Formalization and Correctness of Predictive Shift-Reduce Parsers for Graph Grammars based on Hyperedge Replacement[☆]

Frank Drewes[a], Berthold Hoffmann[b], Mark Minas[c]

[a] *Institutionen för datavetenskap, Umeå universitet, SE-901 87 Umeå, Sweden*
[b] *Fachbereich 3—Informatik, Universität Bremen, D-28334 Bremen, Germany*
[c] *Institut für Softwaretechnologie, Fakultät für Informatik,*
Universität der Bundeswehr München D-85577 Neubiberg, Germany

---

Abstract

Hyperedge replacement (HR) grammars can generate NP-complete graph languages, which makes parsing is hard even for fixed HR languages. Therefore, we study predictive shift-reduce (PSR) parsing that yields efficient parsing for a subclass of HR grammars, by generalizing the concepts of SLR(1) string parsing to graphs. We formalize the construction of PSR parsers and show that it is correct. PSR parsers run in linear space and time, and are more efficient than predictive top-down (PTD) parsers recently developed by the authors.

Keywords: hyperedge replacement grammar, graph parsing, grammar analysis

---

1. Introduction

Everywhere in science and beyond, diagrams occur as a means of illustration and explanation. In computer science and engineering, they are also used as primary source of information: they form visual specification languages with a precise syntax and semantics. For instance, the diagrams of the *Uniform Modeling Language* UML specify software artifacts. (See `www.uml.org/`.) When diagram languages shall be processed by computers, techniques of compiler construction have to be transferred to the domain of diagrams. A processor of a textual language parses its syntax, which is specified by a context-free Chomsky grammar, in order to construct an abstract hierarchical representation that can then be further interpreted or translated. The syntax of a diagram language is its structure. To analyze the structure of diagrams, one thus needs grammars to specify their syntax, and parsers for these grammars that perform the analysis. A successfully parsed diagram can eventually be processed further. Since

---

[☆] This paper formalizes the concepts described in [14] and provides detailed correctness proofs for them.

Email addresses: `drewes@cs.umu.se` (Frank Drewes), `hof@informatik.uni-bremen.se` (Berthold Hoffmann), `mark.minas@unibw.de` (Mark Minas)

diagrams can be represented as graphs, their syntax can be captured by graph grammars.

Here we consider hyperedge replacement (HR) graph grammars.[1] Hyperedges are a generalization of edges that may connect any number of nodes, not just two. In a host graph G, the replacement of a hyperedge e by a graph R glues the nodes connected to e to distinguished nodes of R. The context-free case, where the replacement depends just on the label of e (a nonterminal symbol) is well-studied [22]. Unfortunately, it is well known that hyperedge replacement can generate NP-complete graph languages [1]. In other words, even for fixed HR languages parsing is hard. Moreover, even if restrictions are employed that guarantee L to be in P, the degree of the polynomial depends on L; see [27].[2] Only under rather strong restrictions the problem is known to become solvable in cubic time [34, 9].

Since even a cubic algorithm would not scale to diagrams occurring in realistic applications, the authors have recently transferred results of context-free string parsing to graphs: Simple *LL*-parsing (SLL(k) for short, [28]), a top-down parsing method that applies to a subclass of unambiguous context-free string grammars (using k symbols of lookahead), has been lifted to *predictive top-down parsing* of graphs (*PTD* parsing for short, [12]); the program generating PTD parsers approximates Parikh images of auxiliary grammars in order to determine whether a grammar is PTD-parsable [13], and generates parsers that run in quadratic time, and in many cases in linear time.

In this paper, we devise—somewhat complementary—efficient bottom-up parsers for HR grammars, called *predictive shift-reduce (PSR) parsers*, which extend SLR(1) parsers [8], a member of the LR(k) family of deterministic bottom-up parsers for context-free string grammars [26]. We formalize the construction and *modus operandi* of PSR parsers, show their correctness, and relate them briefly to PTD parsers and to SLL(1) and SLR(1) string parsers.

In Sect. 2 we recall basic notions of HR grammars. To support intuition, we briefly recall SLR(1) string parsing in Sect. 3. In Sections 4–9, we work out in detail how it can be lifted to PSR parsing:

Section 4 develops a naïve shift-reduce parser for HR grammars and shows its correctness. This parser is a stack automaton that, one by one, consumes the edges of the input graph and simply "guesses" nondeterministically a backwards application of rules that take the input graph to the start symbol. While this parser is correct, its nondeterminism renders it impractical. One of its disadvantages is that it can run into "dead ends", situations which can never lead to acceptance, regardless of the remaining input.

Section 5 defines a notion of *viable prefixes* and shows that the naïve shift-reduce parser would avoid running into a dead end if and only if one could make

---

[1] Other graph grammars and parsing algorithms are discussed in Sect. 11.

[2] The polynomial algorithm for a restricted class of (fixed) HR grammars presented in [27] was refined in [4] and implemented in the system *Bolinas* for semantic parsing in natural language processing.

sure that its stack does always contain a viable prefix.

Section 6 thus develops a notion of *nondeterministic characteristic finite automaton* (nCFA) and shows that it recognizes (we say *approves*) exactly the viable prefixes. However, since the nCFA is itself nondeterministic, it cannot reasonably be used in order to improve the naïve shift-reduce parser.

Section 7, therefore, shows how the nCFA can be converted into a *deterministic characteristic finite automaton* (dCFA) that is equivalent to the nCFA.

Section 8 incorporates the dCFA into an improved version of the naïve shift-reduce parser. This *dCFA-assisted shift-reduce parser* will thus avoid to run into a dead end. However, it is still nondeterministic as the dCFA may allow for alternative parser actions in a given situation, i.e., there may be *conflicts*.

Finally, Sect. 9 formalizes conflicts, shows how they can be detected, and ends in the definition of *predictive shift-reduce parsers* (PSR parsers) that, for all practical purposes, run in linear time and space.

In Sect. 10, we compare PSR parsing to SLR(1) and PTD parsing wrt. generative power. Related and future work is discussed in Sect. 11.

This paper formalizes the concepts developed in [14] and provides detailed correctness proofs for them. We would also like to mention that the proof of Theorem 1 of [14] turned out to be wrong; see the discussion at the end of Sect. 10.

Acknowledgment. We thank the reviewers for their comments and criticism; we hope to have made good use of them.

2. Hyperedge Replacement Grammars

We let $\mathbb{N}$ denote the non-negative integers. For set A and B, let 2^A denote the powerset of A and $(A \rightharpoonup B)$ the set of all partial functions $A \rightharpoonup B$. The domain of a partial function $f\colon A \rightharpoonup B$ is denoted by $\mathit{dom}(f)$, i.e., $\mathit{dom}(f) = \{a \in A \mid f(a) \text{ is defined}\}$. For $S \subseteq A$, we let $f(S) = \{f(a) \mid a \in S \cap \mathit{dom}(f)\}$. Given two partial functions f and g, we write $f \sqsubseteq g$ if $f \subseteq g$ as binary relations. The composition $g \circ f$ of (possibly partial) functions $f\colon A \rightharpoonup B$ and $g\colon B \rightharpoonup C$ is defined as usual, i.e., $(g \circ f)(a)$ equals $g(f(a))$ if both $f(a)$ and $g(f(a))$ are defined, and is undefined otherwise.

A^* denotes the set of all finite sequences (or strings) over a set A; the empty sequence is denoted by ε and the length of a sequence α by $|\alpha|$. A *stack* $\mathcal{S}$ of elements in A is a nonempty string in A^*. Its *top* is the rightmost element of the string, which is denoted by $\mathit{top}(\mathcal{S})$.

For a (total) function $f\colon A \to B$, its extension $f^*\colon A^* \to B^*$ to sequences is defined by $f^*(a_1 \cdots a_n) = f(a_1) \cdots f(a_n)$, for all $a_1, \ldots, a_n \in A$, $n \geq 0$. Given a relation $\rightsquigarrow \subseteq A \times A$, we denote its n-fold composition with itself by $\rightsquigarrow^n$ (where $\rightsquigarrow^0$ is the identity on A), its transitive closure by $\rightsquigarrow^+$ and its reflexive and transitive closure by $\rightsquigarrow^*$, as usual.

Throughout the paper, we let X denote a global, countably infinite supply of *nodes* or *vertices*.

Definition 2.1 (Graph). An *alphabet* is a set Σ of *symbols* together with an *arity function* $\mathit{arity}\colon \Sigma \to \mathbb{N}$. Given such an alphabet, a *literal* $\boldsymbol{e} = a(x_1, \dots, x_k)$ *over* Σ consists of a symbol $a \in \Sigma$ and $k = \mathit{arity}(a)$ pairwise distinct nodes $x_1, \dots, x_k \in X$. We write $\ell(\boldsymbol{e}) = a$ and denote the set of all literals over Σ by Lit_Σ.

A *graph* $\gamma = \langle V, \varphi \rangle$ *over* Σ consists of a finite set $V \subseteq X$ of nodes and a sequence $\varphi = \boldsymbol{e}_1 \cdots \boldsymbol{e}_n \in \mathit{Lit}_\Sigma^*$ such that all nodes in these literals are in V. $\mathcal{G}_\Sigma$ denotes the set of all graphs over Σ.

We say that two graphs $\gamma = \langle V, \varphi \rangle$ and $\gamma' = \langle V', \varphi' \rangle$ are *equivalent*, written $\gamma \bowtie \gamma'$, if $V = V'$ and φ is a permutation of φ'.

Note that the set of literals of a graph is ordered, i.e., two graphs $\langle V, \varphi \rangle$ and $\langle V', \varphi' \rangle$ with the same set of nodes, but with different sequences of literals are considered to differ, even if $V = V'$ and φ' is just a permutation of φ. However, such graphs are equivalent, denoted by the equivalence relation $\bowtie$. In contrast, "ordinary" graphs would rather be represented using multisets of literals instead of (ordered) sequences. The equivalence classes of graphs, therefore, correspond to conventional graphs. The ordering of literals is technically convenient for the constructions in this paper. However, input graphs to be parsed should of course be considered up to equivalence. Thus, we will make sure that the developed parsers yield identical results on graphs g, g' with $g \bowtie g'$.

For a graph $\gamma = \langle V, \varphi \rangle$, we use the notations $X(\gamma) = V$ and $\mathit{lit}(\gamma) = \varphi$. By $\Sigma(\gamma)$ we denote the set of symbols $a \in \Sigma$ such that γ contains a literal $a(\cdots)$. An injective function $\sigma\colon X \to X$ is called a *renaming*, and γ^σ denotes the graph obtained by replacing all nodes in γ according to σ. We define the "concatenation" of two graphs $\alpha, \beta \in \mathcal{G}_\Sigma$ as $\alpha\beta = (X(\alpha) \cup X(\beta), \mathit{lit}(\alpha)\, \mathit{lit}(\beta))$. A graph γ is a *prefix* of graph α if there is a graph δ such that $\alpha = \gamma\delta$. Thus, a prefix is a particular kind of subgraph. If a graph γ is completely determined by its sequence $\mathit{lit}(\gamma)$ of literals, i.e., if each node in $X(\gamma)$ also occurs in some literal in $\mathit{lit}(\gamma)$, we simply use $\mathit{lit}(\gamma)$ as a shorthand for γ. In particular, a literal $\boldsymbol{e} \in \mathit{Lit}_\Sigma$ is identified with the graph consisting of just this literal and its nodes.

Definition 2.2 (HR Grammar). Let $\Sigma = \mathcal{N} \cup \mathcal{T}$ be an alphabet which is partitioned into disjoint subsets $\mathcal{N}$ and $\mathcal{T}$ of *nonterminals* and *terminals*, respectively. A *hyperedge replacement rule* $r = (\boldsymbol{A} \to \varrho)$ (a *rule* for short) has a literal $\boldsymbol{A} \in \mathit{Lit}_\mathcal{N}$ as its *left-hand side*, and a graph $\varrho \in \mathcal{G}_\Sigma$ with $X(\boldsymbol{A}) \subseteq X(\varrho)$ as its *right-hand side*.

Consider a graph $\gamma = \alpha \boldsymbol{A'} \beta \in \mathcal{G}_\Sigma$ and a rule r as above. A renaming $\mu\colon X \to X$ is a *match* (of r to γ) if $\boldsymbol{A}^\mu = \boldsymbol{A'}$, and if $X(\gamma) \cap X(\varrho^\mu) \subseteq X(\boldsymbol{A}^\mu)$. A match μ of r *derives* γ to the graph $\gamma' = \alpha \varrho^\mu \beta$. This is denoted as $\gamma \Rightarrow_{r,\mu} \gamma'$, or just as $\gamma \Rightarrow_r \gamma'$. We write $\gamma \Rightarrow_\mathcal{R} \gamma'$ if $\gamma \Rightarrow_r \gamma'$ for some rule r taken from a set $\mathcal{R}$ of rules.

A *hyperedge replacement grammar* $\Gamma = (\Sigma, \mathcal{T}, \mathcal{R}, Z)$ (*HR grammar* for short) consists of finite alphabets $\Sigma, \mathcal{T}$ as above (where $\mathcal{N} = \Sigma \setminus \mathcal{T}$), a finite set $\mathcal{R}$ of rules over Σ, and a *start symbol* $Z \in \mathcal{N}$ of arity 0. Γ generates the language

$$\mathcal{L}(\Gamma) = \{ g \in \mathcal{G}_\mathcal{T} \mid Z() \Rightarrow_\mathcal{R}^* g \}$$

of terminal graphs. We call a graph g *valid* with respect to Γ if $\mathcal{L}(\Gamma)$ contains a graph g' with $g \bowtie g'$.

In the following, $\boldsymbol{Z}$ shall denote the literal $Z()$ of the start symbol of Γ. Moreover, we shall generally omit the subscript in $\Rightarrow_{\mathcal{R}}$ and $\Rightarrow^*_{\mathcal{R}}$, thus writing simply $\Rightarrow$ and $\Rightarrow^*$ instead because the HR grammar in question will always be clear from the context.

We call literals in $Lit_{\mathcal{T}}$ terminal literals and denote them as $\boldsymbol{a}, \boldsymbol{b}, \boldsymbol{c}, \dots$, whereas nonterminals literals are literals in $Lit_{\mathcal{N}}$ denoted as $\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{C}, \dots$. Terminal graphs, those in $\mathcal{G}_{\mathcal{T}}$, are denoted as $a, b, c, \dots$, whereas graphs in $\mathcal{G}_{\Sigma}$, i.e., graphs that may contain nonterminal literals, are denoted as $\alpha, \beta, \gamma, \dots$.

The following lemma follows more or less immediately from the definition of derivation steps. Its proof is straightforward by induction over the length of the derivation.

Lemma 2.3. *$\boldsymbol{Z} \underset{\mathrm{rm}}{\Longrightarrow}^* \gamma$ implies $\boldsymbol{Z} \underset{\mathrm{rm}}{\Longrightarrow}^* \gamma^\mu$ for every renaming $\mu\colon X \to X$.*

Throughout the paper, we shall generally restrict our attention to rightmost derivations:

Definition 2.4 (Rightmost Derivation). A derivation step $\gamma \Rightarrow \gamma'$ with $\gamma = \alpha\boldsymbol{A}\beta$ and $\gamma' = \alpha\delta\beta$ is *rightmost* if $\beta \in \mathcal{G}_{\mathcal{T}}$. We write $\gamma \underset{\mathrm{rm}}{\Longrightarrow} \gamma'$. A *rightmost derivation* is a derivation each step of which is rightmost.

Due to context-freeness, the restriction to rightmost derivations does not affect the generated language. The following fact can be shown analoguously to the string case.

Fact 2.5. *For graphs $g \in \mathcal{G}_{\mathcal{T}}$, $\boldsymbol{Z} \Rightarrow^* g$ iff $\boldsymbol{Z} \underset{\mathrm{rm}}{\Longrightarrow}^* g$.*

An immediate consequence of the definition of the derivation relation is the following:

Fact 2.6. *For all graphs $\alpha, \beta, \gamma \in \mathcal{G}_\Sigma$ and $\delta \in \mathcal{G}_{\mathcal{T}}$, $\alpha \underset{\mathrm{rm}}{\Longrightarrow}^* \beta$ implies $\gamma\alpha\delta \underset{\mathrm{rm}}{\Longrightarrow}^* \gamma\beta\delta$ if and only if $X(\beta) \cap X(\gamma\alpha\delta) \subseteq X(\alpha)$.*

It is a well-known result [22, Theorem IV.4.1.2] that every HR grammar can be transformed into an equivalent reduced HR grammar where every nonterminal contributes to its language:

Definition 2.7 (Reduced HR Grammar). A hyperedge replacement grammar $\Gamma = (\Sigma, \mathcal{T}, \mathcal{R}, Z)$ is called *reduced* if $\mathcal{R} = \varnothing$ or, for every literal $\boldsymbol{A} \in Lit_{\mathcal{N}}$,

(i) there is a terminal graph $g \in \mathcal{G}_{\mathcal{T}}$ such that $\boldsymbol{A} \Rightarrow^* g$, and
(ii) there are graphs $\delta, \delta' \in \mathcal{G}_\Sigma$ such that $\boldsymbol{Z} \Rightarrow^* \delta\boldsymbol{A}\delta'$.

Example 2.8 (Semantic Representation). A HR grammar can derive semantic representations of sentences of natural language. The semantic graphs in this example are much simplified Abstract Meaning Representations [3]. As in [15] (where the more powerful concept of contextual hyperedge replacement [10] was used), we represent the semantics of sentences using the predicates (i.e., verbs) 'persuade', 'try', and 'believe'. These yield interesting semantic graphs (to the extent such a small example reasonably can), because 'persuade' is an object control predicate (the patient of the persuasion is the agent of whatever she is persuaded to do) and 'try' is a subject control predicate (the agent of the trying is also the agent of whatever is being tried).

The represented patterns are

- "x *persuades* y to do z"
- "x *tries* to do z"
- "x *believes* y"
- "x *believes* y about z"
- "x *believes* y about himself"

The nodes of the graphs represent (anonymous) persons when they are leaves, and statements otherwise. Predicates are represented by terminal edges with the corresponding label and arity (with a further, first tentacle to the root of the statement governed by the predicate). The rules are as follows:

$$\begin{array}{rcllll} Z() & \to & T(r,x) & & & [s] \\ T(r,x) & \to & \mathit{per}(r,x,y,z)\; T(z,y) & \mid & \mathit{try}(r,x,z)\; T(z,x) & [p,t] \\ & \mid & \mathit{bel}(r,x,y) & \mid & \mathit{bel}(r,x,y)\; T(y,z) & [b_e,b_o] \\ & \mid & \mathit{bel}(r,x,y)\; T(y,x) & & & [b_t] \end{array} \tag{1}$$

A derivation of the AMR graph g representing the phrase "f *persuades* b to *try* to *believe* m" reads as follows:

$$\begin{array}{rcl} Z & \underset{s}{\Rightarrow} & T(r,f) \\ & \underset{p}{\Rightarrow} & \mathit{per}(r,f,b,d)\; T(d,b) \\ & \underset{t}{\Rightarrow} & \mathit{per}(r,f,b,d)\; \mathit{try}(d,b,s)\; T(s,b) \\ & \underset{b_e}{\Rightarrow} & \mathit{per}(r,f,b,d)\; \mathit{try}(d,b,s)\; \mathit{bel}(s,b,m)\,. \end{array} \tag{2}$$

Since g can be derived, so can the graph $g' = \mathit{per}(r,m,f,d)\,\mathit{try}(d,f,s)\,\mathit{bel}(s,f,b)$, i.e., the node names in derivations are irrelevant. Furthermore, while the graph $h = \mathit{try}(d,b,s)\; \mathit{per}(r,f,b,d)\; \mathit{bel}(s,b,m)$ cannot be derived, it is valid for this grammar since $g \bowtie h$. Fig. 1 shows how the rules for T and the graph g are drawn as diagrams, a visually convenient notation that specifies them up to equivalence.

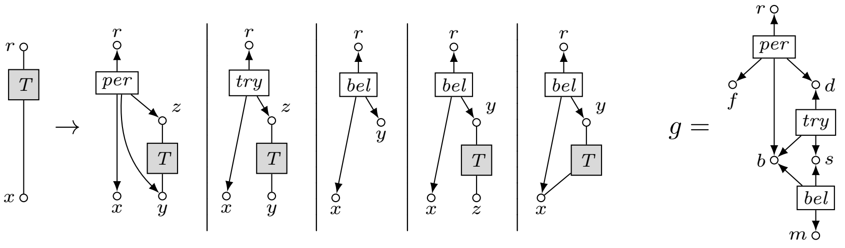

Figure 1: Diagrams of the rules in (1) and of the abstract meaning representation derived in (2). (Circles represent nodes, and boxes represent edges. The box of an edge contains its label, and is connected to the circles of its attached nodes by lines; these lines are ordered counter-clockwise around the edge, starting to its top. Names attached to nodes in rules define the correspondence between left-hand side and right-hand side. Vertical bars separate the right-hand sides of the rules for the nonterminal T.)

