equations (3) and (4) are then replaced by

$$\delta[A\xrightarrow{i}\alpha\bullet B\alpha']y \dashrightarrow^{\varepsilon} \delta\alpha[\bullet B]uy \qquad \text{if } \alpha'\Rightarrow^* u \text{ in } b(\mathcal{G}), \tag{7}$$

$$\delta\alpha[\bullet B]uy \dashrightarrow^{d_j} \delta\alpha d_j[B\xrightarrow{j}\bullet\beta]r_j uy, \tag{8}$$

$$\delta\alpha d_j[B\xrightarrow{j}\beta\bullet]r_j uy \dashrightarrow^{r_j} \delta\alpha[B\bullet]uy, \text{ and} \tag{9}$$

$$\delta\alpha[B\bullet]uy \dashrightarrow^{\varepsilon} \delta[A\xrightarrow{i}\alpha B\bullet\alpha']y \qquad \text{if } \alpha'\Rightarrow^* u \text{ in } b(\mathcal{G}). \tag{10}$$

When the number $|N|$ of nonterminals is small compared to the number $|P|$ of rules of $\mathcal{G}$, as quite often in practical grammars, base positions allow to significantly diminish the size of Γ_{κ_0}.

Lattice of Equivalence Relations The usual partial order on $\mathrm{Eq}(\mathcal{N})$—the complete lattice of all equivalence relations on $\mathcal{N}$—is the inclusion relation $\subseteq$. For any two elements κ_a and κ_b of $\mathrm{Eq}(\mathcal{N})$, $\kappa_a \wedge \kappa_b$ is the greatest lower bound or *meet*, defined as

$$\kappa_a \wedge \kappa_b = \kappa_a \cap \kappa_b. \tag{11}$$

Finer equivalence relations are obtained when using the meet of two equivalence relations; they result in larger collapsed position graphs.

Let $\mathcal{K}$ be the set of all equivalence relations that are included in κ_0; $\mathcal{K}$ is an obvious interval sublattice of $\mathrm{Eq}(\mathcal{N})$, ordered by the inclusion relation. We will only make use of equivalence relations in $\mathcal{K}$: if κ is in $\mathcal{K}$, then $\kappa = \kappa_0 \wedge \kappa'$ for some κ' in $\mathrm{Eq}(\mathcal{N})$. Equivalence relations in $\mathcal{K}$ abound: for instance, a relation κ_k with $\mathrm{LR}(k)$ precision could be written as $\kappa_0 \wedge l_k$ with

$$\delta[A{\to}\alpha\bullet\alpha']y \; l_k \; \gamma[B{\to}\beta\bullet\beta']z \text{ iff } k : h(y) = k : h(z); \tag{12}$$

the set of equivalence classes using l_k is $[\mathcal{N}]_{l_k} = T'^k$—the set of different sequences of k terminals. An experimental parser generator with a much finer equivalence relation is currently available from the Internet at the following address: `http://serdis.dis.ulpgc.es/~ii-pl/ftp/dr`.

If $\kappa = \kappa_0 \wedge \kappa'$, we can still optimize the size of Γ_κ with base positions. If p is a position in $\mathcal{N}$, then we only have to identify $[p]_\kappa$ as a pair $([p]_{\kappa_0}, [p]_{\kappa'})$, where $[p]_{\kappa_0}$ can be a base position.

3.3 Nondeterministic Automaton

We call a collapsed position graph Γ_κ using an equivalence relation κ in $\mathcal{K}$ a *nondeterministic automaton*.

Preserving Grammar Derivations Let us denote by $\nu_0 = [\varepsilon[S'\rightarrow\bullet S\$]\varepsilon]_\kappa$ the equivalence class on $\mathcal{N}$ using κ containing $\varepsilon[S'\rightarrow\bullet S\$]\varepsilon$, and by $\nu_1 = [\varepsilon[S'\rightarrow S\bullet\$]\varepsilon]_\kappa$ the one containing $\varepsilon[S'\rightarrow S\bullet\$]\varepsilon$. We also denote by $\xmapsto{\chi}{}^*{}_\kappa$ the transitive reflexive closure of $\longmapsto_\kappa$, labeled with χ the sequence of labels on the individual relations. We show here a simple result: paths in a nondeterministic automaton correspond to derivations in the bracketed grammar.

