Symbolic Lookaheads for Bottom-Up Parsing

Paola Quaglia

University of Trento, Italy
paola.quaglia@unitn.it

Abstract

We present algorithms for the construction of LALR(1) parsing tables, and of LR(1) parsing tables of reduced size. We first define specialized characteristic automata whose states are parametric w.r.t. variables symbolically representing lookahead-sets. The propagation flow of lookaheads is kept in the form of a system of recursive equations, which is resolved to obtain the concrete LALR(1) table. By inspection of the LALR(1) automaton and of its lookahead propagation flow, we decide whether the grammar is LR(1) or not. In the positive case, an LR(1) parsing table of reduced size is computed by refinement of the LALR(1) table.

1998 ACM Subject Classification F.4.2 Grammars and Other Rewriting Systems, F.4.3 Formal Languages

Keywords and phrases LALR(1) grammars, LR(1) grammars, Bottom-up parsing

Digital Object Identifier 10.4230/LIPIcs.MFCS.2016.79

1 Introduction

Various classes of grammars can be parsed bottom-up by applying the same shift/reduce algorithm driven by different parsing tables (e.g., SLR(1) [4], LALR(1) [3], LR(1) [2]). Parsing tables are defined on top of deterministic finite state characteristic automata whose size is crucial to the applied parsing technique: the finer the information encoded by automata, the larger the class of parsed grammars, and the bigger the size of parsing tables.

LR(1)-automata are the richer structures in LR(1) parsing. The number of states of these automata has the striking upper bound $O(2^{n(t+1)})$ in the size of the grammar and in the number of terminal symbols (n and t, resp.) [12]. LALR(1) grammars have been defined as a technical compromise between the abundance of syntactic constructs of the generated languages and the size of the associated parsing tables. All the states of the LR(1)-automaton sharing the same LR(0) projection are collapsed into a single state of the LALR(1)-automaton. So, the size of LALR(1) parsing tables is much smaller than that of the corresponding LR(1) tables, and definitely tractable, as widespread parser generators clearly show [9, 2, 6]. At the same time, though, either debugging an LALR(1) grammar or choosing the appropriate directives for resolving conflicts is made harder by the fact that the user is compelled to reason about the propagation of LR(1) lookaheads modulo the – technical, and not necessarily intuitive - merging of LR(1)-states.

In this setting, and especially for large automata, it can be beneficial having some sort of explicit representation of the lookahead propagation flow among the states of the underlying automaton. We work towards this direction and propose a technique for the construction of LALR(1)-automata which provides a compact encoding of the propagation of lookaheads from one state to the other. The approach is based on the definition of symbolic characteristic automata that use items with two components: an LR(0)-item, and a symbolic lookahead-set. When a new state P is added to the automaton, each kernel LR(0)-item is associated with a variable that is propagated to the closure items of P. All the contributions to the lookahead-sets of the items in P are recorded by equations over the variables owned by the kernel items of the state. The propagation flow of lookaheads is embedded by defining equations for variables. For instance, a plausible equation for the variable x can look like $x \doteq \{b, x'\}$, meaning that x can take the value b and all the values that x' can take. To disclose the actual values of lookahead-sets, we resolve the system of equations. We reduce this problem to a reachability problem on the dependency graph of a propagation relation over equivalence classes of variables.

Once symbolic lookahead-sets are instantiated to ground elements, we construct the LALR(1) parsing table for the given grammar. We prove that the algorithm for the LALR(1) construction is correct. The proof of the assertion is based on two auxiliary results. One of them is the symbolic correspondence between the proposed automata and the merged LR(1)-automata used in the simplest algorithm for the construction of LALR(1) parsing tables. The other intermediate result is the correctness of the actualization of lookahead-sets.

Further, we describe an algorithm for the construction of LR(1) parsing tables. Such an algorithm is itself an example of use of the explicit representation of the lookahead propagation in LALR(1)-automata. The table for the larger class is obtained by refining the LALR(1) table. We look after reduce/reduce conflicts of the LALR(1) table and check whether they can be eliminated by unrolling the relevant LALR(1) mergings of states. This by-need strategy has the further benefit of providing opportunities for the early detection that the grammar at hand is not LR(1).

The rest of the paper is organized as follows. Basic definitions and conventions are introduced in Sec. 2. Sec. 3 presents symbolic automata, and their properties. The construction of LALR(1) parsing tables is the subject of Sec. 4, and the algorithm for the construction of LR(1) parsing tables is sketched in Sec. 5. Sec. 6 concludes this extended abstract.

2 Preliminaries

In this section we will introduce the basic definitions and the conventions which will be used in this manuscript. Some familiarity with the theory of LR(1)-parsing is assumed.

A context-free grammar is a tuple $\mathcal{G} = (V, T, S, \mathcal{P})$ whose elements represent, respectively, the vocabulary, the set of terminal symbols in the vocabulary, the start symbol, and the set of productions. Productions are written $A \to \beta$ where $A \in V \setminus T$, and $\beta \in V^*$. The reflexive and transitive closure of the one-step rightmost derivation relation is denoted by '$\Rightarrow^*$'.

We assume that grammars are reduced, and the following notational conventions are adopted. Members of V are denoted by $Y, Y_0, \ldots$; members of V^* by $\alpha, \beta, \ldots$; members of $(V \setminus T)$ by $A, B, \ldots$; members of T by $a, b, \ldots$; and members of T^* by $w, w_0, \ldots$. The empty string is denoted by ϵ. For every $\alpha \in V^*$, first(α) denotes the set of terminals that begin strings w such that $\alpha \Rightarrow^* w$. Moreover, if $\alpha \Rightarrow^* \epsilon$ then $\epsilon \in$ first(α).

Given any context-free grammar $\mathcal{G}$, parsing is applied to strings followed by the endmarker symbol $\$ \notin V$. Also, the parsing table refers to an augmented version of $\mathcal{G}$, denoted by $\mathcal{G}' = (V', T, S', \mathcal{P}')$ where, for a fresh symbol S', $V' = V \cup \{S'\}$, and $\mathcal{P}' = \mathcal{P} \cup \{S' \to S\}$. An LR(0)-item of $\mathcal{G}'$ is a production of $\mathcal{G}'$ with the marker "$\cdot$" at some position of its body. An LR(1)-item of $\mathcal{G}'$ is a pair consisting of an LR(0)-item of $\mathcal{G}'$ and of a symbol in the set $T \cup \{\$\}$. The LR(0)-item $A \to \alpha \cdot \beta$ is called *kernel item* if either $\alpha \neq \epsilon$ or $A = S'$, *closure item* if it is not kernel, *reducing item* if $\beta = \epsilon$, and *bypassing item* if it is not reducing. The same terminology is naturally extended to the LR(1)-items with first projection $A \to \alpha \cdot \beta$. For a set L of LR(1)-items, prj(L) is the set of LR(0)-items occurring as first components of the elements of L, and kernel(L) is the set of the kernel items of L.