3. Shift-Reduce Parsing of Strings

The predictive shift-reduce parser for HR grammars borrows and extends concepts known from the family of context-free LR(k) parsers for context-free string grammars [26], which is why we recall these concepts first. As context-free grammars, shift-reduce parsing, and LR(k) parsing in particular can be found in every textbook on compiler construction, we discuss these matters only by means of a small example.

A Context-Free String Grammar for the Dyck Language. The *Dyck language* of matching nested square brackets "[" and "]" is generated by the context-free string grammar with the nonterminals Z, T, and B, and set of rules

$$\mathcal{D} = \{Z \to T,\ T \to [\,B\,],\ B \to TB,\ B \to \varepsilon\},$$

where Z is the start symbol. An example deriving a string of the Dyck language is

$$Z \underset{0}{\Rightarrow} T \underset{1}{\Rightarrow} [B] \underset{2}{\Rightarrow} [TB] \underset{3}{\Rightarrow} [T] \underset{1}{\Rightarrow} [[B]] \underset{3}{\Rightarrow} [[\,]]\,. \tag{3}$$

The derivation is *rightmost*: every derivation step replaces the rightmost nonterminal of the current string.

A Naïve Shift-Reduce Parser for the Dyck Grammar. A parser checks whether a string like "[[ ]]" belongs to the language of a grammar, and constructs a derivation such as the one in (3) if this is the case. A shift-reduce parser can be formalized as a stack automaton. It reads an input string from left to right and uses its stack for remembering its moves. In a naïve shift-reduce parser, a configuration can be represented as $\alpha \centerdot w$, where α is the stack, consisting of the nonterminal and terminal symbols that have been parsed so far, and w is the consumed part[3] of the input, a terminal string. (As defined in the previous section, the rightmost symbol of α is the top of the stack.) The parser is named after the kind of moves it performs (where α and w are as explained above):

- *Shift* consumes the next input symbol, and pushes it onto the stack. The parser for the Dyck language shifts square brackets:

  $$\alpha \centerdot w \vdash \alpha[\centerdot w[\qquad \alpha \centerdot w \vdash \alpha]\centerdot w]$$

- *Reduce* pops symbols from the stack if they form the right-hand side of a rule, and pushes its left-hand side onto it. Thus, in effect, it applies the rule in reverse. The parser for the Dyck language performs the following reductions:

  $$T \centerdot w \underset{0}{\vdash} Z \centerdot w \quad \alpha[B]\centerdot w \underset{1}{\vdash} \alpha T \centerdot w \quad \alpha T B \centerdot w \underset{2}{\vdash} \alpha B \centerdot w \quad \alpha \centerdot w \underset{3}{\vdash} \alpha B \centerdot w$$

  The parser *accepts* the string w if it reduces the start rule, and its consumed input is w, as in the first reduction.

A successful *parse* of a string w is a sequence of shifts and reductions starting from the *initial configuration* $\varepsilon \centerdot \varepsilon$ to the *accepting configuration* $Z \centerdot w$, as below:

$$\begin{array}{lllllllllll}
\varepsilon \centerdot \varepsilon & \vdash & [\centerdot [& \vdash & [[\underline{\;}\centerdot [[& \underset{3}{\vdash} & [[B \centerdot [[& \vdash & [\underline{[B]}\centerdot [[\,] & \underset{1}{\vdash} & [T\underline{\;}\centerdot [[\,] \\
 & \underset{3}{\vdash} & [\underline{TB}\centerdot [[\,] & \underset{2}{\vdash} & [B \centerdot [[\,] & \vdash & \underline{[B]}\centerdot [[\,] & \underset{1}{\vdash} & \underline{T}\centerdot [[\,]] & \underset{0}{\vdash} & Z \centerdot [[\,]]
\end{array}$$

(The places where reductions apply are underlined.) The reductions of a successful parse, read in reverse, yield a rightmost derivation, in this case the derivation (3) above.

The naïve shift-reduce parser is *correct*, i.e., a string has a successful parse if and only if it has a rightmost derivation.

Nondeterminism. The naïve parser is *nondeterministic*: E.g., in the configuration "$[TB \centerdot [[\,]$" above, the following moves are possible:

(i) a reduction by the rule $B \to T\,B$, leading to the configuration $[B \centerdot [[\,]$;
(ii) a reduction by the rule $B \to \varepsilon$, leading to the configuration $[TBB \centerdot [[\,]$; and
(iii) a shift of the symbol "]", leading to the configuration $[TB] \centerdot [[\,]]$.

Only move (i) will lead to a successful parse, namely the one above. After move (ii) or (iii), further reduction is impossible. In such situations, the parser would

---

[3] The configurations of a shift-reduce parser can also be defined as $\alpha \centerdot u$, where α is the stack, and u is the *unread part* of the input. Then successful parses have the form $\varepsilon \centerdot w \vdash^* Z \centerdot \varepsilon$ (where the definition of shift and reduce moves is adapted in the obvious way). It is easy to show that both definitions are equivalent, i.e., that $\varepsilon \centerdot w \vdash^* Z \centerdot \varepsilon$ if and only if $\varepsilon \centerdot \varepsilon \vdash^* Z \centerdot w$. Here we have chosen configurations that contain the consumed input as this can more easily be lifted to configurations of graph parsers.

have to backtrack, i.e., undo shifts and reductions and try alternative moves, until it finds a successful parse, or fails altogether.

Backtracking makes parsing inefficient. To avoid this, the naïve shift-reduce parser can be refined by gathering information from the grammar that helps to predict which moves lead to successful parses:

- The rules of a grammar allow to predict *viable prefixes*: these are prefixes of sequences of nonterminal and terminal symbols that occur during rightmost derivations of terminal strings. In a successful parse, the stack of the parser does always contain a viable prefix. In cases (i) to (iii) discussed above, the sequences "$[TB$" and "$[B$" are viable prefixes, whereas the sequences "$[TBB$" and "$[TB]$" are not.

- A *lookahead* of the $k > 0$ next input symbols may help to decide which move must be taken to make a parse successful. In the situation sketched above (where a lookahead of $k = 1$ suffices), the reductions (i) and (ii) should only be made if the next input symbol is "]", which is the only terminal symbol that may follow B in derivations with the grammar. Such a symbol is called a *follower symbol* (of the nonterminal B).

Several ways to determine viable prefixes, and different lengths of lookahead can be used to construct predictive shift-reduce parsers. The most general one is Knuth's LR(k) method [26]; here we just consider the simplest case of DeRemer's SLR(k) parser [8], namely for a single symbol of lookahead, i.e., $k = 1$.

Nondeterministic Characteristic Finite-State Automata. The viable prefixes of a context-free grammar form a regular language of nonterminal and terminal symbols that is generated by an automaton, known as *characteristic finite-state automaton* (CFA, for short), which can be derived from the grammar as follows:

- The *states* of the CFA are so-called *items*, rules with an additional dot occurring in the right-hand side. The dot indicates how far parsing has proceeded. For instance, the rule $T \to [\,B\,]$ of the Dyck grammar leads to items $T \to {\centerdot}[\,B\,]$, $T \to [{\centerdot}B\,]$, $T \to [\,B{\centerdot}]$, and $T \to [\,B\,]{\centerdot}$.

- A state like $T \to {\centerdot}[\,B\,]$, where the dot is before some symbol (terminal or nonterminal), has a transition under this symbol to the state where the dot is behind that symbol, here a transition under the terminal "[" to $T \to [{\centerdot}B\,]$.

  A state like $T \to [{\centerdot}B\,]$, with the dot before a nonterminal, does furthermore have spontaneous transitions under the empty string ε to all items for that nonterminal in which the dot is before the first symbol of the right-hand side, e.g. to states $B \to {\centerdot}TB$ and $B \to {\centerdot}\varepsilon$.

Fig. 2 shows the CFA for the Dyck grammar; it is nondeterministic, due to its transitions under the empty string ε. Its *start state* q_0 (distinguished by the ingoing edge without source node and label) represents the situation $Z \to {\centerdot}T$ where nothing has been recognized yet. A path from q_0 to some state q

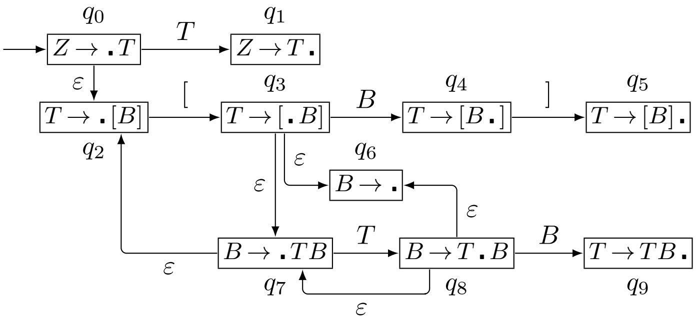

Figure 2: Nondeterministic characteristic finite-state automaton for the Dyck grammar

in the CFA is an alternating sequence of states and labels of the transitions connecting them; the concatenations of the labels along such a path defines a string generated by the CFA. (Note that a path may contain states and labels repeatedly.)

Now a well-known result for shift-reduce parsing reads as follows: a string is generateed by the CFA of a context-free grammar if and only if it is a viable prefix of a successful parse for that grammar. E.g., the viable prefixes "[TB" and "[B" are generated by the CFA, whereas the sequences "[TBB" and "[TB]" are not.

Deterministic Characteristic Finite-State Automata. The nondeterministic CFA of a context-free grammar is easy to define, but less practical for parsing. Fortunately, it can be turned into a deterministic CFA defining the same language (of viable prefixes). The well-known powerset construction works as follows: a state set Q joins some state q with all states q' reachable from q by ε-transitions; q is called a *kernel item* of Q, whereas the q' are called its *closure items.* Then the non-ε-transitions of the items in Q have corresponding transitions to successor state sets Q' that again contain core and closure items. Thus state q_0 of the nondeterministic CFA is joined with state q_2 to form a

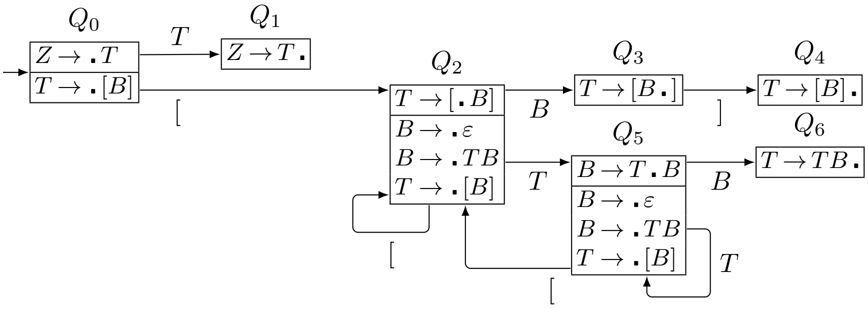

Figure 3: Deterministic characteristic finite-state automaton for the Dyck grammar

state set Q_0, and states q_3 and q_8 are both joined with states q_6, q_7, and q_2 to form state sets Q_2 and Q_5, respectively, while the states q_1, q_4, q_5, and q_9 form singleton state sets Q_1, Q_3, Q_4, and Q_6 of the deterministic CFA. The transition diagram of the deterministic characteristic finite-state automaton for the Dyck grammar is shown in Fig. 3.

The powerset construction may let the number of states explode (2^n state sets for n states of the nondetermistic CFA). However, this rarely occurs in practice; in our example, the number of states does even decrease.

SLR(1) *Parsing.* The stack of the SLR(1) parser is modified to contain a sequence like "$Q_0[Q_2[Q_2TQ_5BQ_6$", recording a path $Q_0 \xrightarrow{[} Q_2 \xrightarrow{[} Q_2 \xrightarrow{T} Q_5 \xrightarrow{B} Q_6$ in its deterministic CFA, starting in its initial state. The moves of the parser are determined by its current (topmost) state, and are modified in comparison to those of the nondeterministic parser as follows:

- *Shift* consumes the next input symbol a if the current state is Q and if the deterministic CFA contains a transition $Q \xrightarrow{a} Q'$. The move pushes a onto the stack, together with the successor state Q'. For our grammar, and $i \in \{0, 2, 4\}$:

$$\alpha Q_i \centerdot w \vdash \alpha q_i \,[\, Q_2 \centerdot w \,[\qquad \alpha Q_3 \centerdot w \vdash \alpha Q_3 \,]\, Q_4 \centerdot w \,]$$

- *Reduce* pops the right-hand side of a rule $A \to \beta$ (and the intermediate states) off the stack, leaving a state Q on top, which has a transition $Q \xrightarrow{A} Q'$. Then A and Q' are pushed onto the stack. The SLR(1) parser performs a reduction only if the lookahead—the nect input symbol—is a follower symbol of A. We write "**if** $\ell = a$" if this is subject to the lookahead symbol a. If $A = Z$, the parser accepts the string. Then a successful parse is as follows:

$$\begin{array}{rcll}
Q_0TQ_1 \centerdot \varepsilon & \underset{0}{\vdash} & Z & \\
\alpha Q_0[Q_2BQ_3]Q_4 \centerdot w & \underset{1}{\vdash} & \alpha Q_0TQ_1 \centerdot w & \\
\alpha Q_2[Q_2BQ_3]Q_4 \centerdot w & \underset{1}{\vdash} & \alpha Q_2TQ_5 \centerdot w & \\
\alpha Q_5[Q_2BQ_3]Q_4 \centerdot w & \underset{1}{\vdash} & \alpha Q_5TQ_5 \centerdot w & \\
\alpha Q_2TQ_5BQ_6 \centerdot w & \underset{2}{\vdash} & \alpha Q_2BQ_3 \centerdot w & \textbf{if } \ell =] \\
\alpha Q_5TQ_5BQ_6 \centerdot w & \underset{2}{\vdash} & \alpha Q_5BQ_6 \centerdot w & \textbf{if } \ell =] \\
\alpha Q_2 \centerdot w & \underset{3}{\vdash} & \alpha Q_2BQ_3 \centerdot w & \textbf{if } \ell =] \\
\alpha Q_5 \centerdot w & \underset{3}{\vdash} & \alpha Q_5BQ_6 \centerdot w & \textbf{if } \ell =]
\end{array}$$

The SLR(1) parser is correct as well: it recognizes the same language as the naïve shift-reduce parser.

Conflicts. The CFA may reveal conflicts for SLR(1) parsing:

- If a state allows to shift some terminal a, and to reduce some rule under the same lookahead symbol a, this is a *shift-reduce conflict*.
- If a state allows reductions of different rules under the same lookahead symbol, this is a *reduce-reduce conflict*.

Whenever the automaton is conflict-free, the SLR(1) parser exists, and can choose its moves in a deterministic way.

The deterministic CFA for the Dyck grammar is indeed conflict-free: In states Q_2 and Q_5, rule $B \to \varepsilon$ can be reduced if the input begins with the only follower symbol "]" of B, which is not in conflict with the shift transitions from these states under the terminal "[".

A deterministic parse with the SLR(1) parser is as follows:

$$\begin{array}{lllll}
Q_0 \,\blacksquare\, \varepsilon & \vdash & Q_0\,[\,Q_2 \,\blacksquare\, [& \vdash & Q_0\,[\,Q_2\,[\,Q_2 \,\blacksquare\, [\,[\\
 & \underset{3}{\vdash} & Q_0\,[\,Q_2\,[\,Q_2 B\,Q_3 \,\blacksquare\, [\,[& \vdash & Q_0\,[\,Q_2\,[\,Q_2 B\,Q_3\,]\,Q_4 \,\blacksquare\, [\,[\,] \\
 & \underset{1}{\vdash} & Q_0\,[\,Q_2 T\,Q_5 \,\blacksquare\, [\,[\,] & \underset{3}{\vdash} & Q_0\,[\,Q_2 T\,Q_5 B\,Q_6 \,\blacksquare\, [\,[\,] \\
 & \underset{2}{\vdash} & Q_0\,[\,Q_2 B\,Q_3 \,\blacksquare\, [\,[\,] & \vdash & Q_0\,[\,Q_2 B\,Q_3\,]\,Q_4 \,\blacksquare\, [\,[\,]\,] \\
 & \underset{1}{\vdash} & Q_0 T\,Q_1 \,\blacksquare\, [\,[\,]\,] & \underset{0}{\vdash} & Z.[\,[\,]\,]
\end{array}$$

Each run of the deterministic parser corresponds to a run of the corresponding naïve shift-reduce parser when we ignore states and just consider the symbols on the stack. Thus the deterministic parser is correct, but it does only apply to grammars that are free of SLR(1) conflicts.

4. A Naïve Shift-Reduce Parser for HR Grammars

We now start to transfer the ideas of shift-reduce string parsing to HR grammars. In this section, we describe a naïve nondeterministic shift-reduce parser, which will be made more practical in the sections to follow. We prove the correctness of the naïve parser, i.e., that it can (nondeterministically) find a derivation for an input graph if and only if there is one.

Assumption 4.1. Throughout the rest of the paper, let $\Gamma = (\Sigma, \mathcal{T}, \mathcal{R}, Z)$ be the HR grammar for which we want to construct a parser. Without loss of generality, we assume that Γ is reduced.

In the remainder of this paper, we will use a HR grammar generating trees as a running example.

Example 4.2 (HR Grammar for Trees). The HR grammar with start symbol Z and the following rules derives n-ary trees.

$$Z \to \mathit{root}(x)\, T(x) \qquad T(y) \to T(y)\, e(y,z)\, T(z) \qquad T(y) \to \varepsilon$$

We shall refer to these rules by the number $1, 2, 3$. Note that the unique edge label *root* designates the unique node where parsing has to start. The empty sequence ε in the last rule is actually a short-hand for the graph $\langle \{y\}, \varepsilon \rangle$ consisting of a single node rather than for the empty graph. Fig. 6 shows a derivation of the tree $t = \mathit{root}(1)\, e(1,3)\, e(1,2)\, e(2,4)$, which is rightmost. The tree $t' = e(2,4)\, \mathit{root}(1)\, e(1,3)\, e(1,2)$ is valid wrt. the grammar since $t' \bowtie t$.

The diagrams of the rules are shown in Fig. 4, and a diagram of the tree t is shown in Fig. 5.

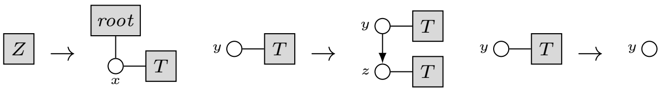

Figure 4: HR rules deriving trees. The binary terminal edge $e(y, z)$ is drawn as an arrow from node x to node y.

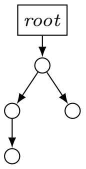

Figure 5: A tree

$$\begin{aligned}
\underline{Z} &\underset{\mathrm{rm}}{\Longrightarrow}_1 \mathit{root}(1)\, \underline{T(1)} \\
&\underset{\mathrm{rm}}{\Longrightarrow}_2 \mathit{root}(1)\, T(1)\, e(1,2)\, \underline{T(2)} \\
&\underset{\mathrm{rm}}{\Longrightarrow}_2 \mathit{root}(1)\, T(1)\, e(1,2)\, T(2)\, e(2,4)\, \underline{T(4)} \\
&\underset{\mathrm{rm}}{\Longrightarrow}_3 \mathit{root}(1)\, T(1)\, e(1,2)\, \underline{T(2)}\, e(2,4) \\
&\underset{\mathrm{rm}}{\Longrightarrow}_3 \mathit{root}(1)\, \underline{T(1)}\, e(1,2)\, e(2,4) \\
&\underset{\mathrm{rm}}{\Longrightarrow}_2 \mathit{root}(1)\, T(1)\, e(1,3)\, \underline{T(3)}\, e(1,2)\, e(2,4) \\
&\underset{\mathrm{rm}}{\Longrightarrow}_3 \mathit{root}(1)\, \underline{T(1)}\, e(1,3)\, e(1,2)\, e(2,4) \\
&\underset{\mathrm{rm}}{\Longrightarrow}_3 \mathit{root}(1)\, e(1,3)\, e(1,2)\, e(2,4)
\end{aligned}$$

Figure 6: A rightmost derivation of a tree

Similarly to the string case, a shift-reduce parser of graphs is modeled by a stack automaton that reads the literals of the input graph in an appropriate order and uses a stack for remembering its actions. A configuration consists of the current stack γ, which is a graph that may contain nonterminals, and the subgraph g of the (terminal) input graph that has been processed already:

Definition 4.3 (Parser Configuration). A *(shift-reduce parser) configuration* $\gamma \centerdot g$ consists of graphs $\gamma \in \mathcal{G}_\Sigma$ and $g \in \mathcal{G}_\mathcal{T}$. The former is the stack whereas the latter is the already consumed subgraph of the input graph.

The parser begins with both the stack prefix already consumed literals being empty, i.e., the *initial configuration* is $\varepsilon \centerdot \varepsilon$. The parser then tries to turn it into an accepting configuration using shift and reduce moves similar to the string case. Shift moves in fact process literals of the input graph, which are then stored in the parser configuration. The parser accepts the input graph if it is able to terminate with a stack consisting of just the start graph $\boldsymbol{Z}$ and g having been processed completely. This situation is represented by an *accepting configuration* $\boldsymbol{Z} \centerdot g'$. We will show in the following that reaching $\boldsymbol{Z} \centerdot g'$ in fact means $\boldsymbol{Z} \Rightarrow^* g' \bowtie g$, i.e., the parser has identified a permutation of the input graph literals which shows that g is valid with respect to the grammar.