Theorem 1 *If* $S\Rightarrow^*\delta A\rho\Rightarrow\delta r_i\alpha\alpha' d_i\rho = \gamma\alpha' d_i\rho = \gamma\sigma$ *holds in* $b(\mathcal{G})$*, then* $\nu_0\xmapsto{\gamma}{}^*_\kappa[\delta r_i[A\xrightarrow{i}\alpha\bullet\alpha']d_i x]_\kappa\xmapsto{\sigma}{}^*_\kappa\nu_1$ *with* $\rho\Rightarrow^* x$ *holds in* Γ_κ.
Proof. A straightforward induction on the length of γ. □

Size of the Nondeterministic Automaton The index of κ_0 is $|\mathcal{G}|$, thus, the size of $\Gamma_{\kappa_0\wedge\kappa'}$ is in the worst case $O(|[\mathcal{N}]_{\kappa'}|.|\mathcal{G}|)$ where $|[\mathcal{N}]_{\kappa'}|$ is the index of κ'.

4 Shift-Resolve Parsers

4.1 Shift-Resolve Parser Construction

We now describe how to extract a deterministic shift-resolve parser from a nondeterministic automaton Γ_κ. The algorithm is based on a subset construction.

States of the Shift-Resolve Parser The states of the shift-resolve parser are sets of *items* $[\nu, sr, d]$, where ν is an equivalence class on $\mathcal{N}$ using κ—*i.e.* a state in Γ_κ—, sr a parsing action—either a production number or 0 to code a shift—, and d is a nonnegative integer to code the distance to the resolution point. By convention, we assume that d is null whenever sr denotes a shift.

Initial state's item set is computed as $I_{q_0} = \mathcal{C}(\{[\nu_0, 0, 0]\})$, where the closure $\mathcal{C}$ of an item set I is the minimal set such that

$$\begin{aligned}\mathcal{C}(I) = I \cup\\ &\{[\nu', 0, 0] \mid [\nu, sr, d] \in \mathcal{C}(I), \nu \xmapsto{d_i}_\kappa \nu'\} \cup\\ &\{\iota \mid [\nu, sr, d] \in \mathcal{C}(I), \nu \xmapsto{r_i}_\kappa \nu', \neg(null(i) \text{ and } null(I)),\\ &\quad ((sr = 0 \text{ and } \iota = [\nu', i, 0]) \text{ or } (sr \neq 0 \text{ and } \iota = [\nu', sr, d]))\},\end{aligned}$$

where, by noting $\mathcal{L}$ the terminal language produced by a sequence of symbols, we discard superfluous ε-reductions with the help of the conditions

$$\begin{aligned}null(i) \quad &\text{iff} \quad A\xrightarrow{i}\alpha,\ \mathcal{L}(\alpha) = \{\varepsilon\}\\ null(I) \quad &\text{iff} \quad [[\delta[A\rightarrow\alpha X\bullet\beta]y]_\kappa, sr, d] \in I,\ \mathcal{L}(X) = \{\varepsilon\}.\end{aligned}$$

Transition from state item set I with symbol X is defined as follows.

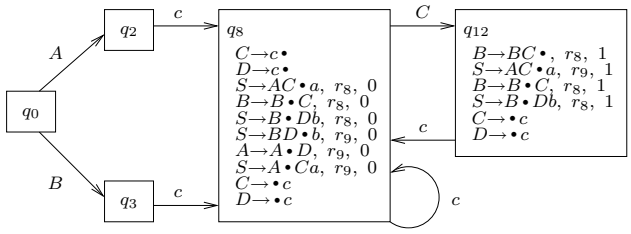


Figure 3: Item sets of some states of the shift-resolve parser for Grammar $\mathcal{G}_1$.

$$\Delta(I, X) = \mathcal{C}(\{[\nu', sr, d'] \mid [\nu, sr, d] \in I, \nu \overset{X}{\longmapsto}_\kappa \nu', \\ ((sr = 0 \text{ and } d' = 0) \text{ or } (sr \neq 0 \text{ and } d' = d + 1))\})$$

Figure 3 presents the details of the item sets computations for states q_8 and q_{12} of the shift-resolve parser presented in Table 1. Lemma 1 states that two positions reachable from ν_0 by the same language will appear in a single item set of the shift resolve parser.