We will call *LR(0)-automata*, and *LR(1)-automata* respectively, the characteristic automata constructed by collecting sets of LR(0)-items, and sets of LR(1)-items respectively. Also, we will call *LRm(1)-automata* the characteristic "merged" LR(1)-automata which are at the basis of the simplest, although inefficient, algorithm for the construction of LALR(1) parsing tables. We assume that LR(1)-automata and LRm(1)-automata are constructed after the methodology fully detailed in [1] and in its previous editions.

Below, we will denote the above mentioned characteristic automata by tuples of the form $(\mathcal{Q}, V, \tau, Q_0, \mathcal{F})$ where $\mathcal{Q}$ is the set of states, V the vocabulary, $\tau : (\mathcal{Q} \times V) \to \mathcal{Q}$ the transition function, $Q_0 \in \mathcal{Q}$ the initial state, and $\mathcal{F} \subseteq \mathcal{Q}$ the set of final states, i.e. of the states containing at least one reducing item. In particular, we will use the tuple $(St_l, V, \tau_l, L_0, F_l)$, called $\mathscr{A}_l$ for short, to stand for the LR(1)-automaton for $\mathcal{G}$. The tuple $(St_m, V, \tau_m, M_0, F_m)$, named $\mathscr{A}_m$, will denote the LRm(1)-automaton for $\mathcal{G}$.

Bottom-up parsing tables are filled in after an algorithm which is shared by various techniques (SLR(k), LALR(k), etc.). *Shift* and *goto* moves are determined after the transition function of the appropriate characteristic automaton. Further, the reducing items in the final states of the automaton, and the lookahead-sets associated with them, are used to set up *reduce* moves. By that, we will call *parsing table* the pair consisting of a characteristic automaton and of the collection of lookahead-sets for the reducing items of its final states.

3 Symbolic characteristic automata

In this section we will present the construction of the symbolic characteristic automaton that is central to further developments. In what follows, the prototypical LALR(1) grammar $\mathcal{G}_1$ with start symbol S_1 and production set $\mathcal{P}_1 = \{S_1 \to L = R \mid R,\ L \to *R \mid \text{id},\ R \to L\}$ [2] will be used as running example.

We let $\mathbb{V}$ be a set of symbols disjoint from $V' \cup \{\$\}$. Elements of $\mathbb{V}$ stand for *variables* and are ranged over by $x, x', \ldots$. We use $\Delta, \Delta', \ldots, \Gamma, \Gamma', \ldots$ to denote subsets of $\mathbb{V} \cup T \cup \{\$\}$. Also, we let $\text{ground}(\Delta) = \Delta \cap (T \cup \{\$\})$, and $\text{var}(\Delta) = \Delta \cap \mathbb{V}$. A *symbolic item* of $\mathcal{G}'$ is a pair of the shape $[A \to \alpha \cdot \beta, \Delta]$, whose second component is called *lookahed-set*. Below, symbolic items will be shortly called items when no confusion may arise. We assume the existence of a function newVar() which returns a fresh symbol of $\mathbb{V}$ at any invocation. This assumption on newVar() induces a strict total order over the generated variables, and we write $x \prec x'$ if the call to newVar() which returns x precedes the invocation of newVar() whose response is x'.

The definitions of kernel, closure, reducing, and bypassing items are extended to symbolic items in the natural way. Also, functions prj(_) and kernel(_) are overloaded to be applied to sets of symbolic items. Function first(_) is extended to arguments of the form $\beta\Delta$ as follows:

$$\text{first}(\beta\Delta) = \begin{cases} \text{first}(\beta) & \text{if } \epsilon \notin \text{first}(\beta) \\ (\text{first}(\beta) \setminus \{\epsilon\}) \cup \Delta & \text{otherwise.} \end{cases}$$

The closure of a set of symbolic items P, written closure(P), is defined as the smallest set of items, with smallest lookahead-sets, that satisfies the following equation:

$$\begin{aligned} \text{closure}(P) = P \cup \{&[B \to \cdot\gamma, \Gamma] \text{ such that} \\ &[A \to \alpha \cdot B\beta, \Delta] \in \text{closure}(P) \text{ and } B \to \gamma \in \mathcal{P} \text{ and first}(\beta\Delta) \subseteq \Gamma\}. \end{aligned}$$

The symbolic characteristic automaton for $\mathcal{G}$ is a tuple $(St_s, V, \tau_s, P_0, F_s, \mathit{Vars}, \mathit{Eqs})$, that we will shortly denote by $\mathscr{A}_s$. The first five elements of the tuple represent the set of states, the vocabulary, the transition function, the initial state, and the set of final states. *Vars* is a set of variables, and *Eqs* is a queue of defining equations of the form $x \doteq \Delta$ for the variables in *Vars*. When the actual ordering of the enqueued equations is irrelevant, we will interpret *Eqs* just as a set.

```
$x_0 \longleftarrow$ newVar(); $Vars \longleftarrow \{x_0\}$; $P_0 \longleftarrow$ closure($\{[S' \to \cdot S, \{x_0\}]\}$);
initialize $Eqs$ to contain the equation $x_0 \doteq \{\$\}$; initialize $St_s$ to contain $P_0$;
set $P_0$ unmarked;
while there is some unmarked state in $St_s$ do
    foreach unmarked state $P$ in $St_s$ do
        foreach grammar symbol $Y$ do
            $tmp \longleftarrow \emptyset$;
            foreach $[A \to \alpha \cdot Y\beta, \Delta]$ in $P$ do
                add $[A \to \alpha Y \cdot \beta, \Delta]$ to $tmp$;
            if $tmp \neq \emptyset$ then
                if prj($tmp$) = prj(kernel($Q$)) for some $Q$ in $St_s$ then /* Refine */
                    foreach pair
                    ($[A \to \alpha Y \cdot \beta, \Gamma] \in$ kernel($Q$) , $[A \to \alpha Y \cdot \beta, \Delta] \in tmp$) do
                        if $\beta = \epsilon$ then
                            update $[A \to \alpha Y \cdot, \Gamma]$ to $[A \to \alpha Y \cdot, \Gamma \cup \Delta]$ in kernel($Q$);
                        else if $\Gamma = \{x\}$ and $(x \doteq \Delta_1) \in Eqs$ then
                            update $(x \doteq \Delta_1)$ to $(x \doteq \Delta_1 \cup \Delta)$ in $Eqs$;
                        $\tau_s(P, Y) \longleftarrow Q$;
                else /* Generate */
                    foreach $[A \to \alpha Y \cdot \beta, \Delta] \in tmp$ such that $\beta \neq \epsilon$ do
                        $x \longleftarrow$ newVar();
                        $Vars \longleftarrow Vars \cup \{x\}$;
                        enqueue $(x \doteq \Delta)$ into $Eqs$;
                        change $[A \to \alpha Y \cdot \beta, \Delta]$ into $[A \to \alpha Y \cdot \beta, \{x\}]$ in $tmp$;
                    $\tau_s(P, Y) \longleftarrow$ closure($tmp$);
                    add $\tau_s$(P , Y ) to $St_s$ as an unmarked state;
        mark state $P$ ;
```