Definition 4.4 (Shift and Reduce Steps). A *reduce move* turns a configuration $\gamma \centerdot g$ into $\gamma' \centerdot g$ if there is a graph $\alpha \in \mathcal{G}_\Sigma$, a rule $\boldsymbol{A} \to \varrho$ and a renaming $\mu \colon X \to X$ such that $\gamma = \alpha\varrho^\mu$, $\gamma' = \alpha\boldsymbol{A}^\mu$, and $X(\alpha) \cap X(\varrho^\mu) \subseteq X(\boldsymbol{A}^\mu)$. We write $\alpha\varrho^\mu \centerdot g \vdash_{\overline{\boldsymbol{A}^\mu \Rightarrow \varrho^\mu}} \alpha\boldsymbol{A}^\mu \centerdot g$.

A *shift move* turns a configuration $\gamma \centerdot g$ into $\gamma\boldsymbol{a} \centerdot g\boldsymbol{a}$ for a literal $\boldsymbol{a} \in \mathit{Lit}_{\mathcal{T}}$ if $X(\boldsymbol{a}) \cap X(g) \subseteq X(\gamma)$. We write $\gamma \centerdot g \vdash_{\overline{\mathsf{sh}}} \gamma\boldsymbol{a} \centerdot g\boldsymbol{a}$.

We write $\gamma \centerdot g \vdash\!\!- \gamma' \centerdot g'$ if $\gamma \centerdot g \vdash_{\overline{\mathsf{sh}}} \gamma' \centerdot g'$ or $\gamma \centerdot g \vdash_{\overline{\boldsymbol{B} \Rightarrow \beta}} \gamma' \centerdot g'$ and call $\gamma \centerdot g \vdash\!\!- \gamma' \centerdot g'$ a *move* of the parser.

Let us briefly discuss the difference between these shift and reduce moves on the one hand and their counterparts in string parsing on the other hand.

A shift move in string parsing always reads the first symbol of the remaining input; the string parser cannot choose the symbol to be shifted. The graph parser, in contrast, can pick any of the remaining (terminal) literals for a shift move, as long as the application condition is satisfied. This adds another dimension of nondeterminism to the parsing of graphs.

A reduce move in string parsing replaces the right-hand side of a rule on the stack by its left hand side without further consideration. The graph parser, in contrast, must rename the nodes in the rule first (cf. Def. 2.2). The condition $X(\alpha) \cap X(\varrho^\mu) \subseteq X(\boldsymbol{A}^\mu)$ makes sure that $\gamma' = \alpha\boldsymbol{A}^\mu \Rightarrow \alpha\varrho^\mu = \gamma$, i.e., γ' is derived to γ using rule $\boldsymbol{A} \to \varrho$. A reduce move indeed removes all nodes from the stack that are generated by the derivation step $\gamma' \Rightarrow \gamma$ (when the literals of ϱ^μ are removed.) The application condition $X(\boldsymbol{a}) \cap X(g) \subseteq X(\gamma)$ of shift moves eventually checks that these nodes do not occur in the rest graph when its literals are processed. Note that if a literal of the rest graph violates the condition for a shift move once, it will never satisfy this condition, and will thus never be shifted. Once a condition for a shift move fails, the parse fails altogether.

Example 4.5 (Nondeterministic Shift-Reduce Parser for Trees). The nontederministic shift-reduce parser for the tree grammar of Example 4.2 has the following operations:

- *Shift* operations, for the edges labeled with *root* and e, and
- *Reductions* for the tree-generating rules.

Fig. 7 show the moves of a nondeterministic shift-reduce parser when recognizing the tree t with $t \bowtie \mathit{root}(1)\, e(1,2)\, e(1,3)\, e(2,4)$. In many steps of this parse, the parser has a choice where a “wrong” decision could lead it into a dead end:

1. In the third step, the parser shifts the edge $e(1,2)$; it could have chose $e(1,3)$ instead, which is another match of the edge pattern $e(y,z)$. It is easy to see that the parser could succeed in this case, accepting the graph $g' = \mathit{root}(1)e(1,3)e(1,2)e(2,4)$. However, $g \bowtie g'$. Shifting $e(2,4)$ is also possible, but leads to a dead end.
2. In the fourth step, the parser could shift edge $e(2,4)$ instead of reducing rule 3. This choice of a shift instead of a reduction would lead into a dead end.
3. Instead of reducing $T(2)e(2,4)T(4)$ to $T(2)$ in the seventh step, the parser could reduce rule 3. Reduction of a rule like $T(y) \to \varepsilon$ is possible in every step. Here it would also lead to a dead end.
4. Another choice would have to be made if the grammar is extended by a rule $T(y) \to \mathit{node}^y$ (which is rather ridiculous): In the third step, the parser would then have to choose between a shift of $e(1,2)$ and a shift of $\mathit{node}(1)$.

$$
\begin{array}{lrll}
 & \varepsilon & \centerdot\, \varepsilon & \\
\vdash & \mathit{root}(1)\_ & \centerdot\, \mathit{root}(1) & \\
\underset{3}{\vdash} & \mathit{root}(1)T(1) & \centerdot\, \mathit{root}(1) & y/1 \\
\vdash & \mathit{root}(1)T(1)e(1,2)\_ & \centerdot\, \mathit{root}(1)e(1,2) & \\
\underset{3}{\vdash} & \mathit{root}(1)T(1)e(1,2)T(2) & \centerdot\, \mathit{root}(1)e(1,2) & y/2 \\
\vdash & \mathit{root}(1)T(1)e(1,2)T(2)e(2,4)\_ & \centerdot\, \mathit{root}(1)e(1,2)e(2,4) & \\
\underset{3}{\vdash} & \mathit{root}(1)T(1)e(1,2)\underline{T(2)e(2,4)T(4)} & \centerdot\, \mathit{root}(1)e(1,2)e(2,4) & y/4 \\
\underset{2}{\vdash} & \mathit{root}(1)\underline{T(1)e(1,2)T(2)} & \centerdot\, \mathit{root}(1)e(1,2)e(2,4) & y/2, z/4 \\
\underset{2}{\vdash} & \mathit{root}(1)T(1) & \centerdot\, \mathit{root}(1)e(1,2)e(2,4) & y/1, z/2 \\
\vdash & \mathit{root}(1)T(1)e(1,3)\_ & \centerdot\, \mathit{root}(1)e(1,2)e(2,4)e(1,3) & \\
\underset{3}{\vdash} & \mathit{root}(1)\underline{T(1)e(1,3)T(3)} & \centerdot\, \mathit{root}(1)e(1,2)e(2,4)e(1,3) & y/3 \\
\underset{2}{\vdash} & \underline{\mathit{root}(1)T(1)} & \centerdot\, \mathit{root}(1)e(1,2)e(2,4)e(1,3) & y/1, z/3 \\
\underset{1}{\vdash} & Z & \centerdot\, \mathit{root}(1)e(1,2)e(2,4)e(1,3) & x/1
\end{array}
$$

Figure 7: Moves of the nondeterminstic shift-reduce parser when recognizing the tree in Example 4.2. Places on the stack where reductions occur are underlined. Matches for rules in reductions appear in the rightmost column.

We now show that a parse consisting of shift and reduce moves corresponds to a rightmost derivation and vice versa. We first show that each parse yields a rightmost derivation (Lemma 4.6) and then that each rightmost derivation yields a parse (Lemma 4.8).

Lemma 4.6. *For every sequence* $\gamma \bullet g \vdash^{*} \gamma' \bullet g'$ *of moves with* $X(\gamma) \subseteq X(g)$*, there is a graph* $u \in \mathcal{G}_{\mathcal{T}}$ *such that* $g' = gu$ *and* $\gamma' \mathrel{\underset{\mathrm{rm}}{\Longrightarrow}}^{*} \gamma u$*. Moreover,* $X(\gamma') \subseteq X(g')$*.*

Proof. Let $\gamma \bullet g \vdash^{n} \gamma' \bullet g'$ be any sequence of moves with $X(\gamma) \subseteq X(g)$. We prove that there is a graph $u \in \mathcal{G}_{\mathcal{T}}$ such that $g' = gu$ and $\gamma' \mathrel{\underset{\mathrm{rm}}{\Longrightarrow}}^{*} \gamma u$ by induction over n. The proposition follows for $n = 0$ from $\gamma = \gamma'$, $g = g'$ and $u = \varepsilon$.

For $n > 0$ and the last move being a shift move, the sequence has the form

$$\gamma \bullet g \vdash^{n-1} \gamma'' \bullet g'' \mathrel{\underset{\mathsf{sh}}{\vdash}} \gamma''\boldsymbol{a} \bullet g''\boldsymbol{a} = \gamma' \bullet g'$$

for some $\boldsymbol{a} \in \mathit{Lit}_{\mathcal{T}}$, $g'' \in \mathcal{G}_{\mathcal{T}}$, and $\gamma'' \in \mathcal{G}_{\Sigma}$. Let $u = u'a$. By the induction hypothesis, there is a graph $u' \in \mathcal{G}_{\mathcal{T}}$ such that $g'' = gu'$ and $\gamma'' \mathrel{\underset{\mathrm{rm}}{\Longrightarrow}}^{*} \gamma u'$. Moreover, because of $X(\gamma) \subseteq X(g)$ and by the definition of shift moves, $X(\boldsymbol{a}) \cap X(\gamma u') \subseteq X(\boldsymbol{a}) \cap X(gu') = X(\boldsymbol{a}) \cap X(g'') \subseteq X(\gamma'')$. Therefore, by Fact 2.6, $\gamma' = \gamma''\boldsymbol{a} \mathrel{\underset{\mathrm{rm}}{\Longrightarrow}}^{*} \gamma u'\boldsymbol{a} = \gamma u$ and $g' = g''\boldsymbol{a} = gu'\boldsymbol{a} = gu$.

For $n > 0$ and the last move being a reduce move, the sequence has the form

$$\gamma \bullet g \vdash^{n-1} \alpha\varrho^{\mu} \bullet g' \mathrel{\underset{\boldsymbol{A}^{\mu} \Rightarrow \varrho^{\mu}}{\longmapsto}} \alpha\boldsymbol{A}^{\mu} \bullet g' = \gamma' \bullet g'$$

for a rule $\boldsymbol{A} \to \varrho$ and a renaming $\mu \colon X \to X$. By the induction hypothesis, there is a graph $u \in \mathcal{G}_{\mathcal{T}}$ such that $g' = gu$ and $\alpha\varrho^{\mu} \mathrel{\underset{\mathrm{rm}}{\Longrightarrow}}^{*} \gamma u$. Moreover, by the definition of reduce moves, $X(\alpha) \cap X(\varrho^{\mu}) \subseteq X(\boldsymbol{A}^{\mu})$. Therefore, $\gamma' = \alpha\boldsymbol{A}^{\mu} \mathrel{\underset{\mathrm{rm}}{\Longrightarrow}} \alpha\varrho^{\mu} \mathrel{\underset{\mathrm{rm}}{\Longrightarrow}}^{*} \gamma u$.

Finally, $X(\gamma') \subseteq X(\gamma u) \subseteq X(gu) = X(g')$ as $\gamma' \mathrel{\underset{\mathrm{rm}}{\Longrightarrow}}^{*} \gamma u$ and $X(\gamma) \subseteq X(g)$. □

The following lemma is needed in the proof of Lemma 4.8; it generalizes the condition for applying one shift move to sequences of shift moves:

Lemma 4.7. $X(g) \cap X(u) \subseteq X(\gamma)$ *implies* $\gamma \bullet g \mathrel{\underset{\mathsf{sh}}{\vdash}}^{*} \gamma u \bullet gu$ *for all graphs* $\gamma \in \mathcal{G}_{\Sigma}$ *and* $g, u \in \mathcal{G}_{\mathcal{T}}$*.*

Proof. We prove the proposition by induction over $n = |u|$. The proposition follows for $n = 0$ from $u = \varepsilon$. For $n > 0$, let g, u, γ as in the lemma, $u = \boldsymbol{a}u'$ for some $\boldsymbol{a} \in \mathit{Lit}_{\mathcal{T}}$ and $u' \in \mathcal{G}_{\mathcal{T}}$. Then, $X(g) \cap X(\boldsymbol{a}) \subseteq X(g) \cap X(u) \subseteq X(\gamma)$, and therefore $\gamma \bullet g \mathrel{\underset{\mathsf{sh}}{\vdash}} \gamma\boldsymbol{a} \bullet g\boldsymbol{a}$. Further, $X(g\boldsymbol{a}) \cap X(u') = (X(g) \cap X(u')) \cup (X(\boldsymbol{a}) \cap X(u')) \subseteq X(\gamma) \cup X(\boldsymbol{a}) = X(\gamma\boldsymbol{a})$, which satisfies the condition of the lemma, hence $\gamma\boldsymbol{a} \bullet g\boldsymbol{a} \mathrel{\underset{\mathsf{sh}}{\vdash}}^{*} \gamma\boldsymbol{a}u' \bullet g\boldsymbol{a}u' = \gamma u \bullet gu$ by the induction hypothesis. □

Lemma 4.8. $\gamma \mathrel{\underset{\mathrm{rm}}{\Longrightarrow}}^{*} g$ *implies* $\varepsilon \bullet \varepsilon \vdash^{*} \gamma \bullet g$ *for all graphs* $\gamma \in \mathcal{G}_{\Sigma}$ *and* $g \in \mathcal{G}_{\mathcal{T}}$*.*

Proof. Let $\gamma \underset{\mathrm{rm}}{\Longrightarrow}^{n} g$ be any derivation as in the lemma. We prove the proposition by induction over n.

For $n = 0$, we have $\gamma = g$ and, by Lemma 4.7, $\varepsilon \mathbin{\boldsymbol{.}} \varepsilon \underset{\mathsf{sh}}{\vdash}^{*} g \mathbin{\boldsymbol{.}} g = \gamma \mathbin{\boldsymbol{.}} g$.

For $n > 0$, the derivation must be of the form

$$\gamma = \alpha \boldsymbol{A}^{\mu} v \underset{\mathrm{rm}}{\Longrightarrow} \alpha \varrho^{\mu} v \underset{\mathrm{rm}}{\Longrightarrow}^{n-1} uv = g \tag{4}$$

for some $u, v \in \mathcal{G}_{\mathcal{T}}$, $\alpha \in \mathcal{G}_{\Sigma}$, rule $\boldsymbol{A} \to \varrho$, and renaming $\mu \colon X \to X$. By $\alpha \varrho^{\mu} \underset{\mathrm{rm}}{\Longrightarrow}^{n-1} u$ and the induction hypothesis,

$$\varepsilon \mathbin{\boldsymbol{.}} \varepsilon \vdash^{*} \alpha \varrho^{\mu} \mathbin{\boldsymbol{.}} u.$$

Moreover, by the definition of derivation moves, $X(\alpha) \cap X(\varrho^{\mu}) \subseteq X(\boldsymbol{A}^{\mu})$, which satisfies the condition for the following reduce move:

$$\alpha \varrho^{\mu} \mathbin{\boldsymbol{.}} u \underset{\boldsymbol{A}^{\mu} \Rightarrow \varrho^{\mu}}{\vdash} \alpha \boldsymbol{A}^{\mu} \mathbin{\boldsymbol{.}} u.$$

$X(u) \cap X(v) \subseteq X(\alpha \boldsymbol{A}^{\mu})$ follows from (4) and Fact 2.6, and therefore,

$$\alpha \boldsymbol{A}^{\mu} \mathbin{\boldsymbol{.}} u \underset{\mathsf{sh}}{\overset{*}{\vdash}} \alpha \boldsymbol{A}^{\mu} v \mathbin{\boldsymbol{.}} uv = \gamma \mathbin{\boldsymbol{.}} g.$$

by Lemma 4.7. □

Lemma 4.6 and Lemma 4.8 prove the correctness of the naïve shift-reduce parser:

Theorem 4.9. *For each graph $h \in \mathcal{G}_{\mathcal{T}}$, $\varepsilon \mathbin{\boldsymbol{.}} \varepsilon \vdash^{*} \boldsymbol{Z} \mathbin{\boldsymbol{.}} h$ if and only if $\boldsymbol{Z} \underset{\mathrm{rm}}{\Longrightarrow}^{*} h$.*

Proof. For the *only-if* direction, set $\gamma = g = \varepsilon$, $\gamma' = \boldsymbol{Z}$, and $g' = h$ in Lemma 4.6, and for the *if* direction, set $\gamma = \boldsymbol{Z}$ in Lemma 4.8. □

5. Viable Prefixes of Graphs

The naïve shift-reduce parser may always find a successful parse for a valid graph (and only for those), but it must always choose the right move to avoid backtracking. Bear in mind that the parser can always perform a shift move as long as the input graph has not yet been consumed in its entirety. In particular, all literals can be shifted right away. Also, a rule like $T^{y} \to \varepsilon$ in Example 4.2 can always be reduced. This will typically lead into a dead end. We shall now distinguish stacks that may occur in successful parses from those that do not. This will eventually result in the characteristic finite automaton that "assists" the parser. We follow a similar line of argument as for string parsing and define so-called *viable prefixes* first [2, Sect. 5.3.2].

Assumption 5.1. For the remainder of the paper, we add to Γ a new nonterminal Start of arity zero, and the rule $(\boldsymbol{\mathit{Start}} \to \boldsymbol{Z})$ with $\boldsymbol{\mathit{Start}} = \mathit{Start}()$. Thus, the derivations starting with $\boldsymbol{\mathit{Start}}$ are just those in the original grammar, but with an additional first step $\boldsymbol{\mathit{Start}} \underset{\text{rm}}{\Longrightarrow} \boldsymbol{Z}$. Clearly, the generated language is independent of whether Z or Start is considered to be the initial nonterminal.

A viable prefix is a prefix of a graph derivable from $\boldsymbol{\mathit{Start}}$ by a nonempty rightmost derivation, provided that this prefix does not extend past the right-hand side of the most recently applied rule. More formally:

Definition 5.2 (Viable Prefix). A graph $\gamma \in \mathcal{G}_\Sigma$ is called a *viable prefix* if there are graphs $\alpha, \beta \in \mathcal{G}_\Sigma$ as well as $z \in \mathcal{G}_\mathcal{T}$ and a literal $\boldsymbol{A} \in \mathit{Lit}_\mathcal{N}$ such that $\boldsymbol{\mathit{Start}} \underset{\text{rm}}{\Longrightarrow}^* \alpha\boldsymbol{A}z \underset{\text{rm}}{\Longrightarrow} \alpha\beta z$ and γ is a prefix of $\alpha\beta$.

Example 5.3. We illustrate Def. 5.2 using an initial segment of the rightmost derivation in Fig. 6 (though now beginning with $\mathit{Start}()$):

$$\begin{aligned} \mathit{Start}() &\underset{\text{rm}}{\Longrightarrow}^* \overbrace{\mathit{root}(1)}^{\alpha}\,\overbrace{T(1)}^{\boldsymbol{A}}\,\overbrace{e(1,2)\,e(2,4)}^{z}\\ &\underset{\text{rm}}{\Longrightarrow}_2 \underbrace{\overbrace{\mathit{root}(1)}^{\alpha}\,\overbrace{T(1)\,e(1,3)\,T(3)}^{\beta}}_{\gamma}\,\overbrace{e(1,2)\,e(2,4)}^{z}\;. \end{aligned}$$

The graph $\gamma = \mathit{root}(1)\,T(1)\,e(1,2)\,T(3)$ is the longest viable prefix of the derived graph; all prefixes of γ are viable as well.

Before we show that the set of viable prefixes is just the set of all stacks occurring in successful parses, we need the following two technical lemmata. Lemma 5.4 states that the set of viable prefixes does not change if we add to Def. 5.2 the additional requirement that the suffix v with $\gamma v = \alpha\beta$ is a terminal graph.