Lemma 1 *Let z be a terminal string in T^*, u and v two strings in $(T \cup \{d_i \mid i \leq |P|\} \cup \{r_i \mid i \leq |P|\})^*$ such that $h(u) = h(v) = z$, $\nu_0 \overset{u}{\longmapsto}{}^*_\kappa \nu$, $\nu_0 \overset{v}{\longmapsto}{}^*_\kappa \nu'$ in Γ_k and $\nu \neq \nu'$.*

Then there exists a cover *string φ in V^* for (u, v) such that $\{[\nu, sr, d], [\nu', sr', d']\}$ is included in the item set $\Delta(I_{q_0}, \varphi)$.*

Proof. We outline the proof by induction on the length $|u| + |v|$.

If $u = v = \varepsilon$, then $z = \varepsilon$ and $\varphi = \varepsilon$ is a cover for $(\varepsilon, \varepsilon)$. There are three possible atomic steps that allow to increase $|u| + |v|$: append a single d_i or r_i to either u or v, or append a terminal symbol a to both.

If $u = u'd_i$ and φ is a cover for (u', v), then φ is also a cover for (u, v). If $u = u'a$, $v = v'a$ and φ is a cover for (u', v'), then φa is clearly a cover for (u, v). If $u = u'r_i$ and φ is a cover for (u', v), then two different cases arise. The first case occurs when $null(i)$, $\varphi = \varphi'\gamma$, $\mathcal{L}(\gamma) = \varepsilon$, $\varphi' = \delta X$ with $\mathcal{L}(X) \neq \varepsilon$ or $\varphi' = \varepsilon$. Then φ' is a cover for (u, v). The second case occurs otherwise, and then φ is clearly a cover for (u, v). □

Parser Table Parser table entries, *i.e.*, shifts and resolves, are computed from the item set I_q of each state q as follows.

$T(q, X) =$
  if $\forall \iota = [[\delta[A \rightarrow \alpha \bullet X\beta]x]_\kappa, sr, d] \in I_q$,
    $sr = r$: **resolve** r **with pushback** d (if $r = 1$ and $d = 0$, **accept**)
    otherwise: **shift** to q' such that $I_{q'} = \Delta(I_q, X)$

4.2 Shift-Resolve Grammars

Rejection Condition A grammar is *inadequate* if and only if two different state item sets are built with identical item sets except for some pushback

length(s); otherwise, we write it is a ShRe(κ) grammar.

It follows that the worst-case space complexity of the shift-reduce parser for $\mathcal{G}$ is $O(2^{|\Gamma_\kappa||P|})$. More powerful shift-resolve parsers can be obtained at the price of descriptional complexity if we add to the condition that one such state should be Δ-reachable from the other.

Theorem 2 *If $\mathcal{G}$ is ambiguous, then it is not ShRe(κ) for any κ in $\mathcal{K}$.*

Proof. We merely outline the proof.

Since $\mathcal{G}$ is an ambiguous context-free grammar, we can find two leftmost derivations $S \underset{\text{lm}}{\Rightarrow}^* xA\rho \underset{\text{lm}}{\Rightarrow} x\alpha\alpha'\rho = x\sigma$ and $S \underset{\text{lm}}{\Rightarrow}^* yB\sigma \underset{\text{lm}}{\Rightarrow} y\beta\sigma$ in $\mathcal{G}$, with $A\alpha\alpha' \neq B\beta$, and such that there is a z in T^* with $x\alpha \Rightarrow^* z$ and $y\beta \Rightarrow^* z$.