Algorithm 1: Construction of $\mathscr{A}_s$ for $\mathcal{G} = (V, T, S, \mathcal{P})$

The algorithm for the construction of the symbolic characteristic automaton for $\mathcal{G}$ is reported as Alg. 1. The collection of states is initialized to contain the initial state P_0, which is defined as the closure of $\{[S' \to \cdot S, \{x_0\}]\}$ where x_0 is a fresh variable. Correspondingly, *Eqs* is let to contain the equation $x_0 \doteq \{\$\}$. The generation of further states goes together with the incremental definition of the transition function. For every state P already found, and for all the grammar symbols Y such that some bypassing item $[A \to \alpha \cdot Y\beta, \Delta]$ is in P, a temporary set *tmp* is computed. Such a set represents, in the LR(0) sense, the kernel of $\tau_s(P, Y)$, and is used to check – irrespectively of lookahead-sets – whether the target state for the Y-transition from P has already been collected or not.

If the wanted target $\tau_s(P, Y)$ has not been collected yet, then a new state, say P', is created by closing up a set which is derived from *tmp* as follows. Reducing items of *tmp* are left untouched. On the other hand, each bypassing item is given a lookahead-set containing a fresh variable. Also, an equation for each such variable is installed in *Eqs* to record the lookahead-set carried by *tmp* from the corresponding item of P. The closure procedure is then applied to the modified instance of the kernel set *tmp*. This ensures the possible propagation of variables, and hence of symbolic lookaheads, to the closure items of P'. As an example, when we start running the algorithm for $\mathcal{G}_1$, we get $[S_1 \to \cdot L = R, \{x_0\}], [L \to \cdot * R, \{=, x_0\}] \in P_0$. By that, $[S_1 \to L \cdot = R, \{x_1\}] \in \tau_s(P_0, L)$, and $[L \to * \cdot R, \{x_2\}] \in \tau_s(P_0, *)$, with *Eqs* provisionally containing the equations $x_0 \doteq \{\$\}$, $x_1 \doteq \{x_0\}$, and $x_2 \doteq \{=, x_0\}$.

Either the states already generated or the equations which have been installed for them can still undergo refinements. This happens when the collected state P' is recognized, again in the LR(0) sense, as the target of the Y-transition from yet another state, say P''. If this is the case, then both the reducing kernel items of P' and the right-hand sides of the equations installed for the kernel bypassing items of P' are treated as accumulators, to record the contributions coming from the items in P''. No further modification is applied to P', nor a closure procedure invoked. Essentially, multiple incoming edges to the same state bring in multiple contributions, and all of them are encoded either by equations over variables or in the lookahead-sets of reducing kernel items. For instance, upon termination of the construction of the automaton for $\mathcal{G}_1$, *Eqs* is given by $E_1 = \langle x_0 \doteq \{\$\}, x_1 \doteq \{x_0\}, x_2 \doteq \{=, x_0, x_2, x_3\}, x_3 \doteq \{x_1\}\rangle$, where x_3 is the variable installed for the kernel item $[S \to L = \cdot R, \{x_3\}]$ of the state generated as $\tau_s(\tau_s(P_0, L), =)$.

A few basic properties of symbolic characteristic automata follow. An easy consequence of the technique used for the construction of symbolic automata is that, seen as graphs, $\mathscr{A}_s$ and the LR(0)-automaton for $\mathcal{G}$ are isomorphic. Moreover, the items contained in each state of the LR(0)-automaton are just the projections of the symbolic items in the corresponding state of $\mathscr{A}_s$. Lemma 1 below characterizes the rôle of variables in the symbolic construction. The lookahead-set of every bypassing kernel item of every state is a singleton set consisting of a distinct variable. Hence, each variable implicitly identifies a pair consisting of a bypassing kernel item and of the state the item belongs to.

▶ **Lemma 1.** *Let $\mathscr{A}_s$ be the symbolic characteristic automaton for $\mathcal{G}$, and let $P, P' \in St_s$. Also, assume that $[A \to \alpha \cdot Y\beta, \Delta] \in P$ and $[A' \to \alpha' \cdot Y'\beta', \Delta'] \in P'$ are such that $A \to \alpha \cdot Y\beta = A' \to \alpha' \cdot Y'\beta'$ implies $P \neq P'$. Then, for some x, x' with $x \neq x'$, $\Delta = \{x\}$ and $\Delta' = \{x'\}$.*

We observe that, by Lemma 1 and by definition of closure, if the lookahead-sets of the items not in the kernel contain any variable, then such a variable must be one of those installed for the bypassing items in the kernel of the same state.

The next lemma accounts for the properties of the equations for the variables identifying bypassing kernel items. The single equation installed for the initial state P_0 remains unchanged throughout the computation of the whole automaton. Hence, there is at least one equation with a ground right-hand side in *Eqs*. Moreover, every equation $x \doteq \Delta$ installed for the bypassing kernel item of a state $P \neq P_0$ is such that Δ collects contributions from all the predecessors of P. The set Δ can contain x itself, due, e.g., to a self-loop on P in the graph. By construction, however, the first provisional versions of every state P and of the relative equations are generated when processing the possible transitions of a state collected before P. So, even if Δ can contain x, it can never contain x alone.