Lemma 5.4. *A graph $\gamma \in \mathcal{G}_\Sigma$ is a viable prefix if and only if there are graphs $\alpha, \beta \in \mathcal{G}_\Sigma$ as well as $v, z \in \mathcal{G}_\mathcal{T}$ and a literal $\boldsymbol{A} \in \mathit{Lit}_\mathcal{N}$ such that $\boldsymbol{\mathit{Start}} \underset{\text{rm}}{\Longrightarrow}^* \alpha\boldsymbol{A}z \underset{\text{rm}}{\Longrightarrow} \alpha\beta z = \gamma v z$.*

Proof. The *if* direction follows immediately from the definition of viable prefixes. For the *only-if* direction, let γ be any viable prefix. Hence, there is a rightmost derivation

$$\boldsymbol{\mathit{Start}} \underset{\text{rm}}{\Longrightarrow}^n \alpha\boldsymbol{A}z \underset{\text{rm}}{\Longrightarrow} \alpha\beta z \tag{5}$$

and $\alpha\beta = \gamma\delta$ (because we assume that Γ is reduced). Without loss of generality, assume that this derivation is maximal in the sense that there is no longer rightmost derivation $\boldsymbol{\mathit{Start}} \underset{\text{rm}}{\Longrightarrow}^m \alpha'\boldsymbol{A}'z' \underset{\text{rm}}{\Longrightarrow} \alpha'\beta'z'$ such that γ is a prefix of $\alpha'\beta'$ and $m > n$. Further assume that $\delta \notin \mathcal{G}_\mathcal{T}$, that is, there is a nonterminal literal $\boldsymbol{B}$ and graphs $\delta = \delta'\boldsymbol{B}u \underset{\text{rm}}{\Longrightarrow} \delta'\beta'u$. Then $\beta', \delta' \in \mathcal{G}_\Sigma$ and $u \in \mathcal{G}_\mathcal{T}$ such that $\boldsymbol{Start} \underset{\text{rm}}{\Longrightarrow}^n \alpha \boldsymbol{A} z \underset{\text{rm}}{\Longrightarrow} \alpha\beta z = \gamma\delta z = \gamma\delta' \boldsymbol{B} u z \underset{\text{rm}}{\Longrightarrow} \gamma\delta'\beta u z$ is a rightmost derivation, and γ is a prefix of $\gamma\delta'\beta$. But this rightmost derivation is longer than (5), contradicting the assumption. Hence, $\delta \in \mathcal{G}_{\mathcal{T}}$. ☐

We now show that the set of viable prefixes is just the set of all stacks occurring in successful parses. In Sect. 6, we shall then define nondeterministic CFAs and show that they just approve viable prefixes. Such a CFA will therefore allow to identify the stacks of successful parses.

Lemma 5.5. *For every sequence* $\varepsilon \bullet \varepsilon \vdash^* \gamma \bullet g$ *of moves such that* γ *is a viable prefix, there is a graph* $g' \in \mathcal{G}_{\mathcal{T}}$ *with* $\gamma \bullet g \vdash^* \boldsymbol{Z} \bullet g g'$.

Proof. Let $\varepsilon \bullet \varepsilon \vdash^* \gamma \bullet g$ be a sequence of moves as in the lemma. By Lemma 4.6, $\gamma \underset{\text{rm}}{\Longrightarrow}^* g$. We first show that there is a rightmost derivation

$$\boldsymbol{Start} \underset{\text{rm}}{\Longrightarrow}^n \alpha \boldsymbol{A} z \underset{\text{rm}}{\Longrightarrow} \alpha\beta z = \gamma v z \underset{\text{rm}}{\Longrightarrow}^* g v z \tag{6}$$

where $v \in \mathcal{G}_{\mathcal{T}}$. Since γ is a viable prefix and according to Lemma 5.4, there is a rightmost derivation $\boldsymbol{Start} \underset{\text{rm}}{\Longrightarrow}^n \hat{\alpha} \hat{\boldsymbol{A}} \hat{z} \underset{\text{rm}}{\Longrightarrow} \hat{\alpha}\hat{\beta}\hat{z}$ for some $n \in \mathbb{N}$ such that $\gamma\hat{v} = \hat{\alpha}\hat{\beta}$ for a terminal graph $\hat{v} \in \mathcal{G}_{\mathcal{T}}$. However, one cannot conclude $\gamma\hat{v}\hat{z} \underset{\text{rm}}{\Longrightarrow}^* g\hat{v}\hat{z}$ because of possible naming conflicts. To circumvent this problem, we rename all nodes that may cause such conflicts. For this purpose, choose any renaming $\mu \colon X \to X$ with $\mu(x) = x$ if $x \in X(\gamma)$ and $\mu(x) \notin X(g)$ otherwise, and let $\alpha = \hat{\alpha}^\mu$, $\beta = \hat{\beta}^\mu$, $v = \hat{v}^\mu$, $z = \hat{z}^\mu$, and $\boldsymbol{A} = \hat{\boldsymbol{A}}$. By the choice of μ, $\gamma = \gamma^\mu$ and $\gamma v = \alpha\beta$, as well as

$$X(vz) \cap X(g) \subseteq X(\gamma). \tag{7}$$

Hence, $\gamma v z \underset{\text{rm}}{\Longrightarrow}^* g v z$ by Fact 2.6. By Lemma 2.3, we thus have a derivation as in (6). Note that

$$X(\alpha) \cap X(\beta) \subseteq X(\boldsymbol{A}) \tag{8}$$

follows from $\alpha \boldsymbol{A} z \underset{\text{rm}}{\Longrightarrow} \alpha\beta z$.

We now show that, for every sequence $\varepsilon \bullet \varepsilon \vdash^* \gamma \bullet g$ of moves and every rightmost derivation (6), there is a sequence $\gamma \bullet g \vdash^* \boldsymbol{Z} \bullet g v z$ by induction over the length of the derivation in (6).

For $n = 0$, we have $\alpha = z = \varepsilon$ and $\boldsymbol{Start} \underset{\text{rm}}{\Longrightarrow} \beta = \boldsymbol{Z} = \gamma v$, i.e., $\gamma = \boldsymbol{Z}$ and $v = \varepsilon$. Hence, $\gamma \bullet g = \boldsymbol{Z} \bullet g v z \vdash^0 \boldsymbol{Z} \bullet g v z$.

For $n > 0$, we distinguish between two cases.

(1) The derivation (6) has the form

$$\boldsymbol{Start} \underset{\text{rm}}{\Longrightarrow}^{n-1} \delta \boldsymbol{B} u \underset{\text{rm}}{\Longrightarrow} \delta\varphi \boldsymbol{A} w u \underset{\text{rm}}{\Longrightarrow} \alpha\beta z = \gamma v z \underset{\text{rm}}{\Longrightarrow}^* g v z \tag{9}$$

where $\alpha = \delta\varphi$, $z = wu$, and $\alpha\beta = \gamma v$. By (7), (8), and Lemma 4.7,

$$\gamma \bullet g \vdash_{\mathsf{sh}}^* \gamma v \bullet g v = \alpha\beta \bullet g v \vdash_{\boldsymbol{A} \Rightarrow \beta} \alpha \boldsymbol{A} \bullet g v = \delta\varphi \boldsymbol{A} \bullet g v. \tag{10}$$

Note that (9) has the form of (6) where $\boldsymbol{B}$ plays the role of $\boldsymbol{A}$, $\delta\varphi\boldsymbol{A}$ the role of γ, w the role of v, and gv the role of g. Because of (10), we can make use of the induction hypothesis so that $\delta\varphi\boldsymbol{A} \bullet gv \vdash^* \boldsymbol{Z} \bullet gvz$.

(2) The given derivation (6) has the form

$$\boldsymbol{Start} \underset{\mathrm{rm}}{\Longrightarrow}^{n-1} \alpha\boldsymbol{A}u\boldsymbol{B}w \underset{\mathrm{rm}}{\Longrightarrow} \alpha\boldsymbol{A}uv'w \underset{\mathrm{rm}}{\Longrightarrow} \alpha\beta z = \gamma vz \underset{\mathrm{rm}}{\Longrightarrow}^* gvz \tag{11}$$

with $\boldsymbol{B} \Rightarrow v' \in \mathcal{G}_{\mathcal{T}}$ and $z = uv'w$. By (7), (8), and Lemma 4.7,

$$\gamma \bullet g \vdash^*_{\mathsf{sh}} \gamma v \bullet gv = \alpha\beta \bullet gv \vdash_{\boldsymbol{A} \Rightarrow \beta} \alpha\boldsymbol{A} \bullet gv. \tag{12}$$

Note that (11) has the form of (6) where $\boldsymbol{B}$ plays the role of $\boldsymbol{A}$, $\alpha\boldsymbol{A}$ the role of γ, uv' the role of v, and gv the role of g. Because of (12), we can make use of the induction hypothesis so that $\alpha\boldsymbol{A} \bullet gv \vdash^* \boldsymbol{Z} \bullet gvz$. ☐

Lemma 5.6. *For every sequence $\varepsilon \bullet \varepsilon \vdash^* \gamma \bullet u \vdash^* \boldsymbol{Z} \bullet uv$ of moves, γ is a viable prefix.*

Proof. If $\gamma = \boldsymbol{Z}$, then there is nothing to show because $\boldsymbol{Z}$ is a viable prefix due to $\boldsymbol{Start} \underset{\mathrm{rm}}{\Longrightarrow} \boldsymbol{Z}$. Otherwise, the sequence of moves has the form $\varepsilon \bullet \varepsilon \vdash^* \gamma \bullet u \vdash^+ \boldsymbol{Z} \bullet uv$. As the last move is a reduce move, the sequence can be written as

$$\varepsilon \bullet \varepsilon \vdash^* \gamma \bullet u \vdash^*_{\mathsf{sh}} \gamma v' \bullet uv' = \alpha\beta \bullet uv' \vdash_{\boldsymbol{A} \Rightarrow \beta} \alpha\boldsymbol{A} \bullet uv' \vdash^* \boldsymbol{Z} \bullet uv'v''$$

for some graphs $v', v'' \in \mathcal{G}_{\mathcal{T}}$ with $v = v'v''$. We can make use of Lemma 4.8 for each of these moves, yielding $\boldsymbol{Start} \underset{\mathrm{rm}}{\Longrightarrow} \boldsymbol{Z} \underset{\mathrm{rm}}{\Longrightarrow}^* \alpha\boldsymbol{A}v'' \underset{\mathrm{rm}}{\Longrightarrow} \alpha\beta v'' = \gamma v'v''$, i.e., γ is a viable prefix. ☐

An immediate consequence of Lemma 5.5 and Lemma 5.6 is the following:

Theorem 5.7. *For every sequence $\varepsilon \bullet \varepsilon \vdash^* \gamma \bullet u$ of moves, there is a graph $v \in \mathcal{G}_{\mathcal{T}}$ with $\gamma \bullet u \vdash^* \boldsymbol{Z} \bullet uv$ if and only if γ is a viable prefix.*

In other words, the stack of a reachable configuration is a viable prefix if and only if the nondeterministic parser can reach the accepting configuration for *some* possible rest graph.

6. Nondeterministic Characteristic Finite-State Automata

Let us now start to develop the means to “assist” the shift-reduce parser to restrict its moves to promising ones. The first step towards this goal is the construction of nondeterministic characteristic finite automata, which we define next.

Definition 6.1 (Nondeterministic CFA). The *nondeterministic characteristic finite automaton* (nCFA) for Γ is the tuple $\mathfrak{A} = (Q, q_0, \Delta)$ consisting of the following components:

1. $Q = \{\boldsymbol{A} \to \alpha \boldsymbol{.} \beta \mid (\boldsymbol{A} \to \alpha\beta) \in \mathcal{R}\}$ is a finite set of *states*.
2. $q_0 = (\boldsymbol{Start} \to \boldsymbol{.} \boldsymbol{Z})$ ist the *initial state*.
3. $\Delta \subseteq Q \times (Lit_\Sigma \cup \{\varepsilon\}) \times Q$ is a ternary *transition relation*. Writing $p \xrightarrow{x} q$ if $(p, x, q) \in \Delta$, the transitions constituting Δ are:
   (a) $(\boldsymbol{A} \to \alpha \boldsymbol{.} \boldsymbol{l}\beta) \xrightarrow{\boldsymbol{l}} (\boldsymbol{A} \to \alpha\boldsymbol{l} \boldsymbol{.} \beta)$ for every state $(\boldsymbol{A} \to \alpha \boldsymbol{.} \boldsymbol{l}\beta) \in Q$, where $\boldsymbol{l} \in Lit_\Sigma$; such a transition is called *goto transition*.
   (b) $q \xrightarrow{\varepsilon} p$ for all states $q = (\boldsymbol{A} \to \alpha \boldsymbol{.} \boldsymbol{B}\beta) \in Q$ and $p = (\boldsymbol{C} \to \boldsymbol{.} \gamma) \in Q$ such that $\ell(\boldsymbol{B}) = \ell(\boldsymbol{C}) \in \mathcal{N}$; such a transition is called *closure transition*.

Assumption 6.2. Since we assume a fixed HR grammar Γ, we assume a fixed nCFA $\mathfrak{A} = (Q, q_0, \Delta)$ obtained from Γ from now on.

Following the ideas discussed earlier, each item is a rule with a dot somewhere between literals in its right-hand side, indicating the division between literals that have already been processed and those which have not. Accordingly, the dot is moved across a literal when a corresponding literal is processed. We now formalize how an nCFA approves graphs (which will later turn out to be viable prefixes).

An nCFA approves a graph φ if the sequence $lit(\varphi)$ of literals corresponds to a sequence of state transitions, starting at the initial state. We define the notion of *nCFA configurations* (or just *configurations* if it is clear from the context) to formalize this:

Definition 6.3 (nCFA Configuration). An *nCFA configuration* $\varphi\Diamond[q]^\mu$ consists of

- a graph $\varphi \in \mathcal{G}_\Sigma$,
- a state $q = (\boldsymbol{A} \to \alpha \boldsymbol{.} \beta) \in Q$, and
- an injective partial function $\mu \colon X \rightharpoonup X$ with $dom(\mu) = X(\alpha)$.

The function μ in an nCFA configuration $\varphi\Diamond[q]^\mu$ corresponds to the match defined in Def. 2.2, which maps rule nodes to nodes of the graph processed so far. In an nCFA configuration, this match has in general not completely been determined yet; the mapping of nodes that have not yet been processed is still undefined. The mapping μ is extended when a literal is processed, which means that all its attached nodes, if they have not been processed earlier, are now processed as well. As a consequence, nodes of state q must be mapped by μ to nodes in φ—unless they have not been processed yet, in which case they are not in $dom(\mu)$. Such nodes may only occur in literals behind the dot in q, which is reflected by the requirement that $dom(\mu) = X(\alpha)$.

To compare literals that have only partially been matched, let $- \notin X$ be a special value denoting 'undefined'. Given a partial injective function $\mu \colon X \rightharpoonup X$ and a literal $\boldsymbol{l} = a(x_1, \ldots, x_k)$, we let $\boldsymbol{l}^\mu = a(y_1, \ldots, y_k)$ where, for all $1 \le i \le k$, $y_i = \mu(x_i)$ if $x_i \in dom(\mu)$ and $y_i = -$ otherwise. Note that $\boldsymbol{l}^\mu$ is a literal if (and only if) $x_1, \ldots, x_k \in \mathit{dom}(\mu)$. “Literals” with ‘–’ are called *pseudo-literals*. We let $X(\boldsymbol{l}^\mu)$ denote $\mu(X(\boldsymbol{l}))$.

An nCFA works by processing and consuming literals step by step while moving from state to state, represented by a corresponding sequence of nCFA configurations, starting at $\varepsilon\diamond[q_0]^\iota$, the initial configuration. Here, $\iota\colon X \rightharpoonup X$ is the totally undefined function with $\mathit{dom}(\iota) = \varnothing$. Intuitively, $\varepsilon\diamond[q_0]^\iota$ being the initial configuration means that the nCFA starts with the empty graph ε in $q_0 = (\boldsymbol{Start} \to \bullet\boldsymbol{Z})$ and with no nodes mapped yet, the latter being indicated by the empty domain of ι.

Each step of the nCFA is modeled by a move, defined as follows:

Definition 6.4 (nCFA Move). Let $\varphi\diamond[q]^\mu$ be a configuration. A goto transition $q \xrightarrow{\boldsymbol{l}} q'$ induces a *goto move*

$$\varphi\diamond[q]^\mu \underset{\mathsf{go}}{\vdash\!\!\frown} \varphi\boldsymbol{l}^\nu\diamond[q']^\nu$$

where $\mu \sqsubseteq \nu$, $\mathit{dom}(\nu) = \mathit{dom}(\mu) \cup X(\boldsymbol{l})$, and $X(\boldsymbol{l}^\nu) \cap X(\varphi) \subseteq X(\boldsymbol{l}^\mu)$.

A closure transition $q \xrightarrow{\varepsilon} q'$ with $q = (\boldsymbol{A} \to \alpha\bullet\boldsymbol{B}\beta)$ and $q' = (\boldsymbol{C} \to \bullet\delta)$ induces a *closure move*

$$\varphi\diamond[q]^\mu \underset{\mathsf{cl}}{\vdash\!\!\frown} \varphi\diamond[q']^\nu$$

where $\boldsymbol{B}^\mu = \boldsymbol{C}^\nu$ and $\mathit{dom}(\nu) \subseteq X(\boldsymbol{C})$.

We write $C \vdash\!\!\frown C'$ if either $C \underset{\mathsf{go}}{\vdash\!\!\frown} C'$ or $C \underset{\mathsf{cl}}{\vdash\!\!\frown} C'$, and call this a *move*.

A goto move applies a goto transition by processing a (possibly nonterminal) literal $\boldsymbol{l}$. The parser consumes the corresponding literal, which also means that all its nodes have been consumed after this move, and all nodes of $\boldsymbol{l}$ are mapped by the resulting node mapping ν. The processed literal is hence $\boldsymbol{l}^\nu$, which is added to the end of the approved graph, resulting in $\varphi\boldsymbol{l}^\nu$. The first two conditions, $\mu \sqsubseteq \nu$ and $\mathit{dom}(\nu) = \mathit{dom}(\mu) \cup X(\boldsymbol{l})$, state that the mapping ν extends the previous mapping μ so as to map the entire literal $\boldsymbol{l}$ to the input graph. The remaining condition $X(\boldsymbol{l}^\nu) \cap X(\varphi) \subseteq X(\boldsymbol{l}^\mu)$ ensures that nodes that have already been consumed (i.e., those in φ) are not matched another time by extending μ to ν.

A closure move applies a closure transition and corresponds to a (rightmost) derivation step, i.e., the mapping μ of nodes in $\boldsymbol{B}$ is translated into a mapping ν of the corresponding nodes in $\boldsymbol{C}$. Note that $\mathit{dom}(\mu)$ and $\mathit{dom}(\nu)$ are unrelated because the nodes in $\boldsymbol{B}$ and $\boldsymbol{C}$ may differ. Only nodes appearing in $\boldsymbol{C}$—but not necessarily all of them—are mapped by ν; other nodes of state q' are not mapped because their corresponding nodes have not yet been consumed.

Example 6.5 (The nCFA for the Tree-Generating Grammar). Fig. 8 shows the transition diagram of the nondeterministic CFA for the tree-generating grammar in Example 4.2. In Fig. 9 we show moves of the nondeterministic CFA.

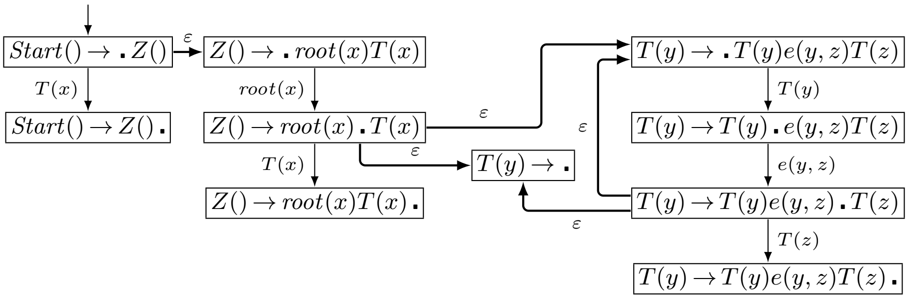

Figure 8: Nondeterministic CFA for the tree-generating grammar in Example 4.2. The initial state appears in the upper left. Closure transitions are drawn with thicker lines.

$$
\begin{array}{lr}
 & \varepsilon \diamond [\mathit{Start}() \to \centerdot Z()] \\
\underset{\mathsf{cl}}{\vdash\!\!\sim} & \varepsilon \diamond [Z() \to \centerdot \mathit{root}(x)T(x)]^{x/x} \\
\underset{\mathsf{go}}{\vdash\!\!\sim} & \mathit{root}(1) \diamond [Z() \to \mathit{root}(x) \centerdot T(x)]^{x/1} \\
\underset{\mathsf{cl}}{\vdash\!\!\sim} & \mathit{root}(1) \diamond [T(y) \to \centerdot T(y)e(y,z)T(z)]^{y/1} \\
\underset{\mathsf{go}}{\vdash\!\!\sim} & \mathit{root}(1)\, T(1) \diamond [T(y) \to T(y) \centerdot e(y,z)T(z)]^{y/1} \\
\underset{\mathsf{go}}{\vdash\!\!\sim} & \mathit{root}(1)\, T(1)\, e(1,2) \diamond [T(y) \to T(y)e(y,z) \centerdot T(z)]^{y/1,z/2} \\
\underset{\mathsf{cl}}{\vdash\!\!\sim} & \mathit{root}(1)\, T(1)\, e(1,2) \diamond [T(y) \to \centerdot T(y)e(y,z)T(z)]^{y/2} \\
\underset{\mathsf{go}}{\vdash\!\!\sim} & \mathit{root}(1)\, T(1)\, e(1,2)\, T(2) \diamond [T(y) \to T(y) \centerdot e(y,z)T(z)]^{y/2} \\
\underset{\mathsf{go}}{\vdash\!\!\sim} & \mathit{root}(1)\, T(1)\, e(1,2)\, T(2)\, e(2,4) \diamond [T(y) \to T(y)e(y,z) \centerdot T(z)]^{y/2,z/4} \\
\underset{\mathsf{go}}{\vdash\!\!\sim} & \mathit{root}(1)\, T(1)\, e(1,2)\, T(2)\, e(2,4)\, T(4) \diamond [T(y) \to T(y)e(y,z)T(z) \centerdot]^{y/2,z/4}
\end{array}
$$

Figure 9: Approval of a graph with a nondeterministic CFA. Renamings μ of states q_i with $\mu(x_1) = y_1, \dots, \mu(x_k) = y_k$ are represented by exponents $x_1/y_1, \dots, x_k/y_k$

Every graph approved by the automaton is a viable prefix occurring in the rightmost derivation in Fig. 6 of Example 4.2, and in the parse shown in Fig. 7 of Example 4.5. We will show in the sequel that this is not a coincidence. ☐

Definition 6.6. The nCFA *approves* a graph $\varphi \in \mathcal{G}_\Sigma$ if there is a configuration $C = \varphi \diamond [q]^\mu$ such that $\varepsilon \diamond [q_0]^\iota \vdash\!\!\sim^* C$.