Such derivations are mirrored in the nondeterministic automaton Γ_κ by two positions $\nu = [\delta[A \to \alpha \bullet \alpha']s]_\kappa$ and $\nu' = [\gamma[B \to \beta \bullet]t]_\kappa$ such that $\nu_0 \overset{u}{\longmapsto}{}^*_\kappa \nu \overset{\chi}{\longmapsto}{}^*_\kappa \nu_1$ and $\nu_0 \overset{u'}{\longmapsto}{}^*_\kappa \nu' \overset{\chi'}{\longmapsto}{}^*_\kappa \nu_1$, with

$$h(u) = h(u') = z \tag{13}$$

$$h(\chi) = h(\chi') = \sigma. \tag{14}$$

In such a situation, there is a prefix φ in V^* such that some items $[\nu, sr, d]$ and $[\nu', sr', d']$ are included in the item set of $\Delta(I_{q_0}, \varphi)$. The right context of this shift-resolve parser state is $\sigma\$^*$, an infinite regular language. Since $\nu \neq \nu'$ ($A\alpha\alpha' \neq B\beta$ and $\kappa = \kappa_0 \wedge \kappa'$), we are bound to find two item sets only differing on the pushback lengths, and therefore $\mathcal{G}$ is found inadequate. □

Grammar Classes The classes of ShRe(κ_k)—κ_k is defined as the meet of k_0 and l_k from Equation (12)—grammars are not comparable with the classes of LR(k) grammars. For instance, we can produce a shift-resolve parser for the grammar with rules

$$S \to AC \,|\, BCb,\ A \to d,\ B \to d,\ C \to aCb \,|\, c \tag{$\mathcal{G}_2$}$$

using κ_0, but $\mathcal{G}_2$ is not LR(k) for any value of k—as a matter of fact, it is not LR-Regular either.

Conversely, for $k > 0$, we can put an unbounded number of null nonterminals between a conflict and its resolution. For instance, the grammar with rules

$$S \to Aa \,|\, Bb,\ A \to cAE \,|\, c,\ B \to cBE \,|\, c,\ E \to \varepsilon \tag{$\mathcal{G}_3$}$$

is LR(1) but not ShRe(κ) for any κ: once we reach the a or b symbol allowing to resolve, we would need to pushback an unbounded number of E symbols in order to have the c we intend to reduce on top of the parsing stack.

A simplification we made in the shift-resolve construction makes it possible for a LR(0) to be inadequate using κ_k. This is the case for the grammar with rules

$$S \to Sa \,|\, B,\ A \to a,\ B \to dBA \,|\, b. \tag{$\mathcal{G}_4$}$$

Figure 4 shows how the resolution in a shift-resolve state with a single possible reduction (here $B \to b$) can be tricked into an useless exploration of the right context caused by the κ_k approximations. The issue can be tackled on the

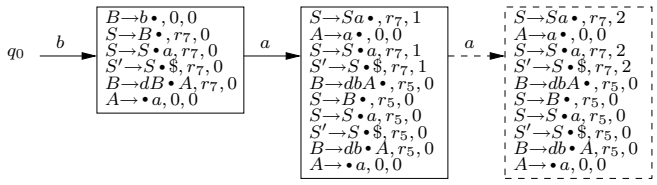


Figure 4: Item sets exhibiting the inadequacy of Grammar $\mathcal{G}_4$ using κ_0.

nondeterministic automaton level by choosing a finer equivalence relation, for instance $\kappa = \kappa_0 \wedge c_1$ where

$$\delta[A \to \alpha \bullet \alpha']y \; c_1 \; \gamma[B \to \beta \bullet \beta']z \text{ iff } h(\delta) : 1 = h(\gamma) : 1. \tag{15}$$

The issue can also be tackled on the subset construction level if we test whether following r_i transitions in the nondeterministic automaton is necessary for a resolution, and if not, fill the entire parser table line with this resolution.

5 Related Work

Shift-resolve parsing is related to two areas: parsing techniques and nondeterministic grammatical representations.