▶ **Lemma 2.** *Let $\mathscr{A}_s$ be the symbolic characteristic automaton for $\mathcal{G}$.*

1. *If $[S' \to \cdot S, \{x\}] \in P_0$ then the equation for x in Eqs is $x \doteq \{\$\}$.*

2. *Assume that $P \in St_s \setminus \{P_0\}$, and that $[A \to \alpha Y \cdot \beta, \{x\}] \in \mathrm{kernel}(P)$ is a bypassing item. Also, let $x \doteq \Delta$ be the equation for x in Eqs. Then the following holds.*
   - *Let $\mathbb{D} = \{\Delta_i \mid [A \to \alpha \cdot Y\beta, \Delta_i] \in Q_i$ for $Q_i \in St_s$ such that $\tau_s(Q_i, Y) = P\}$. Then $\Delta = \bigcup_{\Delta_i \in \mathbb{D}} \Delta_i$.*
   - *$\Delta \setminus \{x\} \neq \emptyset$. Also, if $\Delta \setminus \{x\} = \{x'\}$ then $x' \prec x$.*

The following lemma, dual to Lemma 2, states the main properties of the lookahead-sets of kernel reducing items.

▶ **Lemma 3.** *Let $\mathscr{A}_s$ be the symbolic characteristic automaton for $\mathcal{G}$. Assume that $P \in St_s$, and that $[A \to \alpha Y \cdot, \Delta] \in \mathrm{kernel}(P)$. Also, let $\mathbb{D} = \{\Delta_i \mid [A \to \alpha \cdot Y, \Delta_i] \in Q_i$ for $Q_i \in St_s$ such that $\tau_s(Q_i, Y) = P\}$. Then $P \neq P_0$, $\Delta \neq \emptyset$, and $\Delta = \bigcup_{\Delta_i \in \mathbb{D}} \Delta_i$.*

4 LALR(1) tables

In this section we will first focus on the symbolic correspondence existing between $\mathscr{A}_s$ and the LRm(1)-automaton for the given grammar $\mathcal{G}$. Then, to make this correspondence concrete, we will show how *Eqs* can be resolved.

A key point for the proof of the symbolic correspondence between $\mathscr{A}_s$ and $\mathscr{A}_m$ is the relation between the states of the symbolic automaton and those of the LR(1)-automaton for $\mathcal{G}$. In order to capture such a relation, an appropriate handle on the propagation flow of ground lookaheads through the defining equations in *Eqs* is needed. To this end, we introduce a notion of reachability which relates ground lookaheads to lookahead-sets via equations. Intuitively, we say that the lookahead l reaches Δ if either $l \in \Delta$ or if l is in the right-hand side of the equation for some x, and x propagates, through a chain of uses and definitions, to a variable belonging to Δ. Consider for instance the system of equations $E_1 = \langle x_0 \doteq \{\$\}, x_1 \doteq \{x_0\}, x_2 \doteq \{=, x_0, x_2, x_3\}, x_3 \doteq \{x_1\}\rangle$ mentioned in Sec. 3. In E_1 the ground lookahead \$ reaches $\{\$\}$ simply because it is contained in the set. Moreover, there is a sequence of hops from right-hand sides to left-hand sides of equations which takes, e.g., \$ to x_3. In fact, \$ is used in the definition of x_0, which is used in the definition of x_1, which is used in the definition of x_3. By that, we conclude that \$ reaches, among the rest, every lookahead-set Δ containing x_3. Def. 4 below captures this intuition.

▶ **Definition 4.** Let $E = \{x_i \doteq \Delta_i\}_i$ be a set of defining equations for the variables in the finite subset $\{x_i\}_i$ of $\mathbb{V}$. Also, let $l \in T \cup \{\$\}$. Then:
- x_j gets x_k in E, written x_j gets$_E$ x_k, iff $x_j, x_k \in \{x_i\}_i$ and $x_k \in \Delta_j$;
- Δ takes l in E, written Δ takes$_E$ l, iff $l \in \Delta$ or $x_j, x_k \in \{x_i\}_i$ exist such that $x_j \in \Delta$ and x_j gets$^*_E$ x_k and $l \in \Delta_k$.

Next, we show in what respect the membership of the LR(1)-item $[A \to \alpha \cdot \beta, l]$ in a state of $\mathscr{A}_l$ is related to the membership of $[A \to \alpha \cdot \beta, \Delta]$, with l taken to Δ in *Eqs*, in a state of $\mathscr{A}_s$. The asymmetry of the statements of Lemma 5 and Lemma 6 below is due to the existence of a one-to-many correspondence between the states of $\mathscr{A}_s$ and those of $\mathscr{A}_l$. In fact, each state of $\mathscr{A}_l$ is simulated by an appropriate cut of *Eqs* and of the lookahead-sets of the reducing items of a state of $\mathscr{A}_s$. On the other hand, however, each state $P \in \mathscr{A}_s$, together with *Eqs*, stands for all the states of $\mathscr{A}_l$ whose projection is the same as the projection of P. So, if $[A \to \alpha \cdot \beta, \Delta] \in P$, then not all the states of the LR(1)-automaton with projection $\mathrm{prj}(P)$ necessarily contain the pairing of $A \to \alpha \cdot \beta$ with each of the lookahead l that are taken to Δ.

The proofs of the lemmata below crucially rely upon the following key issue. Both in $\mathscr{A}_l$ and in $\mathscr{A}_s$, the lookahead \$ is generated by the single kernel item of their initial state. All the other lookaheads show up by the application of the closure procedure, and this depends, in either automata and in corresponding ways, on the projection of the item undergoing closure. Once a lookahead has been generated, it propagates along the paths of the two automata in lock-step fashion.

▶ **Lemma 5.** *Let $\mathscr{A}_l$ and $\mathscr{A}_s$ be the LR(1)-automaton and the symbolic automaton for $\mathcal{G}$, respectively. Then for every $L \in St_l$ there exists $P \in St_s$ such that $\mathrm{prj}(P) = \mathrm{prj}(L)$, and, for every $[A \to \alpha \cdot \beta, l] \in L$, if $[A \to \alpha \cdot \beta, \Delta] \in P$ then Δ takes_{Eqs} l.*

Proof sketch. By construction of $\mathscr{A}_s$ and of $\mathscr{A}_l$, if we walk on both automata a path starting from the initial state and labelled by some γ, we reach states with equal projections. Also, given any $L \in St_l$, there is a unique state $P \in St_s$ such that $\mathrm{prj}(P) = \mathrm{prj}(L)$. Hence, all the lookaheads carried to the items of L are also carried, through the corresponding paths, to the relevant items of that state P. ◀

▶ **Lemma 6.** *Let $\mathscr{A}_s$ and $\mathscr{A}_l$ be the symbolic automaton and the LR(1)-automaton for $\mathcal{G}$, respectively. Also, let $[A \to \alpha \cdot \beta, \Delta] \in P \in St_s$, and let l be such that Δ takes_{Eqs} l. Then there exists $L \in St_l$ such that $\mathrm{prj}(L) = \mathrm{prj}(P)$ and $[A \to \alpha \cdot \beta, l] \in L$.*