It is rather obvious that one can arbitrarily rename input graph nodes without affecting approval by the nCFA:

Fact 6.7. *$\varepsilon \diamond [q_0]^\iota \vdash\!\!\sim^n \varphi \diamond [q]^\mu$ implies $\varepsilon \diamond [q_0]^\iota \vdash\!\!\sim^n \varphi^f \diamond [q]^{f \circ \mu}$ for every renaming $f \colon X \to X$.*

Moreover, properties of goto moves can be generalized to sequences of goto moves:

Lemma 6.8. *$\varphi\diamond[\boldsymbol{A}\to\alpha\boldsymbol{.}\beta\gamma]^{\mu} \mathrel{\vdash\!\!\frown}_{\mathsf{go}}^{*} \varphi\psi\diamond[\boldsymbol{A}\to\alpha\beta\boldsymbol{.}\gamma]^{\nu}$ implies $\mu\sqsubseteq\nu$, $dom(\nu)=dom(\mu)\cup X(\beta)$, and $X(\varphi)\cap X(\alpha^{\nu}\beta^{\nu}\gamma^{\nu})\subseteq X(\alpha^{\mu}\beta^{\mu}\gamma^{\mu})$.*

Proof. Consider any sequence $\varphi\diamond[\boldsymbol{A}\to\alpha\boldsymbol{.}\beta\gamma]^{\mu} \mathrel{\vdash\!\!\frown}_{\mathsf{go}}^{n} \varphi\psi\diamond[\boldsymbol{A}\to\alpha\beta\boldsymbol{.}\gamma]^{\nu}$. We prove the proposition by induction over n. The proposition follows for $n=0$ from $\mu=\nu$.

For $n>0$, we have the sequence

$$\varphi\diamond[\boldsymbol{A}\to\alpha\boldsymbol{.}\beta\gamma]^{\mu} \mathrel{\vdash\!\!\frown}_{\mathsf{go}}^{n-1} \varphi\psi'\diamond[\boldsymbol{A}\to\alpha\beta'\boldsymbol{.}\boldsymbol{e}\gamma]^{\nu'} \mathrel{\vdash\!\!\frown}_{\mathsf{go}} \varphi\psi\diamond[\boldsymbol{A}\to\alpha\beta\boldsymbol{.}\gamma]^{\nu}$$

with $\beta=\beta'\boldsymbol{e}$ and $\psi=\psi'\boldsymbol{e}^{\nu}$. By the induction hypothesis and the definition of goto moves,

$$\begin{aligned}
\mu\sqsubseteq\nu'\sqsubseteq\nu \qquad\qquad & \tag{13}\\
X(\varphi)\cap X(\alpha^{\nu'}\beta^{\nu'}\gamma^{\nu'})\subseteq X(\alpha^{\mu}\beta^{\mu}\gamma^{\mu}) \qquad & \tag{14}\\
dom(\nu)=dom(\nu')\cup X(\boldsymbol{e})&=dom(\mu)\cup X(\beta')\cup X(\boldsymbol{e})\\
&=dom(\mu)\cup X(\beta) \tag{15}\\
X(\boldsymbol{e}^{\nu})\cap X(\varphi\psi')\subseteq X(\boldsymbol{e}^{\nu'}) \qquad & \tag{16}
\end{aligned}$$

By Def. 6.3 and (13), $X(\alpha^{\nu})=X(\alpha^{\nu'})$ and $X(\beta'^{\nu})=X(\beta'^{\nu'})$. Moreover, $X(\gamma^{\nu})\subseteq X(\gamma^{\nu'})\cup X(\boldsymbol{e}^{\nu})$ using (13) and (15), and $X(\boldsymbol{e}^{\nu})\cap X(\varphi)\subseteq X(\boldsymbol{e}^{\nu'})\cap X(\varphi)$ using (16). Therefore, $X(\varphi)\cap X(\alpha^{\nu}\beta^{\nu}\gamma^{\nu})\subseteq X(\varphi)\cap X(\alpha^{\nu'}\beta^{\nu'}\gamma^{\nu'})\subseteq X(\alpha^{\mu}\beta^{\mu}\gamma^{\mu})$ using (14). ☐

The following lemma shows that all mapped nodes of the current nCFA state have been consumed already, i.e., occur in consumed literals.

Lemma 6.9. *$\varepsilon\diamond[q_0]^{\iota} \mathrel{\vdash\!\!\frown}^{*} \varphi\diamond[\boldsymbol{A}\to\alpha\boldsymbol{.}\beta]^{\mu}$ implies $X(\alpha^{\mu}\beta^{\mu})\subseteq X(\varphi)$.*

Proof. We prove the proposition by induction over the length n of the sequence of moves. The proposition follows for $n=0$ from $\boldsymbol{A}=\boldsymbol{Start}$, $\alpha=\varnothing$, $\beta=\boldsymbol{Z}$, and $\mu=\iota$.

For $n>0$ and the last move being a closure move, the sequence has the form

$$\varepsilon\diamond[q_0]^{\iota} \mathrel{\vdash\!\!\frown}^{n-1} \varphi\diamond[\boldsymbol{B}\to\delta\boldsymbol{.}\boldsymbol{C}\gamma]^{\nu} \mathrel{\vdash\!\!\frown}_{\mathsf{cl}} \varphi\diamond[\boldsymbol{A}\to\alpha\boldsymbol{.}\beta]^{\mu}.$$

with $\alpha=\varepsilon$, $\boldsymbol{C}^{\nu}=\boldsymbol{A}^{\mu}$, and $dom(\mu)\subseteq X(\boldsymbol{A})$. Therefore, $X(\alpha^{\mu}\beta^{\mu})=X(\boldsymbol{A}^{\mu})=X(\boldsymbol{C}^{\mu})\subseteq X(\delta^{\nu}\boldsymbol{C}^{\nu}\gamma^{\nu})\subseteq X(\varphi)$ using the induction hypothesis.

For $n>0$ and the last being a goto move, the sequence has the form

$$\varepsilon\diamond[q_0]^{\iota} \mathrel{\vdash\!\!\frown}^{n-1} \varphi'\diamond[\boldsymbol{A}\to\alpha'\boldsymbol{.}\boldsymbol{e}\beta]^{\nu} \mathrel{\vdash\!\!\frown}_{\mathsf{go}} \varphi\diamond[\boldsymbol{A}\to\alpha\boldsymbol{.}\beta]^{\mu}$$

with $\alpha = \alpha'\boldsymbol{e}$, $\varphi = \varphi'\boldsymbol{e}^\nu$, $\nu \sqsubseteq \mu$, $\mathit{dom}(\mu) = \mathit{dom}(\nu) \cup X(\boldsymbol{e})$, and $X(\boldsymbol{e}^\mu) \cap X(\varphi') \subseteq X(\boldsymbol{e}^\nu)$. Therefore, $X(\beta^\mu) \subseteq X(\beta^\nu) \cup X(\boldsymbol{e}^\mu)$ and

$$\begin{aligned}
X(\alpha^\mu \beta^\mu) &\subseteq X({\alpha'}^\nu) \cup X(\boldsymbol{e}^\mu) \cup X(\beta^\nu) \\
&\subseteq X({\alpha'}^\nu \boldsymbol{e}^\nu \beta^\nu) \cup X(\boldsymbol{e}^\mu) \\
&\subseteq X(\varphi') \cup X(\boldsymbol{e}^\mu) \\
&= X(\varphi)
\end{aligned}$$

using the induction hypothesis. $\square$

We now show that the graphs approved by the nCFA are viable prefixes (Lemma 6.10) and vice versa (Lemma 6.11).

Lemma 6.10. *For every sequence*

$$\varepsilon\Diamond[q_0]^\iota \mathrel{\vdash\!\!\sim}^* \varphi\Diamond[\boldsymbol{A} \to \alpha \mathbin{\bullet} \beta]^\mu$$

of moves and every injective function $\tau \colon X(\alpha\beta) \to \mathcal{X}$ *with* $\mu \sqsubseteq \tau$ *and* $X(\beta^\tau) \cap X(\varphi) \subseteq X(\beta^\mu)$*, there exist* $\psi \in \mathcal{G}_\Sigma$ *and* $z \in \mathcal{G}_\mathcal{T}$ *such that*

$$\boldsymbol{\mathit{Start}} \underset{\text{rm}}{\Longrightarrow}^* \psi \boldsymbol{A}^\tau z \underset{\text{rm}}{\Longrightarrow} \psi \alpha^\tau \beta^\tau z = \varphi \beta^\tau z.$$

Proof. We prove the proposition by induction over the number n of moves. For $n = 0$ the proposition follows from the definition of initial nCFA configurations (with $\tau = \iota$, $\boldsymbol{A} = \boldsymbol{\mathit{Start}} = \boldsymbol{\mathit{Start}}^\tau$, $\varphi = \alpha = z = \varepsilon$, and $\beta = \boldsymbol{\mathit{Start}}$).

For $n > 1$ and the last move being a goto move, we have

$$\varepsilon\Diamond[q_0]^\iota \mathrel{\vdash\!\!\sim}^{n-1} \varphi'\Diamond[\boldsymbol{A} \to \alpha' \mathbin{\bullet} \boldsymbol{e}\beta]^\nu \mathrel{\underset{\mathsf{go}}{\vdash\!\!\sim}} \varphi\Diamond[\boldsymbol{A} \to \alpha'\boldsymbol{e} \mathbin{\bullet} \beta]^\mu = \varphi\Diamond[\boldsymbol{A} \to \alpha \mathbin{\bullet} \beta]^\mu$$

where

$$\varphi = \varphi'\boldsymbol{e}^\mu \tag{17}$$

$$\nu \sqsubseteq \mu \tag{18}$$

$$\mathit{dom}(\mu) = \mathit{dom}(\nu) \cup X(\boldsymbol{e}) \tag{19}$$

$$X(\boldsymbol{e}^\mu) \cap X(\varphi') \subseteq X(\boldsymbol{e}^\nu), \tag{20}$$

Let τ be as in the lemma. Then $\nu \sqsubseteq \mu \sqsubseteq \tau$. In order to make use of the induction hypothesis, we additionally need to show that $X(\boldsymbol{e}^\tau\beta^\tau) \cap X(\varphi') \subseteq X(\boldsymbol{e}^\nu\beta^\nu)$. In fact, we have

$$\begin{aligned}
X(\boldsymbol{e}^\tau\beta^\tau) \cap X(\varphi') &= (X(\boldsymbol{e}^\tau) \cap X(\varphi')) \cup (X(\beta^\tau) \cap X(\varphi')) \\
&= (X(\boldsymbol{e}^\mu) \cap X(\varphi')) \cup (X(\beta^\tau) \cap X(\varphi) \cap X(\varphi')) \\
&\subseteq X(\boldsymbol{e}^\nu) \cup (X(\beta^\mu) \cap X(\varphi')) \\
&\subseteq X(\boldsymbol{e}^\nu) \cup X(\beta^\nu) \\
&= X(\boldsymbol{e}^\nu\beta^\nu).
\end{aligned}$$

Hence the induction hypothesis applies, yielding $\psi \in \mathcal{G}_\Sigma$ and $z \in \mathcal{G}_\mathcal{T}$ such that $\boldsymbol{Start} \underset{\text{rm}}{\Longrightarrow}^* \psi \boldsymbol{A}^\tau z \underset{\text{rm}}{\Longrightarrow} \psi \alpha'^\tau \boldsymbol{e}^\tau \beta^\tau z = \varphi \beta^\tau z$, which proves the proposition.

For $n > 0$ and the last move being a closure move, we have

$$\varepsilon \diamond [q_0]^\iota \vdash\!\!\sim^{n-1} \varphi \diamond [\boldsymbol{B} \to \gamma \,\boldsymbol{.}\, \boldsymbol{C} \delta]^\nu \vdash\!\!\sim \varphi \diamond [\boldsymbol{A} \to \,\boldsymbol{.}\, \beta]^\mu$$

where $\boldsymbol{C}^\nu = \boldsymbol{A}^\mu$, $dom(\mu) \subseteq X(\boldsymbol{A})$, and $\alpha = \varepsilon$. Again, let $\tau \colon X(\alpha\beta) \to X$ be as in the statement of the lemma. To be able to use the induction hypothesis we need an injective function $\eta \colon X(\gamma \boldsymbol{C} \delta) \to X$ such that $\nu \sqsubseteq \eta$ and $X(\boldsymbol{C}^\eta \delta^\eta) \cap X(\varphi) \subseteq X(\boldsymbol{C}^\nu \delta^\nu)$. But we also need $\boldsymbol{C}^\eta = \boldsymbol{A}^\tau$ in order to conclude a derivation $\boldsymbol{Start} \underset{\text{rm}}{\Longrightarrow}^* \psi \boldsymbol{B}^\eta w \underset{\text{rm}}{\Longrightarrow} \psi \gamma^\eta \boldsymbol{C}^\eta \delta^\eta w = \varphi \boldsymbol{A}^\tau \delta^\eta w$. However, this may be impossible because of naming conflicts.

To circumvent this problem, we rename all nodes that may cause such conflicts and use Fact 6.7. For this purpose, we choose any renaming $f \colon X \to X$ with $f(x) = x$ for $x \in X(\varphi)$ and $f(x) \notin X(\beta^\tau)$ for $x \in X(\beta^\tau) \setminus X(\varphi)$. We then have $X(\boldsymbol{C}^\nu) \subseteq X(\varphi)$ because of Lemma 6.9, and hence $\boldsymbol{C}^{f \circ \nu} = (\boldsymbol{C}^\nu)^f = \boldsymbol{C}^\nu = \boldsymbol{A}^\mu$ as well as $\varphi^f = \varphi$, and therefore

$$\varepsilon \diamond [q_0]^\iota \vdash\!\!\sim^{n-1} \varphi \diamond [\boldsymbol{B} \to \gamma \,\boldsymbol{.}\, \boldsymbol{C} \delta]^{f \circ \nu} \vdash\!\!\sim \varphi \diamond [\boldsymbol{A} \to \,\boldsymbol{.}\, \beta]^\mu.$$

We can now choose a renaming $\eta \colon X \to X$ with $f \circ \nu \sqsubseteq \eta$, $X(\boldsymbol{C}^\eta \delta^\eta) \cap X(\varphi) \subseteq X(\boldsymbol{C}^{f \circ \nu} \delta^{f \circ \nu})$, and $\boldsymbol{C}^\eta = \boldsymbol{A}^\tau$, and by the induction hypothesis, $\boldsymbol{Start} \underset{\text{rm}}{\Longrightarrow}^* \psi \boldsymbol{B}^\eta w \underset{\text{rm}}{\Longrightarrow} \psi \gamma^\eta \boldsymbol{C}^\eta \delta^\eta w = \varphi \boldsymbol{A}^\tau \delta^\eta w$. Although $\boldsymbol{A}^\tau \Rightarrow \beta^\tau$, we cannot conclude $\varphi \boldsymbol{A}^\tau \delta^\eta w \Rightarrow \varphi \beta^\tau \delta^\eta w$ because $\delta^\eta w$ may contain nodes that are created by the derivation $\boldsymbol{A}^\tau \Rightarrow \beta^\tau$. Again, we solve this problem by renaming the conflicting nodes in $\delta^\eta w$ to new nodes. For this purpose, let $Y = X(\delta^\eta w) \setminus X(\varphi \boldsymbol{A}^\tau)$ and choose, for each $y \in Y$, a new node $n_y \in X \setminus X(\varphi \beta^\tau \delta^\eta w)$. Let $h \colon X \to X$ be a renaming with $h(x) = n_x$ if $x \in Y$ and $h(x) = x$ for $x \in X(\varphi \beta^\tau)$. By Lemma 2.3, $\boldsymbol{Start} \underset{\text{rm}}{\Longrightarrow}^* (\varphi \boldsymbol{A}^\tau \delta^\eta w)^h = \varphi \boldsymbol{A}^\tau \delta^{h \circ \eta} w^h$. By the definition of h, and because Γ is reduced, there is a graph $u \in \mathcal{G}_\mathcal{T}$ such that

$$\varphi \boldsymbol{A}^\tau \delta^{h \circ \eta} w^h \Rightarrow \varphi \beta^\tau \delta^{h \circ \eta} w^h \Rightarrow^* \varphi \beta^\tau u w^h.$$

Therefore, $\boldsymbol{Start} \underset{\text{rm}}{\Longrightarrow}^* \varphi \beta^\tau z$ for $z = u w^h$, which proves the proposition. ☐

Lemma 6.11. *For every rightmost derivation* $\boldsymbol{Start} \underset{\text{rm}}{\Longrightarrow}^* \alpha \boldsymbol{A} z \underset{\text{rm}}{\Longrightarrow} \alpha \beta z$ *and each prefix* φ *of* $\alpha\beta$*, there is a sequence* $\varepsilon \diamond [q_0]^\iota \vdash\!\!\sim^* \varphi \diamond [p]^\nu$ *of moves (for a suitable state* $[p]^\nu$*).*

Proof. We prove by induction over n that $\boldsymbol{Start} \underset{\text{rm}}{\Longrightarrow}^n \alpha \boldsymbol{A} z \underset{\text{rm}}{\Longrightarrow} \alpha \beta z$ implies $\varepsilon \diamond [q_0]^\iota \vdash\!\!\sim^* \varphi \diamond [p]^\nu$ for every prefix φ of $\alpha\beta$.

For $n = 0$, the derivation is of the form $\boldsymbol{Start} \underset{\text{rm}}{\Longrightarrow} \boldsymbol{Z} = \beta$ and we have $\varphi = \varepsilon$ or $\varphi = \boldsymbol{Z}$. Hence, $\varphi \in \{\varepsilon, \boldsymbol{Z}\}$, and the proposition follows by making no move at all, or by making the goto move

$$\varepsilon \diamond [q_0]^\iota \underset{\text{go}}{\vdash\!\!\sim} \boldsymbol{Z} \diamond [\boldsymbol{Start} \to \boldsymbol{Z} \,\boldsymbol{.}\,]^\iota.$$

For $n > 0$, the initial part of the derivation up to $\alpha\boldsymbol{A}z$ has the form

$$\boldsymbol{Start} \underset{\mathrm{rm}}{\Longrightarrow}^{n-1} \vartheta\boldsymbol{X}w \underset{\mathrm{rm}}{\Longrightarrow} \vartheta\varrho w = \alpha\boldsymbol{A}z.$$

There are two cases:

(1) $\varrho \in \mathcal{G}_{\mathcal{T}}$.
Then $\vartheta = \alpha\boldsymbol{A}u$ for some $u \in \mathcal{G}_{\mathcal{T}}$, $\boldsymbol{Start} \underset{\mathrm{rm}}{\Longrightarrow}^{n-1} \alpha\boldsymbol{A}u\boldsymbol{X}w \underset{\mathrm{rm}}{\Longrightarrow} \alpha\boldsymbol{A}u\varrho w \underset{\mathrm{rm}}{\Longrightarrow} \alpha\beta u\varrho w$. We distinguish two sub-cases:

(1a) φ is a prefix of α.
Then φ is a prefix of $\alpha\boldsymbol{A}u\varrho$ and hence the proposition follows directly from the induction hypothesis.