Parsing Techniques The presence of conflicts in deterministic parsers is a widely acknowledged issue. Transforming an input grammar until no more conflicts can be found is a tedious task, can obfuscate the grammar, and may result in convoluted semantic actions. It is therefore tempting for a parser developer to trade the two static guarantees—unambiguity and linear time recognition—for his confidence in his own skill in the handling of ambiguities and a reasonable chance of having a linear time parser [16]. Another line of research is to see how far one can go without sacrificing the static guarantees.

This line has given birth to the LR-Regular [4] and noncanonical [14] parser families. To the best of our knowledge, the only other combination of the two families [7] is an extension to DR(k) parsing [8]. It suffers from a worst-case quadratic parsing time complexity inherent to DR(k) parsing with non LR(k) grammars.

Using only completely reduced symbols in noncanonical parsing was already investigated with the Leftmost SLR(1) parsers [15], and discarded as less powerful than Noncanonical SLR(1) parsing. We improve on LSLR(1) parsers by allowing a non-predefined lookahead length and more powerful approximations in our grammatical representations.

Finally, to the extent of our knowledge, Grammar $\mathcal{G}_1$ is the first published instance of a quadratic parsing time complexity with a regular lookahead parser.

Nondeterministic Grammatical Representations Before becoming a classical presentation [9] and a classical implementation [5] for LR(k) parser constructions, nondeterministic grammatical representations were used for efficient LR(k) testing [11]. Item grammars are a very similar representation [10]. They

have also been used as a unifying framework for parsing methods [13]. Our idea of using the lattice of equivalence relations for the various possible approximations seems to be new.

6 Conclusion

Shift-resolve parsing is a novel parsing method with an attractive combination of properties: the produced parsers are deterministic, they can use an unbounded lookahead, and they run in linear time. Their generation is the result of a highly generic algorithm working on a nondeterministic automaton. It is easy to design new approximations for the automaton in order to improve the grammatical coverage.

The next logical step is the investigation of which conditions would yield shift-resolve parsers that keep running in linear time even if we allow unbounded pushback lengths.

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.

[2] Theodore P. Baker. Extending lookahead for LR parsers. *Journal of Computer and System Sciences*, 22(2):243–259, 1981. ISSN 0022-0000. doi: 10.1016/0022-0000(81)90030-1.

[3] 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.

[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. ISSN 0022-0000.

[5] Charles Donnely and Richard Stallman. *Bison version 2.1*, September 2005. URL `http://www.gnu.org/software/bison/manual/`.

[6] 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`.

[7] 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.

[8] José Fortes Gálvez. *A Discriminating Reverse Approach to LR(k) Parsing.* PhD thesis, Universidad de Las Palmas de Gran Canaria and Université de Nice-Sophia Antipolis, 1998.

[9] Dick Grune and Ceriel J. H. Jacobs. *Parsing Techniques: A Practical Guide*. Ellis Horwood Limited, 1990. ISBN 0-13-651431-6. URL `http://www.cs.vu.nl/~dick/PTAPG.html`.

[10] Stephan Heilbrunner. A parsing automata approach to LR theory. *Theoretical Computer Science*, 15(2):117–157, 1981. ISSN 0304-3975. doi: 10.1016/0304-3975(81)90067-0.

[11] Harry B. Hunt III, Thomas G. Szymanski, and Jeffrey D. Ullman. On the complexity of LR(k) testing. *Communications of the ACM*, 18(12):707–716, 1975. ISSN 0001-0782. doi: 10.1145/361227.361232.

[12] Sylvain Schmitz. Noncanonical LALR(1) parsing. In Zhe Dang and Oscar H. Ibarra, editors, *DLT'06*, volume 4036 of *Lecture Notes in Computer Science*, pages 95–107. Springer, 2006. ISBN 3-540-35428-X. doi: 10.1007/11779148_10.

[13] Thomas Schöbel-Theuer. Towards a unifying theory of context-free parsing. In G. Pighizzini and P. San Pietro, editors, *ASMICS Workshop on Parsing Theory*, Technical Report 126-1994, pages 89–100. Università di Milano, 1994. URL `http://citeseer.ist.psu.edu/117476.html`.

[14] 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.

[15] 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.

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