Proof sketch. By the assumption that Δ takes_{Eqs} l, there is at least one state $P_g \in St_s$ which contains an item, precisely related to $[A \to \alpha \cdot \beta, \Delta]$, that generates l. If $l = \$$, then $P_g = P_0$ and the generating item is the kernel item of P_0. Otherwise, the generating item has the form $[B \to \cdot\delta, \Gamma]$ and $l \in \Gamma$. In either case, for some γ_1, there is a γ_1-path from P_g to P. The string γ_1 can be traced backwards from P to P_g, and depends both on the chain of hops among variables which takes l to Δ, and on the structure of the items found along the way. Now, let γ be a path from P_0 to P_g. By construction of $\mathscr{A}_l$, the state L reached from L_0 by the path $\gamma\gamma_1$ has the same projection as that of P, and contains the item $[A \to \alpha \cdot \beta, l]$. ◀

The relation between the states of $\mathscr{A}_s$ and of $\mathscr{A}_l$ is at the basis of the symbolic correspondence between $\mathscr{A}_s$ and $\mathscr{A}_m$ which is stated by the following theorem.

▶ **Theorem 7.** *Let $\mathscr{A}_s$ and $\mathscr{A}_m$ be the symbolic automaton and the LRm(1)-automaton for $\mathcal{G}$, respectively. Then the following holds.*

- *For every $P \in St_s$ there exists $M \in St_m$ such that $\mathrm{prj}(M) = \mathrm{prj}(P)$, and, for every Y, if $\tau_s(P, Y) = P'$ then $\tau_m(M, Y) = M'$ with M' such that $\mathrm{prj}(M') = \mathrm{prj}(P')$.*
- *For every $M \in St_m$ there exists $P \in St_s$ such that $\mathrm{prj}(P) = \mathrm{prj}(M)$, and, for every Y, if $\tau_m(M, Y) = M'$ then $\tau_s(P, Y) = P'$ with P' such that $\mathrm{prj}(P') = \mathrm{prj}(M')$.*
- *If $P \in St_s$ and $M \in St_m$ are such that $\mathrm{prj}(P) = \mathrm{prj}(M)$, then $[A \to \alpha \cdot \beta, l] \in M$ iff $[A \to \alpha \cdot \beta, \Delta] \in P$ and Δ takes_E l.*

Proof. The first two assertions are consequences of the construction procedures used to obtain $\mathscr{A}_s$ and $\mathscr{A}_m$, which can both be projected into the LR(0)-automaton for $\mathcal{G}$. The third assertion comes from Lemma 5 and Lemma 6, by construction of $\mathscr{A}_m$ from $\mathscr{A}_l$. ◀

To set up the the LALR(1) parsing table that we want to construct, we still need to compute the actual lookahead-set of reducing items. Suppose that $Vars = \{x_0, \ldots, x_n\}$ and $Eqs = \{x_i \doteq \Delta_i\}_{i=0,\ldots,n}$. Our goal is to compute the set of actual instantiations of $x_0, \ldots, x_n$, hereby called $val(x_0), \ldots, val(x_n)$. By definition of takes_{Eqs}, $val(x_i)$ is given by the union of ground(Δ_i) with $val(x_k)$, for all the variables x_k such that x_i gets$_{Eqs}$ x_k. Hence, we actually look for the solution of a system of recursive equations of the form

$$val(x_i) = \text{ground}(\Delta_i) \cup \bigcup_{x_k \,:\, x_i \text{gets}_{Eqs} x_k} val(x_k)\,.$$

We observe that $D = (2^{T \cup \{\$\}})^{n+1}$ is a cpo with least element, and that $val : D \to D$ is a monotone function. Relying on standard approximation techniques, we can prove that the least solution of the system of recursive equations for $val(x_i)$ is given by

$$val(x_i) = \bigcup_{x_k \,:\, x_i \text{gets}^*_{Eqs} x_k} \text{ground}(\Delta_k)\,.$$

To gain in efficiency, instead of computing the above solution for all the variables in *Vars*, we first partition variables into equivalence classes. This allows us to define a reduced system of equations *REqs* which induces a reachability relation of smaller size over a relevant subset *RVars* of *Vars*. The intuition behind this reduction is that characteristic automata are typically quite sparse, and lookahead propagation is usually preponderant over lookahead generation. So, in general *Eqs* is expected to contain many equations of the shape $x_i \doteq \{x_j\}$. An obvious optimization is computing only one of $val(x_i)$ and $val(x_j)$ and then, by need, copying it into the other.

```
inizialize $RVars$ and $REqs$ to $\emptyset$ ;
while $Eqs$ not empty do
    $x \doteq \Delta \longleftarrow$ dequeue($Eqs$) ;
    if $\Delta \setminus \{x\} = \{x'\}$ then
        $class(x) \longleftarrow class(x')$ ;
    else
        $class(x) \longleftarrow x$ ;
        add $x$ to $RVars$ ;
foreach $x \in RVars$ such that $x \doteq \Delta \in Eqs$ do
    update each $x'$ in $\Delta$ to $class(x')$ ;
    add $x \doteq \Delta \setminus \{x\}$ to $REqs$ ;
```

Algorithm 2: Reduced system of equations *REqs* for the variables in $RVars \subseteq Vars$

The algorithm for the computation of *REqs* is reported as Alg. 2. For every $x \in Vars$ we record the membership of the variable into an equivalence class. The equations in *Eqs* are processed one at a time exploiting the generation order of the variables in *Vars*. We first check whether $\Delta \setminus \{x\} = \{x'\}$. If so, then the equation at hand has either the shape $x \doteq \{x'\}$ or the shape $x \doteq \{x', x\}$. Hence the variable x is reached, in *Eqs*, exactly by the same ground values that reach x', and we let both variables belong to same equivalence class. Moreover, by Lemma 2, if $\Delta \setminus \{x\} = \{x'\}$ then $x' \prec x$. Hence, by the ordering of equation processing, $class(x')$ has already been set when handling the equation for x. Once the set of representative variables *RVars* has been identified, we start populating *REqs*. Suppose $x \in RVars$, and assume that $x \doteq \Delta \in Eqs$. Then the equation installed into *REqs* for x is obtained by updating Δ as follows. First, non-representive variables in var(Δ) are replaced by the corresponding class representative. Second, the possible occurrence of x is removed from the resulting set. This is a further optimization that, as done above, amounts to disregard a possible self-recurrence in the computation of $val(x)$. As an example, the system of equations $E_1 = \langle x_0 \doteq \{\$\}, x_1 \doteq \{x_0\}, x_2 \doteq \{=, x_0, x_2, x_3\}, x_3 \doteq \{x_1\}\rangle$ for grammar $\mathcal{G}_1$ is reduced to $\langle x_0 \doteq \{\$\}, x_2 \doteq \{=, x_0\}\rangle$.