(1b) $\varphi = \alpha\tau$ for a prefix τ of β.
By the induction hypothesis, there is a sequence $\varepsilon\diamond[q_0]^{\iota} \mathrel{\vdash\!\!\frown}^{m} \alpha\boldsymbol{A}\diamond[p]^{\nu}$ of moves. W.l.o.g, let m be the minimum number of such moves. By Def. 6.4, this sequence must be of the form

$$\varepsilon\diamond[q_0]^{\iota} \mathrel{\vdash\!\!\frown}^{m-1} \alpha\diamond[\boldsymbol{B} \to \delta \mathbin{\centerdot} \boldsymbol{C}\delta']^{\mu} \underset{\mathsf{go}}{\mathrel{\vdash\!\!\frown}} \alpha\boldsymbol{A}\diamond[\boldsymbol{B} \to \delta\boldsymbol{C} \mathbin{\centerdot} \delta']^{\nu}$$

with $\boldsymbol{A} = \boldsymbol{C}^{\nu}$. Suppose that $\boldsymbol{A} \Rightarrow \beta$ uses rule $\boldsymbol{D} \to \psi\bar{\psi}$ with $|\tau| = |\psi|$. By making a closure move instead of the goto move at the end of the sequence above, we obtain

$$\varepsilon\diamond[q_0]^{\iota} \mathrel{\vdash\!\!\frown}^{m-1} \alpha\diamond[\boldsymbol{B} \to \delta \mathbin{\centerdot} \boldsymbol{C}\delta']^{\mu} \underset{\mathsf{cl}}{\mathrel{\vdash\!\!\frown}} \alpha\diamond[\boldsymbol{D} \to \mathbin{\centerdot} \psi\bar{\psi}]^{\sigma}$$

with $\boldsymbol{C}^{\mu} = \boldsymbol{D}^{\sigma}$. The proposition follows when applying the appropriate number of goto moves:

$$\alpha\diamond[\boldsymbol{D} \to \mathbin{\centerdot} \psi\bar{\psi}]^{\sigma} \underset{\mathsf{go}}{\mathrel{\vdash\!\!\frown}}^{*} \alpha\tau\diamond[\boldsymbol{D} \to \psi \mathbin{\centerdot} \bar{\psi}]^{\sigma'} = \varphi\diamond[\boldsymbol{D} \to \psi \mathbin{\centerdot} \bar{\psi}]^{\sigma'}.$$

(2) $\varrho \notin \mathcal{G}_{\mathcal{T}}$.
Then $\boldsymbol{A}$ is a literal in ϱ and the given derivation has the form $\boldsymbol{Start} \underset{\mathrm{rm}}{\Longrightarrow}^{n-1} \vartheta\boldsymbol{X}w \underset{\mathrm{rm}}{\Longrightarrow} \vartheta\gamma\boldsymbol{A}uw \underset{\mathrm{rm}}{\Longrightarrow} \vartheta\gamma\beta uw$, where $\vartheta\gamma = \alpha$ and thus φ is a prefix of $\vartheta\gamma\beta$. We distinguish two sub-cases:

(2a) φ is prefix of $\vartheta\gamma$.
As in case 1a, the proposition follows directly from the induction hypothesis because φ is a prefix of $\vartheta\gamma\boldsymbol{A}u$.

(2b) $\varphi = \vartheta\gamma\tau$ for a prefix τ of β.
By the induction hypothesis, there is a sequence $\varepsilon\diamond[q_0]^{\iota} \mathrel{\vdash\!\!\frown}^{*} \vartheta\gamma\boldsymbol{A}\diamond[p]^{\nu}$, and with a similar argument as in case 1b, $\varepsilon\diamond[q_0]^{\iota} \mathrel{\vdash\!\!\frown}^{*} \vartheta\gamma\tau\diamond[p']^{\sigma}$. ☐

An immediate consequence of Lemma 6.10 and Lemma 6.11 is the following:

Theorem 6.12. *A graph $\varphi \in \mathcal{G}_{\Sigma}$ is a viable prefix if and only if the nCFA approves φ.*

On the one hand, we have shown at the end of Sect. 4 that the naïve nondeterministic parser can reach the accepting configuration (with some rest graph) if and only if the current stack content is a viable prefix (Thm. 5.7). On the other hand, Thm. 6.12 shows that the nCFA approves precisely the viable prefixes. In other words, the nondeterministic parser can avoid running into a situation in which no rest graph could ever make it accept the input by making sure that its moves only produce stacks that are approved by the nCFA.

7. Deterministic Characteristic Finite-State Automata

Because of its spontaneously acting closure transitions, the nCFA cannot efficiently be used to improve the naïve shift-reduce parser by making sure that the stack of the parser is always a viable prefix. This is so because the nCFA, whenever it reaches a configuration, may also be in any configuration reachable by closure moves. In a deterministic implementation, all these configurations must be maintained simultaneously when the next goto move shall be made. To avoid this, we preprocess the nCFA and create the deterministic CFA (dCFA) by combining such simultaneously reachable states into new states, using a procedure similar to the classic powerset construction.

Literals of prefixes approved by an nCFA are images of literals of transitions and nCFA states under node mappings. The idea behind our adaptation of the traditional powerset construction to CFAs is to split such a node mapping into a composition of two mappings; the so-called *parameter mapping* is applied first and the so-called *input graph mapping* second: The parameter mapping maps each node of an nCFA state to a parameter, intuitively providing the node with a formal parameter name under which it can be addressed for instantiation. The input graph mapping performs this instantiation by mapping parameters to nodes of the actual input graph. The input graph mapping will be chosen when "approving" a graph with the dCFA, whereas the parameter mapping is chosen when constructing the dCFA. Different nodes that are always mapped to the same input graph nodes and that belong to nCFA states combined in a common dCFA state, are mapped to the same parameter by the algorithm. Let us use nodes to model parameters. An *item* is then an nCFA state with such a parameter mapping:

Definition 7.1 (Item). An *item* $\langle q, \sigma \rangle$ consists of an nCFA state $q = (\boldsymbol{A} \to \alpha \,\centerdot\, \beta)$ and an injective partial *parameter mapping* $\sigma \colon X \rightharpoonup X$ with $\mathit{dom}(\sigma) = X(\alpha)$. $\mathcal{I}$ denotes the set of all items.

Note that the parameter mapping maps only those nodes of the item which occur in literals preceding the dot.

If an nCFA processes a graph and reaches a state q, it can also reach those states reachable from q by closure transitions. Of course, nodes must be renamed appropriately, as we are dealing with items instead of pure states. An item that is reachable from another item by a closure transition is called *closure item* of the latter. The formal definition reads as follows:

Definition 7.2 (Closure of Items). We call $\langle q, \tau\rangle$ a *closure item* of $\langle p, \sigma\rangle$, written $\langle p, \sigma\rangle \rhd \langle q, \tau\rangle$, if $p = (\boldsymbol{A} \to \alpha \mathbin{.} \boldsymbol{B}\beta)$ and $q = (\boldsymbol{C} \to \mathbin{.} \delta)$, $\boldsymbol{C}^\tau = \boldsymbol{B}^\sigma$, and $dom(\tau) \subseteq X(\boldsymbol{C})$.

The *closure* of a set I of items is the smallest set J that contains all members of I and, for each item in J, all its closure items.

The closure for a given set of items can be computed in the usual way by adding all closure items to the set and repeating this procedure as long as new items are added to the set:

Fact 7.3. *Let J be the closure of a set I of items. Then, for every item $\langle q', \sigma'\rangle \in J$, there is a item $\langle q, \sigma\rangle \in I$ such that $\langle q, \sigma\rangle \rhd^* \langle q', \sigma'\rangle$.*

Before we give the formal definition of dCFAs, we define the notion of states used in it.

Definition 7.4 (dCFA state). A *dCFA state* Q is a finite set of items. It is *closed* if it is its own closure. We let $params(Q)$ denote the set $\bigcup\{\sigma(X) \mid \langle q, \sigma\rangle \in Q'\}$ of all *parameters* of Q.

A *concrete state* Q^τ of $\mathfrak{C}$ is a state $Q \in \mathcal{Q}$ under an input node mapping $\tau\colon params(Q) \to X$. The (infinite) set of all concrete states is denoted by $\mathcal{Q}_\mathrm{M}$.

For a renaming $\mu\colon X \to X$, let

$$Q^\mu = \{\langle q, \mu \circ \nu\rangle \mid \langle q, \nu\rangle \in Q\}\ [4]$$

be the state obtained by mapping each parameter by μ. Two states Q, Q' are equivalent, written $Q \approx Q'$, if there is a μ such that $Q^\mu = Q'$.

We are now ready to give the formal definition of dCFAs. Each dCFA state is a closed set of items. In particular, the initial state Q_0 is the closure of the initial state of the nCFA. Transitions in the dCFA are labeled with pairs that consist of a literal and a node mapping. The literal is the one that triggers the state transition; its nodes are parameters of the source state of the transition and new parameters whose "values" are the corresponding nodes of the consumed edge. The node mapping of the transition will later be used to set the "values" of the target state parameters.

Definition 7.5. A *deterministic characteristic finite automaton* (dCFA) $\mathfrak{C} = (\mathcal{Q}, Q_0, \Delta, Q_\mathsf{A})$ consists of a finite set $\mathcal{Q} \subseteq 2^{\mathcal{I}}$ of dCFA states, initial and final states $Q_0, Q_\mathsf{A} \in \mathcal{Q}$, and a transition relation $\Delta \subseteq \mathcal{Q} \times Lit_{\mathcal{T}} \times (X \rightharpoonup X) \times \mathcal{Q}$ with the following properties:

1. For all states $Q, Q' \in \mathcal{Q}$, $Q \approx Q'$ implies $Q = Q'$.
2. Q_0 is the closure of $\{\langle q_0, \iota\rangle\}$, where $q_0 = (\boldsymbol{Start} \to \mathbin{.})\boldsymbol{Z}$ is the initial state of the nCFA.

---

[4] Here, μ is a total function, but clearly we need only be concerned with defining μ for those parameters actually used.

Algorithm 1: Converting the nCFA $\mathfrak{A}$ into a deterministic CFA.

```
Input : Nondeterministic CFA $\mathfrak{A}$
Output: Equivalent deterministic CFA $\mathfrak{C}$
let $q_0 = (\boldsymbol{Z} \to \centerdot\,\zeta)$ be the initial state of $\mathfrak{A}$
compute $Q_0$ as the closure of $\{\langle q_0, \iota\rangle\}$ and let $\mathfrak{C}$ be the automaton with
    initial state $Q_0$ and no further states yet
$W \leftarrow \{Q_0\}$
while $W \neq \varnothing$ do
    select and remove any set $S$ from $W$
    foreach $\boldsymbol{l} \in leave(S)$ do
        obtain literal $\boldsymbol{e}$ from $\boldsymbol{l}$ by replacing each occurrence of '–' in $\boldsymbol{l}$ by
            a new node not used anywhere else
        $I \leftarrow \varnothing$
        foreach $\langle q, \sigma\rangle \in S$ with $\boldsymbol{l} \in leave(q, \sigma)$ do
            let $q = (\boldsymbol{A} \to \alpha\,\centerdot\,\boldsymbol{f}\beta)$
            let $\nu\colon X \rightharpoonup X$ be injective with $\sigma \sqsubseteq \nu$, $\boldsymbol{f}^\nu = \boldsymbol{e}$, and
                $dom(\nu) = dom(\sigma) \cup X(\boldsymbol{f})$
            add $\langle \boldsymbol{A} \to \alpha\boldsymbol{f}\,\centerdot\,\beta, \nu\rangle$ to $I$
        compute the closure $I'$ of $I$
        if $\mathfrak{C}$ has a state $Q \approx I'$ then
            add a transition $S \xrightarrow{(\boldsymbol{e},\mu)} Q$ to $\mathfrak{C}$ where $I' = Q^\mu$
        else
            add $I'$ as a new state to $\mathfrak{C}$ and $W$
            add a transition $S \xrightarrow{(\boldsymbol{e},\mathrm{id})} I'$ to $\mathfrak{C}$
```

3. $Q_\mathsf{A} = \{\langle \boldsymbol{Start} \to \boldsymbol{Z}\,\centerdot, \iota\rangle\}$.

We write $Q \xrightarrow{(\boldsymbol{e},\mu)} Q'$ if $(Q, \boldsymbol{e}, \mu, Q') \in \Delta$ where μ is an injective function $\mu\colon params(Q') \to X$.

Alg. 1 converts an nCFA into a corresponding dCFA. To determine the set of all transitions leaving a dCFA state S, it considers each element $\boldsymbol{l}$ of the set

$$leave(S) = \bigcup_{\langle q,\sigma\rangle \in S} leave(q, \sigma)$$

where $leave(\boldsymbol{A} \to \alpha\,\centerdot, \sigma) = \varnothing$ and $leave(\boldsymbol{A} \to \alpha\,\centerdot\,\boldsymbol{e}\beta, \sigma) = \{\boldsymbol{e}^\sigma\}$, i.e., $leave(S)$ contains mapped images of those literals that are labels of goto transitions leaving the corresponding nCFA states. These literals are mapped by the parameter mapping, i.e., they are in general pseudo-literals whose "nodes" are either parameters of S, or '–' if they are not (yet) mapped in Q.

Parameter names can be chosen arbitrarily as long as the parameter mappings are injective. That way, Alg. 1 frequently creates new sets of items which

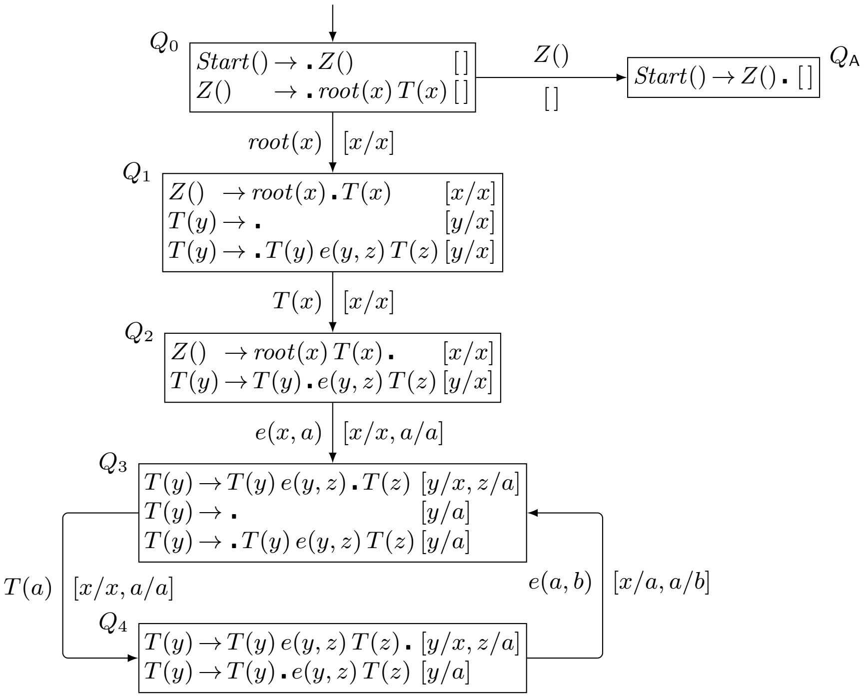

Figure 10: Deterministic CFA created by Alg. 1 from the nCFA in Fig. 8

should become states of the dCFA $\mathfrak{C}$, but are equivalent to sets that have already been added as states to $\mathfrak{C}$ and should thus not be added again. Alg. 1 avoids equivalent states (line 14) and "reuses" existing states instead (line 15). The node mapping being part of transition labels is the parameter renaming that must be applied to reuse an already existing state.

Example 7.6 (The dCFA for the Tree-Generating Grammar). Fig. 10 shows the dCFA obtained by Alg. 1 from the nondeterministic CFA (Fig. 8) for the tree grammar in Example 4.2. It consists of the states $Q_0, \ldots, Q_4, Q_\mathsf{A}$, which are sets of items. Each item $\langle q, \sigma \rangle$ is written in a single line that shows its nCFA state q on the left and its parameter mapping σ on the right (in brackets). Each pair u/v denotes that node u of the item is mapped to the parameter v, i.e., $\sigma(u) = v$; σ is undefined for all other nodes. Transitions are labelled by pairs of literals and node mappings. The latter are represented analogously.

Note that the transitions leading from Q_0 via Q_1 and Q_2 to Q_3 have identity node mappings because the target states of these transitions were new states when Alg. 1 created the transition (line 18). When it created the transition

$$\begin{aligned}
\varepsilon\blacklozenge Q_0 \quad &\mathrel{|\!\!\approx} & root(1)\blacklozenge Q_1^{x/1} \\
&\mathrel{|\!\!\approx} & root(1)\,T(1)\blacklozenge Q_2^{x/1} \\
&\mathrel{|\!\!\approx} & root(1)\,T(1)\,e(1,2)\blacklozenge Q_3^{x/1,a/2} \\
&\mathrel{|\!\!\approx} & root(1)\,T(1)\,e(1,2)\,T(2)\blacklozenge Q_4^{x/1,a/2} \\
&\mathrel{|\!\!\approx} & root(1)\,T(1)\,e(1,2)\,T(2)\,e(2,4)\blacklozenge Q_3^{x/1,a/2} \\
&\mathrel{|\!\!\approx} & root(1)\,T(1)\,e(1,2)\,T(2)\,e(2,4)\,T(4)\blacklozenge Q_4^{x/2,a/4}
\end{aligned}$$

Figure 11: Moves of the dCFA in Fig. 10. Renamings τ (or $\nu \circ \mu$, resp.) of states Q_i with $\tau(x_1) = y_1, \ldots, \tau(x_k) = y_k$ are represented by exponents $x_1/y/_1, \ldots, x_k/y_k$.

leaving Q_4, however, it constructed the set

$$\begin{aligned}
I' = \{\, &\langle T(y) \to T(y)\,e(y,z)\,\centerdot\,T(z), [y/a, z/b]\rangle, \\
&\langle T(y) \to \,\centerdot\,, [y/b]\rangle, \\
&\langle T(y) \to \,\centerdot\,T(y)\,e(y,z)\,T(z), [y/b]\rangle \,\}
\end{aligned}$$

of items, for some new parameter node b. But this set is equal to Q_3^μ where $\mu = [x/a, a/b]$ is the corresponding node mapping, and Alg. 1 has reused state Q_3 when adding the transition in line 15. ☐

Assumption 7.7. Since Alg. 1 constructs the dCFA from an nCFA, and as we have assumed a fixed HR grammar Γ and a fixed nCFA, we also assume a fixed dCFA $\mathfrak{C}$ in the following.

The dCFA approves graphs in a similar way as the nCFA does. We define dCFA configurations, dCFA moves, and approval by a dCFA analogously to their nCFA counterparts (Definitions 6.3, 6.4, and 6.6). The primary difference is that dCFA moves do not have to take closure moves into account:

Definition 7.8. A *dCFA configuration* $\varphi\blacklozenge Q^\tau$ consists of a graph $\varphi \in \mathcal{G}_\Sigma$ and a dCFA state Q mapped under an injective partial function $\tau\colon X \rightharpoonup X$.

A dCFA transition $t = (Q \xrightarrow{(\boldsymbol{e},\mu)} Q')$ turns $\varphi\blacklozenge Q^\tau$ into $\varphi\boldsymbol{e}^\nu\blacklozenge Q'^{\nu\circ\mu}$ with $\tau \sqsubseteq \nu$, $dom(\nu) = dom(\tau) \cup X(\boldsymbol{e})$, and $X(\boldsymbol{e}^\nu) \cap X(\varphi) \subseteq X(\boldsymbol{e}^\tau)$. We write such a *dCFA move* as $\varphi\blacklozenge Q^\tau \mathrel{|\!\!\approx}_t \varphi\boldsymbol{e}^\nu\blacklozenge Q'^{\nu\circ\mu}$ or simply $\varphi\blacklozenge Q^\tau \mathrel{|\!\!\approx} \varphi\boldsymbol{e}^\nu\blacklozenge Q'^{\nu\circ\mu}$.

The dCFA *approves* a graph $\varphi \in \mathcal{G}_\Sigma$ if there is a dCFA configuration $C = \varphi\blacklozenge Q^\tau$ such that $\varepsilon\blacklozenge Q_0^\iota \mathrel{|\!\!\approx}^* C$.

Note that the final state of the dCFA plays no role in the proceedings so far. This will continue to be so for the rest of this section, but change in the next.

Example 7.9 (Moves of the dCFA for the Tree-Generating Grammar). Fig. 11 shows moves of the deterministic CFA in Fig. 10. They approve the same graph as the moves (shown in Fig. 9) of the nondeterministic CFA of Fig. 8. ☐

To prove that the dCFA approves the same graphs as the nCFA, we need the following lemma. It shows that constructing closure items is tightly related to performing closure moves.

Lemma 7.10. *$\langle p, \sigma\rangle \rhd^* \langle q, \tau\rangle$ implies $\varphi\diamond[p]^{\mu\circ\sigma} \underset{\mathsf{cl}}{\rightsquigarrow}^* \varphi\diamond[q]^{\mu\circ\tau}$ for all items $\langle p, \sigma\rangle$ and $\langle q, \tau\rangle$, every injective partial function $\mu\colon X \rightharpoonup X$ with $\mathit{dom}(\mu\circ\sigma) = \mathit{dom}(\sigma)$, and every graph $\varphi \in \mathcal{G}_\Sigma$ such that $\varphi\diamond[p]^{\mu\circ\sigma}$ is a valid nCFA configuration.*

Proof. Consider any items $\langle p, \sigma\rangle$ and $\langle q, \tau\rangle$ such that $\langle p, \sigma\rangle \rhd^n \langle q, \tau\rangle$, and μ as well as φ as in the lemma. We prove $\mathit{dom}(\mu \circ \tau) = \mathit{dom}(\tau)$ and $\varphi\diamond[p]^{\mu\circ\sigma} \underset{\mathsf{cl}}{\rightsquigarrow}^n \varphi\diamond[q]^{\mu\circ\tau}$ by induction over n.

For $n = 0$, $\langle p, \sigma\rangle = \langle q, \tau\rangle$, and therefore, $\varphi\diamond[p]^{\mu\circ\sigma} = \varphi\diamond[q]^{\mu\circ\tau}$ as well as $\mathit{dom}(\mu \circ \tau) = \mathit{dom}(\tau)$, which proves the proposition.

For $n > 0$, we have $\langle p, \sigma\rangle \rhd^{n-1} \langle p', \sigma'\rangle \rhd \langle q, \tau\rangle$, and by the induction hypothesis, $\varphi\diamond[p]^{\mu\circ\sigma} \underset{\mathsf{cl}}{\rightsquigarrow}^{n-1} \varphi\diamond[p']^{\mu\circ\sigma'}$ and $\mathit{dom}(\mu\circ\sigma') = \mathit{dom}(\sigma')$. Then p' and q are of the form $p' = (\boldsymbol{A} \to \alpha \centerdot \boldsymbol{B}\beta)$, $q = (\boldsymbol{C} \to \centerdot\, \delta)$, $\boldsymbol{C}^\tau = \boldsymbol{B}^{\sigma'}$, $\mathit{dom}(\tau) \subseteq X(\boldsymbol{C})$.

We first show that $\mathit{dom}(\mu\circ\tau) = \mathit{dom}(\tau)$. The inclusion $\mathit{dom}(\mu\circ\tau) \subseteq \mathit{dom}(\tau)$ follows immediately from the definition of the composition of partial functions. In order to show the opposite inclusion, consider any node $x \in \mathit{dom}(\tau) \subseteq X(\boldsymbol{C})$. There must be a node $y \in X(\boldsymbol{B})$ such that $\tau(x) = \sigma'(y)$ because $\boldsymbol{C}^\tau = \boldsymbol{B}^{\sigma'}$, and therefore $y \in \mathit{dom}(\sigma') = \mathit{dom}(\mu \circ \sigma')$. In other words, $\tau(x) = \sigma'(y) \in \mathit{dom}(\mu)$, i.e., $x \in \mathit{dom}(\mu \circ \tau)$, which proves $\mathit{dom}(\tau) \subseteq \mathit{dom}(\mu \circ \tau)$.

As a consequence, the equalities $\boldsymbol{C}^{\mu\circ\tau} = (\boldsymbol{C}^\tau)^\mu = (\boldsymbol{B}^{\sigma'})^\mu = \boldsymbol{B}^{\mu\circ\sigma'}$ and $\mathit{dom}(\mu\circ\tau) = \mathit{dom}(\tau) \subseteq X(\boldsymbol{C})$ hold. As $\varphi\diamond[p']^{\mu\circ\sigma'}$ is a valid nCFA configuration, and by Def. 6.4, $\varphi\diamond[p']^{\mu\circ\sigma'} \underset{\mathsf{cl}}{\rightsquigarrow} \varphi\diamond[q]^{\mu\circ\tau}$. □

Lemma 7.11 shows that each graph approved by the nCFA is also approved by the dCFA, and Lemma 7.12 shows the opposite direction. But these lemmata are even more specific: Each dCFA state is a set of items, i.e., nCFA states with parameter mappings, and one can find an approving sequence of nCFA states "within" the corresponding approving sequence of nCFA states. In other words, the relation between the two automata is similar to that between an ordinary nondeterministic finite automaton and its powerset automaton.

Lemma 7.11. *For each sequence*

$$\varepsilon\diamond[q_0]^\iota \rightsquigarrow^* \varphi\diamond[q]^\varrho$$

of nCFA moves, there is a dCFA state Q and an injective partial function $\tau\colon X \rightharpoonup X$ such that $\langle q, \varrho\rangle \in Q^\tau$ and

$$\varepsilon\blacklozenge Q_0^\iota \approx\!\!\!\rightsquigarrow^* \varphi\blacklozenge Q^\tau.$$

Proof. We prove the proposition by induction over the number n of moves in $\varepsilon\diamond[q_0]^\iota \rightsquigarrow^n \varphi\diamond[q]^\varrho$. For $n = 0$, the proposition follows immediately from the definition of initial nCFA configurations and Q_0.

For $n > 0$ and the last move being a closure move, the considered sequence of moves is of the form

$$\varepsilon\Diamond[q_0]^\iota \mathrel{\vdash\!\!\frown}^{n-1} \varphi\Diamond[\boldsymbol{A} \to \alpha \centerdot \boldsymbol{B}\beta]^\kappa \mathrel{\underset{\mathsf{cl}}{\vdash\!\!\frown}} \varphi\Diamond[\boldsymbol{C} \to {\centerdot}\,\delta]^\varrho$$

with $\boldsymbol{B}^\kappa = \boldsymbol{C}^\varrho$ and $dom(\varrho) \subseteq X(\boldsymbol{C})$. By the induction hypothesis, there is a dCFA state Q and an injective partial function $\tau\colon X \rightharpoonup X$ such that $\varepsilon\blacklozenge Q_0^\iota \mathrel{\models\!\!\approx}^* \varphi\blacklozenge Q^\tau$ and $\langle \boldsymbol{A} \to \alpha \centerdot \boldsymbol{B}\beta, \kappa\rangle \in Q^\tau$. Therefore, there is an injective $\eta\colon X \rightharpoonup X$ with $\kappa = \tau \circ \eta$ and $\langle \boldsymbol{A} \to \alpha \centerdot \boldsymbol{B}\beta, \eta\rangle \in Q$. Since each dCFA state is closed under closure (line 2 and 13), we also have $\langle \boldsymbol{C} \to {\centerdot}\,\delta, \xi\rangle \in Q$ with $\boldsymbol{B}^\eta = \boldsymbol{C}^\xi$ and $dom(\xi) \subseteq X(\boldsymbol{C})$. Therefore, $\boldsymbol{C}^\varrho = \boldsymbol{B}^\kappa = (\boldsymbol{B}^\eta)^\tau = (\boldsymbol{C}^\xi)^\tau$. And because of injectivity, $\varrho = \tau \circ \xi$, and therefore $\langle \boldsymbol{C} \to {\centerdot}\,\delta, \varrho\rangle \in Q^\tau$.

For $n > 0$ and the last move being a goto move, the considered sequence of moves is of the form

$$\varepsilon\Diamond[q_0]^\iota \mathrel{\vdash\!\!\frown}^{n-1} \varphi\Diamond[\boldsymbol{A} \to \alpha \centerdot \boldsymbol{f}\beta]^\kappa \mathrel{\underset{\mathsf{go}}{\vdash\!\!\frown}} \varphi\boldsymbol{f}^\varrho\Diamond[\boldsymbol{A} \to \alpha\boldsymbol{f} \centerdot \beta]^\varrho$$

with

$$\kappa \sqsubseteq \varrho,\ dom(\varrho) = dom(\kappa) \cup X(\boldsymbol{f}),\ \text{and } X(\boldsymbol{f}^\varrho) \cap X(\varphi) \subseteq X(\boldsymbol{f}^\kappa). \tag{21}$$

By the induction hypothesis, there is a dCFA state S and an injective $\chi\colon X \rightharpoonup X$ such that $\varepsilon\blacklozenge Q_0^\iota \mathrel{\models\!\!\approx}^* \varphi\blacklozenge S^\chi$ and $\langle \boldsymbol{A} \to \alpha \centerdot \boldsymbol{f}\beta, \kappa\rangle \in S^\chi$. Therefore, there is an injective $\sigma\colon X \rightharpoonup X$ with $\kappa = \chi \circ \sigma$, $\langle \boldsymbol{A} \to \alpha \centerdot \boldsymbol{f}\beta, \sigma\rangle \in S$, and $\boldsymbol{f}^\sigma \in leave(S)$. (Note that the identifiers used here match those in Alg. 1.) Alg. 1, therefore, obtained a literal $\boldsymbol{e}$ from $\boldsymbol{f}^\sigma$ by replacing each occurrence of '–' by a new node not used anywhere else (line 11). It also obtained an injective partial function ν with

$$\sigma \sqsubseteq \nu,\ \boldsymbol{f}^\nu = \boldsymbol{e},\ \text{and } dom(\nu) = dom(\sigma) \cup X(\boldsymbol{f}). \tag{22}$$

And Alg. 1 has added a transition $S \xrightarrow{(\boldsymbol{e},\mu)} Q$ to $\mathfrak{C}$ (line 15 or 18) by constructing a set I' of items such that $\langle \boldsymbol{A} \to \alpha\boldsymbol{f} \centerdot \beta, \nu\rangle \in I'$ (line 12 and 13) and $Q^\mu = I'$. By the construction of $\boldsymbol{e}$, by (21) as well as (22), and because μ is injective, there is an injective $\xi\colon X \rightharpoonup X$ such that $\chi \sqsubseteq \xi$ and $\varrho = \xi \circ \nu$, and therefore $\langle \boldsymbol{A} \to \alpha\boldsymbol{f} \centerdot \beta, \varrho\rangle \in I'^\xi = Q^{\xi\circ\mu}$. Since $\boldsymbol{f}^\varrho = \boldsymbol{e}^\xi$, we can conclude $\varphi\blacklozenge S^\chi \mathrel{\models\!\!\approx} \varphi\boldsymbol{f}^\varrho\blacklozenge Q^{\xi\circ\mu}$. Thus, the lemma holds with $\tau = \xi \circ \mu$. ☐

Lemma 7.12. *For each sequence*

$$\varepsilon\blacklozenge Q_0^\iota \mathrel{\models\!\!\approx}^* \varphi\blacklozenge Q^\tau$$

and each item $\langle q, \vartheta\rangle \in Q^\tau$*, there exists a sequence*

$$\varepsilon\Diamond[q_0]^\iota \mathrel{\vdash\!\!\frown}^* \varphi\Diamond[q]^\vartheta$$

of nCFA moves.

Proof. Let $\varepsilon\blacklozenge Q_0^\iota \models\!\!\approx^n \varphi\blacklozenge Q^\tau$ be any sequence of dCFA moves and $\langle q,\vartheta\rangle$ any item with $\langle q,\vartheta\rangle \in Q^\tau$. We prove $\varepsilon\lozenge[q_0]^\iota \vdash\!\!\sim^* \varphi\lozenge[q]^\vartheta$ by induction over n.

For $n = 0$, the proposition follows from $Q = Q_0$, $\varphi = \varepsilon$, $\tau = \iota$, and therefore $Q^\tau = \{\langle q_0,\iota\rangle\}$, i.e., $q = q_0$ and $\vartheta = \iota$.

For $n > 0$, there is a sequence of moves

$$\varepsilon\blacklozenge Q_0^\iota \models\!\!\approx^{n-1} \varphi\blacklozenge S^\chi \models\!\!\approx \varphi\boldsymbol{e}^\xi\blacklozenge Q^{\xi\circ\mu}$$

with $\tau = \xi\circ\mu$, $\chi \sqsubseteq \xi$, $\mathit{dom}(\xi) = \mathit{dom}(\chi)\cup X(\boldsymbol{e})$ and the last move using transition $S \xrightarrow{(\boldsymbol{e},\mu)} Q$. Alg. 1 added this transition to $\mathfrak{C}$ after computing a set Q^μ. (Note that the identifiers used here match those in Alg. 1.) As $\langle q,\vartheta\rangle \in Q^{\xi\circ\mu}$, there is a π with $\vartheta = \xi\circ\pi$ and $\langle q,\pi\rangle \in Q^\mu = I'$. Since I' was computed as the closure of I (line 13), and by Fact 7.3, there is a item $\langle \boldsymbol{A} \to \alpha\boldsymbol{f}\centerdot\beta,\nu\rangle \in I$ that was added to I in line 12, and

$$\langle \boldsymbol{A} \to \alpha\boldsymbol{f}\centerdot\beta,\nu\rangle \rhd^* \langle q,\pi\rangle. \tag{23}$$

In fact, $\langle \boldsymbol{A} \to \alpha\boldsymbol{f}\centerdot\beta,\nu\rangle$ was added to I after selecting a item $\langle \boldsymbol{A} \to \alpha\centerdot\boldsymbol{f}\beta,\sigma\rangle \in S$, which was been turned into $\langle \boldsymbol{A} \to \alpha\boldsymbol{f}\centerdot\beta,\nu\rangle$. Literal $\boldsymbol{e}$ was obtained from $\boldsymbol{f}^\sigma$ by replacing each occurrence of '–' by a new node not used anywhere else, and the injective partial function $\nu\colon X \rightharpoonup X$ was chosen such that $\sigma \sqsubseteq \nu$, $\boldsymbol{f}^\nu = \boldsymbol{e}$, and $\mathit{dom}(\nu) = \mathit{dom}(\sigma)\cup X(\boldsymbol{f})$. By the induction hypothesis, $\varepsilon\lozenge[q_0]^\iota \vdash\!\!\sim^* \varphi\lozenge[q]^\kappa$ for each item $\langle q,\kappa\rangle \in S^\chi$, and in particular for $\langle \boldsymbol{A} \to \alpha\centerdot\boldsymbol{f}\beta,\kappa\rangle \in S^\chi$ with $\kappa = \chi\circ\sigma$. Let us define $\varrho = \xi\circ\nu$, and therefore $\langle \boldsymbol{A} \to \alpha\boldsymbol{f}\centerdot\beta,\varrho\rangle \in I^\xi \subseteq I'^\xi = Q^{\xi\circ\mu}$ and $\boldsymbol{f}^\varrho = \boldsymbol{e}^\xi$. By this construction, $\kappa \sqsubseteq \varrho$, $\mathit{dom}(\varrho) = \mathit{dom}(\kappa)\cup X(\boldsymbol{f})$, and $X(\boldsymbol{f}^\varrho)\cap X(\varphi) \subseteq X(\boldsymbol{f}^\kappa)$, and therefore

$$\varphi\lozenge[\boldsymbol{A} \to \alpha\centerdot\boldsymbol{f}\beta]^\kappa \mathrel{\vdash\!\!\sim}_{\mathsf{go}} \varphi\boldsymbol{f}^\varrho\lozenge[\boldsymbol{A} \to \alpha\boldsymbol{f}\centerdot\beta]^\varrho = \varphi\boldsymbol{e}^\xi\lozenge[\boldsymbol{A} \to \alpha\boldsymbol{f}\centerdot\beta]^{\xi\circ\nu}.$$

By the construction of ξ and ν, we have $\mathit{dom}(\xi\circ\nu) = \mathit{dom}(\nu)$. And, by (23) and Lemma 7.10,

$$\varphi\boldsymbol{e}^\xi\lozenge[\boldsymbol{A} \to \alpha\boldsymbol{f}\centerdot\beta]^{\xi\circ\nu} \mathrel{\vdash\!\!\sim}_{\mathsf{cl}}^* \varphi\boldsymbol{e}^\xi\lozenge[q]^{\xi\circ\pi} = \varphi\boldsymbol{e}^\xi\lozenge[q]^\vartheta.$$

which completes the proof of the lemma. ☐

An immediate consequence of these lemmata is the following:

Theorem 7.13. *A graph is approved by the dCFA if and only if it is approved by the nCFA if and only if it is a viable prefix.*

Thm. 7.13 implies that the nondeterministic parser can reach the accepting configuration with some rest graph if and only if the dCFA approves its current stack.

Before we describe assisted shift-reduce parsers using deterministic CFA (Sect. 8), let us observe that Alg. 1 does not terminate for all HR grammars. A HR grammar for the visual language of *structured flowcharts* is used to demonstrate this.

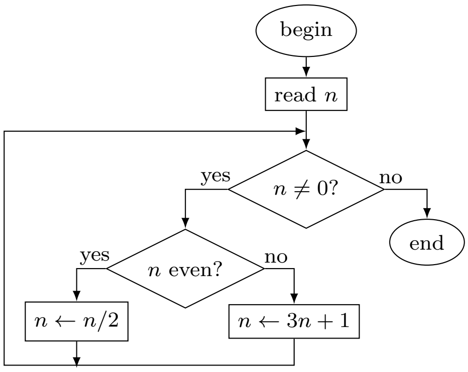

Figure 12: A structured flowchart.

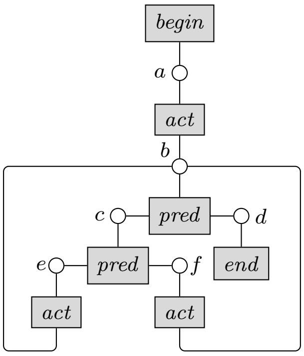

Figure 13: Graph representation of the structured flowchart in Fig. 12.

Example 7.14 (Structured Flowcharts). Structured flowcharts consist of rectangles containing actions, diamonds that indicate conditions, and ovals indicating begin and end of the program. Arrows indicate control flow; see Fig. 12 for an example. They can be represented by graphs using terminal symbols *begin*, *end*, *act*, and *pred* where binary *act* edges represent actions (rectangles) and ternary *pred* edges conditions (diamonds). Nodes correspond to arrows where edges are attached to the same node if the corresponding components (rectangle, diamond, or oval) are connected by an arrow. The example flowchart in Fig. 12 can be represented by the graph shown in Fig. 13 and by the (ordered) graph

$$\mathit{begin}(a)\ \mathit{act}(a,b)\ \mathit{pred}(b,c,d)\ \mathit{pred}(c,e,f)\,\mathit{act}(e,b)\ \mathit{act}(f,b)\ \mathit{end}(d)\,.$$

For instance, literal $\mathit{act}(a,b)$ represents the rectangle "read n", $\mathit{act}(f,b)$ the rectangle "$n \leftarrow 3n+1$", and $\mathit{pred}(c,e,f)$ the diamond "n even?". An HR grammar for the graph representation of structured flowcharts has nonterminal symbols Z, *Seq* (for "sequence"), as well as *Stmt* (for "statement") and the following rules:

$$\begin{aligned}
Z() &\to \mathit{begin}(x)\ \mathit{Seq}(x,y)\ \mathit{end}(y) && \\
\mathit{Seq}(x,y) &\to \mathit{Stmt}(x,y) && \text{(end of sequence)}\\
\mathit{Seq}(x,y) &\to \mathit{Stmt}(x,z)\ \mathit{Seq}(z,y) && \text{(sequence)}\\
\mathit{Stmt}(x,y) &\to \mathit{act}(x,y) && \text{(single action)}\\
\mathit{Stmt}(x,y) &\to \mathit{pred}(x,z,y)\ \mathit{Seq}(z,x) && \text{(while loop)}\\
\mathit{Stmt}(x,y) &\to \mathit{pred}(x,u,v)\ \mathit{Seq}(u,y)\ \mathit{Seq}(v,y) && \text{(selection)}
\end{aligned}$$

The dCFA of this grammar is infinite, i.e., Alg. 1 does not terminate. To see this, consider the excerpt of the dCFA in Fig. 14. In order to save space, we use a compact notation. If a dCFA state contains items $\langle q,\sigma_1\rangle,\dots,\langle q,\sigma_n\rangle$ that share the nCFA state q, we write $q\{\sigma_1,\dots,\sigma_n\}$ where the parameter mappings $\sigma_1,\dots,\sigma_n$ are denoted as introduced earlier. And, if a dCFA state contains

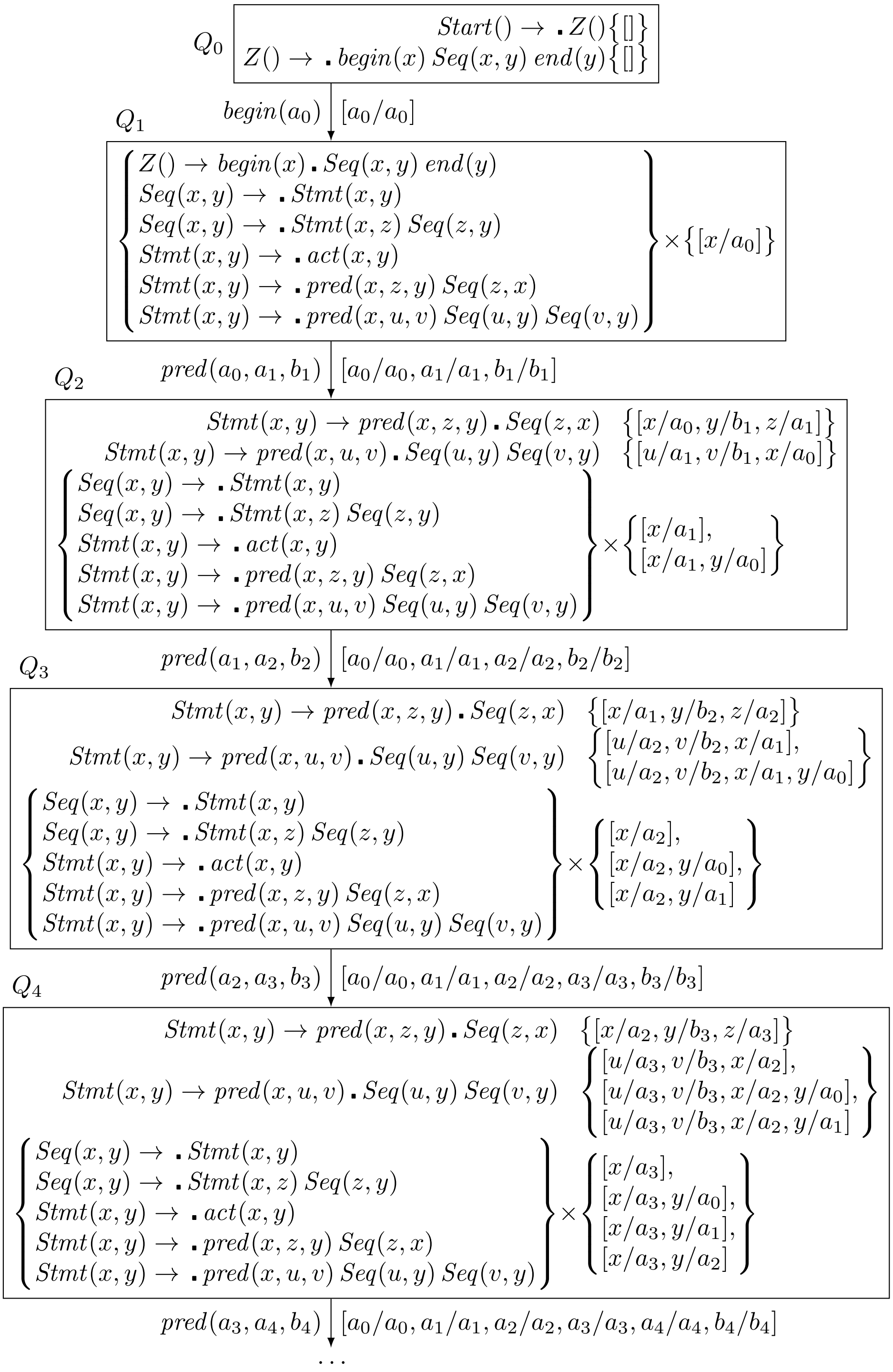

Figure 14: Excerpt of the infinite dCFA of the flowchart grammar.

the full Cartesian product of a set Q of nCFA states and a set P of parameter mappings, we write $Q \times P$. This notation even allows to represent Q_4 with its 24 items.