The following theorem is a consequence of the resolution strategy for *Eqs* that we have illustrated so far.

▶ **Theorem 8.** *Let $\mathscr{A}_s$ be the symbolic automaton for $\mathcal{G}$, $x \in Vars = \{x_i\}_i$, and $Eqs = \{x_i \doteq \Delta_i\}_i$. Also, let RVars, REqs, and class(x) be as computed by Alg. 2. Then, for every $x_i \in RVars$, $val(x_i) = \bigcup_{x_k \,:\, x_i \mathrm{gets}^*_{REqs} x_k} \mathrm{ground}(\Delta_k)$, and, for every $x_i \in Vars \setminus RVars$, $val(x_i) = val(class(x_i))$.*

Commenting on the complexity of the resolution of *Eqs*, we notice that Alg. 2 is linear in the size of *Vars*, and that $val(x_i)$ can be efficiently computed by a depth-first search algorithm run on the dependency graph of the relation gets_{REqs}. Briefly, each node x_i of the graph can be initially associated with the value $\mathrm{ground}(\Delta_i)$. Then the graph is visited and the values associated with the farthest nodes are accumulated with the values of the nodes found along the way back to the origin of the path. The visit can be organized in such a way that strongly connected components, if any, are recognized on-the-fly and traversed only once (see, e.g., [13, 8, 5]). By that, the search algorithm is linear in the size of the dependency graph of the relation gets_{REqs}. Referring again to $\mathcal{G}_1$, the computation of the actual lookahead-sets goes through the depth-first visit of the dependency graph

$$\begin{tikzcd} \overset{[\![x_2]\!]}{\bullet} \arrow[r] & \overset{[\![x_0]\!]}{\bullet} \end{tikzcd}$$

where $[\![x_0]\!]$ and $[\![x_2]\!]$ stand for the equivalence classes of x_0 and of x_2, respectively.

The following result, which is a consequence of Theorem 7 and of Theorem 8, concludes the section by concretizing the symbolic correspondence between $\mathscr{A}_s$ and $\mathscr{A}_m$.

▶ **Theorem 9.** *Let $\mathscr{A}_s$ be the symbolic automaton for $\mathcal{G}$. Also, for $P \in F_s$ and $[A \to \beta\cdot, \Delta] \in P$, let*

$$\mathcal{LA}(P, [A \to \beta\cdot, \Delta]) = \mathrm{ground}(\Delta) \cup \bigcup_{x \,:\, x' \in \Delta \text{ and } x = class(x')} val(x)\,.$$

Then the pair consisting of $\mathscr{A}_s$ and of $\{\mathcal{LA}(P, [A \to \beta\cdot, \Delta])\}_{P \in F_s, [A \to \beta\cdot, \Delta] \in P}$ is an LALR(1) parsing table for $\mathcal{G}$.

5 LR(1) tables

Below we will briefly overview the strategy we propose for deciding whether the grammar at hand is LR(1), and for constructing a compact LR(1) parsing table in the positive case.

We adopt an optimistic approach, and build an LR(1) parsing table by appropriately expanding the LALR(1) table. The cases when the LALR(1) table does not contain any conflict, or only contains shift/reduce (s/r) conflicts are equally not relevant for our argument. Indeed, in the first case the grammar is LALR(1), and in the second case it is surely not LR(1). So, we focus on the reduce/reduce (r/r) conflicts of the LALR(1) table. If the grammar at hand is not LR(1), then at least one of these conflicts is a genuine LR(1) conflict. This is the case, e.g., for the r/r conflict of the table for the ambiguous grammar $\mathcal{G}_2$ with start symbol S_2 and productions in the set $\{S_2 \to Ab \mid Bb, A \to a, B \to a\}$. If instead the grammar is LR(1), then the r/r conflicts in the LALR(1) table depend on the fact that the procedure for the construction of $\mathscr{A}_s$ caused the merging of states which would have remained separated in the construction of the LR(1)-automaton. Our goal

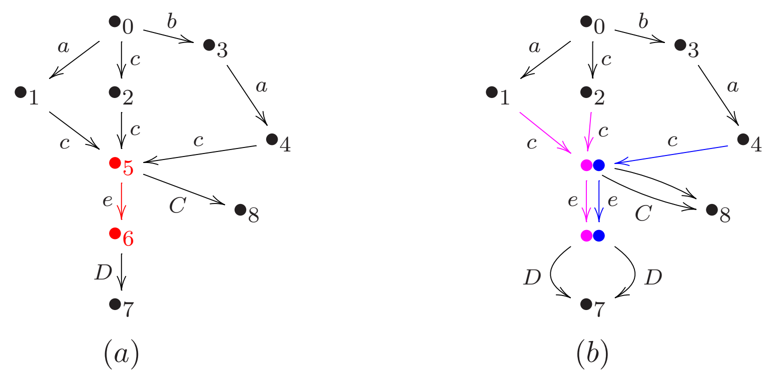

Figure 1 Partial layout of the symbolic automaton for $\mathcal{G}_3$ before (a) and after (b) the splitting due to the r/r conflict at state 6.

is eliminating that sort of r/r conflicts by appropriately splitting the *critical states* that cause those spurious r/r conflicts. Suppose that the symbolic state P is one of such merged states of $\mathscr{A}_s$, and that P stands for the union of the LR(1)-states $L_1, \ldots, L_k$. Also, assume that $P = \{[A_i \to \alpha_i \cdot \beta_i, \Delta_i]\}_i$, and that the instantiation of variables in the lookahead-sets transforms P to $P_{val} = \{[A_i \to \alpha_i \cdot \beta_i, \Sigma_i]\}_i$. Then, there exist k cuts of P_{val} of the form $P_{val_j} = \{[A_i \to \alpha_i \cdot \beta_i, \Sigma_{ij}]\}_i$ where $\bigcup_{j=1,\ldots,k} \Sigma_{ij} = \Sigma_i$, and each P_{val_j} plays the LR(1)-state L_j. Modulo an appropriate tuning of the automaton transition function, a way out for the elimination of the original r/r conflict would be exploding P_{val} in $P_{val_1}, \ldots, P_{val_k}$, and accordingly replicating the subgraph rooted at P_{val}. Two observations are in place here, both related to the space complexity of the resulting structure. Splitting P_{val} in k states might be an overkill, because, e.g., the conflict would be eliminated as well by letting P_{val_1} be still merged with P_{val_2}. For analogous reasons, replicating the whole subgraph rooted at P_{val} can be a waste, too.

We aim at applying the procedure hinted above while limiting the replication of states as much at possible. In the overall, we take the following steps. We start analyzing the r/r conflicts of the LALR(1) table, and check whether the conditions to eliminate them are met or not. If we find any r/r conflict that cannot be eliminated, we infer that the grammar is not LR(1) and conclude. Otherwise, we apply an optimized state splitting procedure to the states involved in the r/r conflicts. Operationally, this is performed by replicating some of the rows of the LALR(1) table, up to minor modifications to their contents. At worst, the resulting table has the same size as the LR(1) table built from the LR(1)-automaton.

A crucial issue in the application of the procedure is the ability to identify critical states. Also, when a state P is recognized as critical, a key point is how we actually split it under the guarantee that the intended LR(1) behaviour is preserved. The lookahead propagation flow embedded by gets_{Eqs} provides the needed support. We describe below the technique for the identification of critical states. The very first step towards this end is locating the states where the lookaheads leading to conflicts, called *critical lookaheads*, are actually generated. Assume we are considering an r/r conflict at state Q for the critical lookahead d. Also, suppose that the symbolic lookahead-sets associated with the conflicting reducing items get instantiated to Σ and to Σ'. There are various combinations here, depending on whether d is a ground element of one or both of the symbolic lookahead-sets, or it is the by-product of variable instantiation. Here we consider this last case which is the most intricate one. An example of this scenario is given by the LR(1) grammar $\mathcal{G}_3$ with start symbol S_3 and production set $\mathcal{P}_3 = \{S_3 \to aAd \mid aBc \mid baAe \mid baBd \mid cAd \mid cBc,\ A \to ce,\ B \to cC,\ C \to eD, D \to \epsilon\}$. The relevant portion of the layout of the symbolic automaton for $\mathcal{G}_3$ is drawn in Fig. 1(a), where 0 is the initial state. The r/r conflict for d is at state 6 and is induced by the reducing items $[A \to ce\cdot, \{x_7\}]$ and $[D \to \cdot, \{x_{11}\}]$ where $x_7 \doteq \{d, e\}$, $x_{11} \doteq \{x_8\}$, and $x_8 \doteq \{c, d\}$ are the relevant equations in Eqs. So, for this specific instance of state Q, $\Sigma = \{d, e\}$ and $\Sigma' = \{c, d\}$. By an analysis of gets^*_{Eqs}, we infer that the lookaheads d and e for $A \to ce\cdot$ come from the kernel item which owns x_7, say i_7, and the lookaheads c and d for $D \to \cdot$ come from the kernel item which owns x_8, say i_8 (both i_7 and i_8 are located at state 5). The items that actually generate the critical lookaheads are located in the predecessors of the state containing i_7 and i_8 (states 1, 2, and 4), and their identity can be inferred by inspection of the projections of i_7 and i_8.

Once the states generating the critical lookaheads have been identified, we check whether the r/r conflict in Q is either genuine or spurious. In the second case, we also decide which is the best possible split of Q. First, we set up two sets of pairs, say $loc(\Sigma)$ and $loc(\Sigma')$, to associate each lookahead in Σ and in Σ' with the state where the lookahead is actually generated. Then, for all the pairs in $loc(\Sigma)$ and $loc(\Sigma')$ that share the same lookahead, we deduce that the associated source states Q_1 and Q_2 are in conflict, written $Q_1 \# Q_2$. E.g., for the running example, $loc(\Sigma) = \{(d, 1), (d, 2), (e, 4)\}$, $loc(\Sigma') = \{(c, 1), (c, 2), (d, 4)\}$, and, by $(d, 1), (d, 2) \in loc(\Sigma)$ and $(d, 4) \in loc(\Sigma')$, we infer $1\#4$, and $2\#4$. The conflict relation $\#$ is the basic tool for deciding whether the r/r conflict at hand is a genuine LR(1) conflict or not. The intuition here is that if the r/r conflict at state Q is spurious, then, as discussed above, it must be possible finding cuts of Q that match the lookaheads contributed by the various merged states. Operationally, we check whether the source states occurring in $loc(\Sigma)$ and in $loc(\Sigma')$ can be partitioned in at least two groups of maximal size so that each group G of the partition meets the following requirements: (i) G contains non-conflicting states; (ii) the restriction of $loc(\Sigma)$ to the pairs whose second component is in G, written $loc(\Sigma)\restriction G$, is non empty; (iii) $loc(\Sigma')\restriction G$ is non empty either. For the example at hand, we end up partitioning $\{1, 2, 4\}$ in the two groups $\{1, 2\}$ and $\{4\}$.

If the above partition cannot be found, then the analyzed r/r conflict in P is a genuine LR(1) conflict, and we conclude that the grammar is not LR(1). This is always the case, e.g., if the conflict relation contains a pair $R\#R$. This conflict reveals that a critical lookahead is generated by distinct items of the state R. Hence this particular lookahead directly depends on the projection of R, which is the same either in $\mathscr{A}_s$ or in $\mathscr{A}_l$. (An instance of this scenario is found in the analysis of the ambiguous grammar $\mathcal{G}_2$ for the critical lookahead b that is generated in the initial state for either $A \to \cdot a$ or $B \to \cdot a$.)

If the source states of the actual lookahead-sets of Q can be partioned into the groups $G_1, \ldots, G_j$, then Q is split in j replica. For each actualized lookahead-set Σ_i, the jth replica of Q is assigned the lookahead-set containing the first elements of the pairs in $\Sigma_i\restriction G_j$. At the same time, relying upon gets^*_{Eqs} we identify the paths $\gamma_1, \ldots, \gamma_m$ that start at the states where the critical lookaheads are sourced and that lead to Q. Such paths share at least one state that is Q at latest. Among the states traversed by $\gamma_1, \ldots, \gamma_m$, those states which are shared by paths from conflicting source states are all critical (states in red in Fig. 1(a)). We split them to grant distinct and parallel routes to the contributions from the states of each group G_j to the corresponding replica of Q. As for the elimination of the spurious r/r conflict at hand, no other state needs to be replicated.

Looking at the splitting procedure from the perspective of the parsing table, the modifications performed are as follows. The row for state Q is copied j times, and each copy retains only the reduction steps for the represented group. The row for each replica of the other critical states is copied from the row of the replicated state. The shift moves of the non-conflicting predecessors of critical states (edges in magenta and in blue in Fig. 1(b)) are redirected to the replica for the appropriate group. Here we notice that each new row of the table has shift moves exactly for the same symbols as the old copy of the row, and fewer reduce moves. So, each pass of the splitting procedure cannot generate neither new s/r conflict nor new r/r conflicts.

The described approach for the construction of LR(1) tables guarantees the early detection that the grammar is not LR(1). In the opposite case, when all the r/r conflicts are eliminated, we get a table for LR(1) parsing where all the states whose merging is not critical are still merged as in an LALR(1) table. The generated table is then expected to be generally smaller than the table constructed on top of LR(1)-automata. An easy example of this is the size of the LR(1) table that we obtain by applying the above algorithm to the grammar $\mathcal{G}_4$ with start symbol S_4 and with production set $\{S_4 \to S_1 \mid S_3\} \cup \mathcal{P}_1 \cup \mathcal{P}_3$. The symbolic automaton for $\mathcal{G}_4$ has two separated sub-graphs corresponding to the symbolic automata for $\mathcal{G}_1$ and for $\mathcal{G}_3$, respectively. The subgraph representing the symbolic automaton for $\mathcal{G}_3$ is refined as described above. The sub-graph for $\mathcal{G}_1$, though, remains untouched, and hence definitely smaller than the LR(1)-automaton for $\mathcal{G}_1$.

6 Concluding remarks

We defined symbolic characteristic automata, and used them as the basis for the construction of LALR(1) parsing tables, for the construction of LR(1) parsing tables, and for early detecting that grammars are non LR(1).

Among the algorithms for the construction of LALR(1) parsing tables, the most popular ones are the `Yacc` algorithm [9, 2], and the algorithm by DeRemer and Pennello [5]. The algorithm we proposed is more similar to the `Yacc` algorithm than to the algorithm by DeRemer and Pennello. In fact, our technique retains, although making it symbolic, the `Yacc` strategy of generating/propagating LR(1) lookaheads. The approach taken by DeRemer and Pennello is instead a refinement of the SLR(1) technique. In a nutshell, the algorithm by DeRemer and Pennello elaborates on the state of the LR(0)-automaton where $A \to \beta\cdot$ is located, and infers which precise subset of the productions of the grammar should be considered when computing the follow-set of that specific occurrence of A.

In [11], Pager presented an algorithm that is used in the implementation of Menhir [7], the parsing engine of OCaml [10]. The algorithm by Pager generates on-the-fly a compact LR(1)-automaton by checking whether already generated states can be the target of the processed transition. We adopt a quite different strategy. Driven by local reasoning on r/r conflicts, we construct LR(1) parsing tables as refinements of the corresponding LALR(1) tables. This delays as much as possible any sort of check on the content of states.

In the algorithms for the construction of parsing tables, the number of set-union operations on lookahead-sets is typically taken as performance measure. Distinct algorithms execute those operations on different kinds of auxiliary structures, and the size of these structures is often grammar-dependent. So, even statistical reasoning about performance, which in many cases is likely as much as we can do, has to be very carefully tuned. Precise comparisons between our algorithms and those mentioned above are untimely at this stage, as they should be based on large test-sets.

The symbolic techniques we presented can be extended to produce parsing tables for grammars in classes bigger than LALR(1). E.g., we used it to define LALR(k) parsing tables for $k > 1$. The major benefit of the symbolic structures we proposed is, however, that equations over variables, together with the fact that each variable uniquely identifies an item, provide a compact and synthetic feedback on the origin of lookaheads, on their flow, and on their inter-dependency. The explicit representation of lookahead propagation can be most useful in the design phase of grammars, i.e. especially when the grammar under investigation is not yet in the wanted class. Orthogonally, the algorithm we presented for upgrading LALR(1) tables to LR(1) tables shows another sort of application of the explicit encoding of lookahead propagation in LALR(1)-automata.

References

1 Alfred V. Aho, Monica S. Lam, Ravi Sethi, and Jeffrey D. Ullman. *Compilers: Principles, Techniques, and Tools (2nd Edition).* Prentice Hall, 2006.

2 Alfred V. Aho and Jeffrey D. Ullman. *Principles of Compiler Design.* Addison-Wesley, 1977.

3 Frank DeRemer. *Practical Translators for LR(k) Languages.* PhD thesis, MIT, Cambridge, Mass., 1969.

4 Frank DeRemer. Simple LR(k) Grammars. *Commun. ACM*, 14(7):453–460, 1971. `doi:10.1145/362619.362625`.

5 Frank DeRemer and Thomas J. Pennello. Efficient Computation of LALR(1) Look-Ahead Sets. *ACM Trans. Program. Lang. Syst.*, 4(4):615–649, 1982. `doi:10.1145/69622.357187`.

6 Charles Donnelly and Richard Stallman. Bison: The Yacc-compatible Parser Generator (Ver. 3.0.4). 2015. URL: `http://www.gnu.org/software/bison/manual/bison.pdf`.

7 François Pottier et Yann Régis-Gianas. Menhir. URL: `http://pauillac.inria.fr/~fpottier/menhir/menhir.html.fr`.

8 J. Eve and Reino Kurki-Suonio. On Computing the Transitive Closure of a Relation. *Acta Inf.*, 8:303–314, 1977. `doi:10.1007/BF00271339`.

9 Stephen C. Johnson. Yacc: Yet Another Compiler-Compiler. Tech. Rep. CSTR 32, Bell Laboratories, Murray Hill, N.J., 1974. URL: `http://dinosaur.compilertools.net/`.

10 Xavier Leroy, Damien Doligez, Alain Frisch, Jacques Garrigue, Didier Rémy, and Jérôme Vouillon. The OCaml system release 4.02. 2014. URL: `http://caml.inria.fr/pub/docs/manual-ocaml/`.

11 David Pager. A Practical General Method for Constructing LR(k) Parsers. *Acta Informatica*, 7:249–268, 1977.

12 Seppo Sippu and Eljas Soisalon-Soininen. *Parsing Theory – Volume II: LR(k) and LL(k) Parsing*, volume 20 of *EATCS Monographs on Theoretical Computer Science.* Springer, 1990. `doi:10.1007/978-3-662-08424-3`.

13 Robert Endre Tarjan. Depth-First Search and Linear Graph Algorithms. *SIAM J. Comput.*, 1(2):146–160, 1972. `doi:10.1137/0201010`.