Fig. 14 shows the states $Q_0, \ldots, Q_4$ of the dCFA and only the transitions between them. One can see that Q_2, Q_3, and Q_4 are identical when ignoring the parameter mappings. Moreover, when renaming parameters, Q_2 is properly contained in Q_3, which in turn is properly contained in Q_4. These three states are the first states of an infinite sequence $Q_2, Q_3, Q_4, \ldots$, which makes the entire dCFA infinite.

As a consequence, Alg. 1 cannot be applied to all HR grammars and their nCFAs. However, a modified algorithm not described here can recognize and handle this situation. For this, it represents the infinite dCFA in a finite way by equipping states with variables that may contain sets of arbitrarily many parameters and using these variables in transitions. This algorithm has been implemented in the *Grappa* tool. In this paper, we have described the simpler algorithm in Alg. 1 instead of the more general one, because the latter is rather technical. In fact, the HR grammar for structured flowcharts is the only HR grammar with an infinite dCFA known to the authors—and it is not even PSR parsable because the finitely represented infinite dCFA has shift-reduce conflicts (see Table 1). We describe the concept of conflicts in Sect. 9.

8. CFA-Assisted Shift-Reduce Parsing

Let us now discuss how the naïve shift-reduce parser discussed in Sect. 4 can read off all permissible moves from the current dCFA state in order to reach the accepting configuration with some rest graph. Recall that the naïve parser just maintains a stack of literals. The extended parser, instead, maintains a stack as an alternating sequence of states and literals and makes sure that its stack (when ignoring the states on the stack) is always approved by the dCFA. The top stack element is always the current state, which is the uniquely determined dCFA state reached when approving the stack—when ignoring the states on the stack. The stack prior to a move is called *current* stack, and the next one is the *successor* stack, thus defining a *successor state*. The successor stack together with the successor state will then be the current stack and the current state, respectively, at the next move.

When performing a *shift move*, the parser selects a literal from the remaining input that matches the label of a transition leaving the current state (on top of the stack). This literal is then pushed onto the stack, together with the successor state reachable by this transition. The successor stack thus consist of the current stack, followed by the shifted literal and the successor state.

A *reduce move* removes the right-hand side (under an appropriate mapping) of a rule from the stack, together with the corresponding states, yielding some intermediate stacks with a state on top. The parser then selects a transition leaving this state and with a label matching the reduced nonterminal literal, which is the left-hand side of the rule, under the same mapping as above. Next, the reduced literal is pushed onto the stack, together with the successor state reachable by this transition. The successor stack thus consist of the intermediate stack, followed by the reduced literal and the successor state.

The parser *accepts* the input graph processed so far when the state on top of the stack is the final state Q_{A} of the dCFA. Note that the entire input graph is accepted that way if there are no unprocessed input literals left when reaching this state.

Let us now define the extension of the naïve shift-reduce parser more precisely. We call this parser *dCFA-assisted shift-reduce parser* or simply *assisted shift-reduce parser*, abbreviated as *ASR parser*.

Definition 8.1 (ASR Parser). An *(ASR parser) configuration* $\mathcal{S} \mathbin{\blacksquare} g$ consists of a *parse stack* $\mathcal{S} \subseteq \mathcal{Q}_{\mathrm{M}} \cdot (\mathit{Lit}_\Sigma \cdot \mathcal{Q}_{\mathrm{M}})^*$ and a graph $g \in \mathcal{G}_{\mathcal{T}}$. Thus, $\mathit{top}(\mathcal{S})$ is always a concrete state.[5] The graph obtained by removing all concrete states from $\mathcal{S}$ is denoted by $\mathit{graph}(\mathcal{S})$. A configuration $\mathcal{S} \mathbin{\blacksquare} g$ is *accepting* if $\mathit{top}(\mathcal{S})$ is the final state Q_{A} of the dCFA.

An *ASR move* turns $\mathcal{S} \mathbin{\blacksquare} g$ into $\mathcal{S}' \mathbin{\blacksquare} g'$ and is either an ASR shift move or an ASR reduce move, defined as follows.

Let $Q^\tau = \mathit{top}(\mathcal{S})$ for a state $Q \in \mathcal{Q}$ and an input node mapping τ.

- Suppose that there is a literal $\boldsymbol{e} \in \mathit{Lit}_{\mathcal{T}}$ and a concrete state $T \in \mathcal{Q}_{\mathrm{M}}$ such that $\mathit{graph}(\mathcal{S}) \blacklozenge Q^\tau \underset{tr}{\mathrel{|\!\!\approx}} \mathit{graph}(\mathcal{S}) \boldsymbol{e} \blacklozenge T$ and $X(\boldsymbol{e}) \cap X(g) \subseteq X(\mathit{graph}(\mathcal{S}))$. Then there is an *ASR shift move* $\mathcal{S} \mathbin{\blacksquare} g \underset{tr}{\models} \mathcal{S} \boldsymbol{e} T \mathbin{\blacksquare} g \boldsymbol{e}$.
- Suppose that Q contains an item $\mathit{it} = \langle \boldsymbol{A} \to \varrho \,\blacksquare, \sigma \rangle$ and one can remove $2 \cdot |\varrho|$ elements from the top of $\mathcal{S}$ to obtain a parse stack $\mathcal{S}''$ with $R = \mathit{top}(\mathcal{S}'')$ such that there exists a concrete state $T \in \mathcal{Q}_{\mathrm{M}}$ with $\mathit{graph}(\mathcal{S}'') \blacklozenge R \mathrel{|\!\!\approx} \mathit{graph}(\mathcal{S}'') \boldsymbol{A}^{\tau \circ \sigma} \blacklozenge T$. Then there is an *ASR reduce move* $\mathcal{S} \mathbin{\blacksquare} g \underset{it}{\models} \mathcal{S}'' \boldsymbol{A}^{\tau \circ \sigma} T \mathbin{\blacksquare} g$.

We may write $\mathcal{S} \mathbin{\blacksquare} g \models \mathcal{S}' \mathbin{\blacksquare} g'$ if $\mathcal{S} \mathbin{\blacksquare} g \underset{tr}{\models} \mathcal{S}' \mathbin{\blacksquare} g'$ for a transition tr or $\mathcal{S} \mathbin{\blacksquare} g \underset{it}{\models} \mathcal{S}' \mathbin{\blacksquare} g'$ for an item *it*.

A configuration $\mathcal{S} \mathbin{\blacksquare} g$ is *reachable* if $Q_0^\iota \mathbin{\blacksquare} \varepsilon \models^* \mathcal{S} \mathbin{\blacksquare} g$. An ASR parser *accepts* a graph $g \in \mathcal{G}_{\mathcal{T}}$ if there is a reachable accepting configuration $\mathcal{S} \mathbin{\blacksquare} g$.

Note that shift and reduce moves of the ASR parser always push (concrete) states onto the stack that are reachable from their immediate predecessor states on the stack. This is expressed in the following fact:

Fact 8.2. *$Q_0^\iota \mathbin{\blacksquare} \varepsilon \models^* \mathcal{S} \mathbin{\blacksquare} g$ implies $\varepsilon \blacklozenge Q_0^\iota \mathrel{|\!\!\approx}^* \mathit{graph}(\mathcal{S}') \blacklozenge \mathit{top}(\mathcal{S}')$ for every ASR parser configuration $\mathcal{S} \mathbin{\blacksquare} g$ and every parse stack $\mathcal{S}'$ being a prefix of $\mathcal{S}$.*

---

[5] Recall from Section 2 that $\mathit{top}(\mathcal{S})$ denotes the top of the stack, i.e., its rightmost element.

	stack	▪	consumed input	match
	Q_0	▪	ε	
$\models$	$Q_0\mathit{root}^1Q^1_{1\_}$	▪	root^1	
$\models_{3}$	$Q_0\mathit{root}^1Q^1_1T^1Q^1_2$	▪	root^1	$y/1$
$\models$	$Q_0\mathit{root}^1Q^1_1T^1Q^1_2e^{12}Q^{12}_3$	▪	root^1e^{12}	
$\models_{3}$	$Q_0\mathit{root}^1Q^1_1T^1Q^1_2e^{12}Q^{12}_3T^2Q^{12}_4$	▪	root^1e^{12}	$y/2$
$\models$	$Q_0\mathit{root}^1Q^1_1T^1Q^1_2e^{12}Q^{12}_3T^2Q^{12}_4e^{24}Q^{24}_{3\_}$	▪	$\mathit{root}^1e^{12}e^{24}$	$x/2, y/4$
$\models_{3}$	$Q_0\mathit{root}^1Q^1_1T^1Q^1_2e^{12}Q^{12}_3\underline{T^2Q^{12}_4e^{24}Q^{24}_3T^4Q^{24}_4}$	▪	$\mathit{root}^1e^{12}e^{24}$	$y/4$
$\models_{2}$	$Q_0\mathit{root}^1Q^1_1\underline{T^1Q^1_2e^{12}Q^{12}_3T^2Q^{12}_4}$	▪	$\mathit{root}^1e^{12}e^{24}$	$x/2, y/4$
$\models_{2}$	$Q_0\mathit{root}^1Q^1_1T^1Q^1_2$	▪	$\mathit{root}^1e^{12}e^{24}$	$x/1, y/2$
$\models$	$Q_0\mathit{root}^1Q^1_1T^1Q^1_2e^{13}Q^{13}_{3\_}$	▪	$\mathit{root}^1e^{12}e^{24}e^{13}$	
$\models_{3}$	$Q_0\mathit{root}^1Q^1_1\underline{T^1Q^1_2e^{13}Q^{13}_3T^3Q^{13}_4}$	▪	$\mathit{root}^1e^{12}e^{24}e^{13}$	$y/3$
$\models_{2}$	$Q_0\underline{\mathit{root}^1Q^1_1T^1Q^1_2}$	▪	$\mathit{root}^1e^{12}e^{24}e^{13}$	$x/1, y/3$
$\models_{1}$	$Q_0\ \mathit{Start}\ Q_{\mathsf{A}}$	▪	$\mathit{root}^1e^{12}e^{24}e^{13}$	

Figure 15: Moves of the ASR parser recognizing the tree in Example 4.2. Places on the stack where reductions occur are underlined. Rules used in reduce moves are indicated as subscripts in the leftmost column, and their corresponding matches appear in the rightmost column.

Example 8.3 (An ASR Parse of a Tree). Fig. 15 shows the moves of the ASR parser when recognizing the tree in Example 4.2.[6]

Moves no. 1 to 6 in Fig. 15 correspond to the moves of the dCFA shown in Fig. 11 of Example 7.9 in the way stated in Fact 8.2:

- The initial configuration of the ASR parser agrees with the initial state of the dCFA.
- The symbol and state pushed in move i agrees with the symbol approved, and the state reached, by move i of the dCFA. In three steps, terminal symbols are pushed by a shift move; the other moves push the nonterminal T in the course of reducing rule 2, which has no literals on its right-hand side so that nothing has to be popped off the stack.
- After move i, the symbols on the stack of the parser (ignoring the states) agree with the viable prefix approved in move i of the dCFA. ☐

Note also that a reduce move of the naïve shift-reduce parser must check a rather complex condition in order to select a reduce move (Def. 4.4); it must

---

[6] In this example and in Fig. 15, we use following abbreviated notation: literals $\ell(x_1, \ldots, x_k)$ are denoted as $\ell^{x_1\cdots x_k}$, and states of the dCFA that were written as $Q_i^{[x_1/y_1,\ldots,x_k/y_k]}$ in Fig. 11 are abbreviated as $Q_i^{y_1\cdots y_k}$.

examine whether the stack contains the right hand side of the rule (under an appropriate match), and it must additionally check condition $X(\alpha) \cap X(\varrho^{\mu}) \subseteq X(\boldsymbol{A}^{\mu})$ of Def. 4.4 to make sure that the corresponding derivation step is valid. The ASR parser, instead, just inspects the top state on the stack and checks whether this state contains an item with the dot at the end of the rule; it can thus read off from the dCFA whether it can select a reduce move. The following lemma states this formally. It will be used for proving the correctness of the ASR parser later.

Lemma 8.4. *For every rule* $\boldsymbol{A} \to \varrho$ *with* $\boldsymbol{A} \neq \boldsymbol{\mathit{Start}}$ *and every sequence* $\varepsilon\blacklozenge Q_0^{\iota} \mathrel{\vDash\!\!\!\approx}^{*} \varphi\blacklozenge Q^{\tau}$*,* Q *contains an item* $\langle \boldsymbol{A} \to \varrho\centerdot, \sigma\rangle$ *if and only if there is a graph* $\alpha \in \mathcal{G}_{\Sigma}$ *such that* $\varphi = \alpha\varrho^{\tau\circ\sigma}$*,* $\alpha\boldsymbol{A}^{\tau\circ\sigma}$ *is also approved by the dCFA, and* $X(\alpha) \cap X(\varrho^{\tau\circ\sigma}) \subseteq X(\boldsymbol{A}^{\tau\circ\sigma})$*.*

Proof. For the *only-if* direction, consider a sequence $\varepsilon\blacklozenge Q_0^{\iota} \mathrel{\vDash\!\!\!\approx}^{*} \varphi\blacklozenge Q^{\tau}$ and an item $\langle \boldsymbol{A} \to \varrho\centerdot, \sigma\rangle \in Q$. We have $\langle \boldsymbol{A} \to \varrho\centerdot, \mu\rangle \in Q^{\tau}$ with $\mu = \tau \circ \sigma$, and by Lemma 7.12, there is a sequence

$$\varepsilon\Diamond[q_0]^{\iota} \mathrel{\vdash\!\!\!\sim}^{*} \varphi\Diamond[\boldsymbol{A} \to \varrho\centerdot]^{\mu}. \tag{24}$$

The dot in $\boldsymbol{A} \to \varrho\centerdot$ must have been moved there by goto moves, starting at $\boldsymbol{A} \to {\centerdot}\varrho$, a state that was reached by a closure move. Therefore, (24) has the form

$$\varepsilon\Diamond[q_0]^{\iota} \mathrel{\vdash\!\!\!\sim}^{*} \alpha\Diamond[\boldsymbol{B} \to \gamma\centerdot\boldsymbol{C}\delta]^{\nu} \mathrel{\vdash\!\!\!\sim}_{\mathsf{cl}} \alpha\Diamond[\boldsymbol{A} \to {\centerdot}\varrho]^{\mu'} \mathrel{\vdash\!\!\!\sim}_{\mathsf{go}}^{*} \alpha\varrho^{\mu}\Diamond[\boldsymbol{A} \to \varrho\centerdot]^{\mu}$$

with $\varphi = \alpha\varrho^{\mu} = \alpha\varrho^{\tau\circ\sigma}$, $\boldsymbol{C}^{\nu} = \boldsymbol{A}^{\mu'}$, $\mu' \sqsubseteq \mu$, $X(\boldsymbol{A}) \subseteq X(\varrho) = \mathit{dom}(\mu)$. Thus $\boldsymbol{A}^{\mu}$ is a literal, and there is an injective $\nu'\colon X \rightharpoonup X$ with $\nu \sqsubseteq \nu'$, $\mathit{dom}(\nu') = \mathit{dom}(\nu) \cup X(\boldsymbol{C})$, and $\boldsymbol{C}^{\nu'} = \boldsymbol{A}^{\mu}$. As a consequence, there is also a sequence

$$\varepsilon\Diamond[q_0]^{\iota} \mathrel{\vdash\!\!\!\sim}^{*} \alpha\Diamond[\boldsymbol{B} \to \gamma\centerdot\boldsymbol{C}\delta]^{\nu} \mathrel{\vdash\!\!\!\sim}_{\mathsf{go}} \alpha\boldsymbol{A}^{\mu}\Diamond[\boldsymbol{B} \to \gamma\boldsymbol{C}\centerdot\delta]^{\nu'}$$

showing that $\alpha\boldsymbol{A}^{\mu}$ is approved by the nCFA and consequently, using Lemma 7.11, by the dCFA. Moreover, $X(\alpha) \cap X(\varrho^{\mu}) \subseteq X(\varrho^{\mu'}) = X(\boldsymbol{A}^{\mu'}) \subseteq X(\boldsymbol{A}^{\mu})$ using Lemma 6.8.

For the *if* direction, let $\boldsymbol{A} \to \varrho$ be a rule with $\boldsymbol{A} \neq \boldsymbol{\mathit{Start}}$, and consider a sequence $\varepsilon\blacklozenge Q_0^{\iota} \mathrel{\vDash\!\!\!\approx}^{*} \alpha\varrho^{\mu}\blacklozenge Q^{\tau}$ with an injective $\mu\colon X \rightharpoonup X$, and $\varepsilon\blacklozenge Q_0^{\iota} \mathrel{\vDash\!\!\!\approx}^{*} \alpha\boldsymbol{A}^{\mu}\blacklozenge\widehat{Q}^{\xi}$ for some dCFA state $\widehat{Q}$ and injective $\xi\colon X \rightharpoonup X$. By Lemma 7.12, there is a sequence

$$\varepsilon\Diamond[q_0]^{\iota} \mathrel{\vdash\!\!\!\sim}^{*} \alpha\Diamond[\boldsymbol{B} \to \gamma\centerdot\boldsymbol{C}\delta]^{\nu'} \mathrel{\vdash\!\!\!\sim}_{\mathsf{go}} \alpha\boldsymbol{A}^{\mu}\Diamond[\boldsymbol{B} \to \gamma\boldsymbol{C}\centerdot\delta]^{\nu}$$

with $\boldsymbol{A}^{\mu} = \boldsymbol{C}^{\nu}$, $\nu' \sqsubseteq \nu$, and $\mathit{dom}(\nu) = \mathit{dom}(\nu') \cup X(\boldsymbol{C})$. Therefore, there is also a sequence

$$\varepsilon\Diamond[q_0]^{\iota} \mathrel{\vdash\!\!\!\sim}^{*} \alpha\Diamond[\boldsymbol{B} \to \gamma\centerdot\boldsymbol{C}\delta]^{\nu'} \mathrel{\vdash\!\!\!\sim}_{\mathsf{cl}} \alpha\Diamond[\boldsymbol{A} \to {\centerdot}\varrho]^{\mu'} \mathrel{\vdash\!\!\!\sim}_{\mathsf{go}}^{*} \alpha\varrho^{\mu}\Diamond[\boldsymbol{A} \to \varrho\centerdot]^{\mu}$$

with $\boldsymbol{C}^{\nu'} = \boldsymbol{A}^{\mu'}$ and, by Lemma 7.11, a sequence $\varepsilon\blacklozenge Q_0^{\iota} \mathrel{\vDash\!\!\!\approx}^{*} \alpha\varrho^{\mu}\blacklozenge Q'^{\tau'}$ with a dCFA state Q', injective $\tau'\colon X \rightharpoonup X$, and $\langle \boldsymbol{A} \to \varrho\centerdot, \mu\rangle \in Q'^{\tau'}$. In fact,

$Q = Q'$ and $\tau = \tau'$ since the dCFA is deterministic. Hence, Q contains an item $\langle \boldsymbol{A} \to \varrho \centerdot, \sigma \rangle$ with $\mu = \tau \circ \sigma$. ☐

We are now ready to prove that the ASR parser is in fact an improved version of the naïve shift-reduce parser (Def. 4.4) that always makes sure that its stack is a viable prefix:

Lemma 8.5. *For every ASR parser configuration $\mathcal{S} \centerdot g$ with $\mathit{graph}(\mathcal{S}) \neq$ **Start** and every $n \in \mathbb{N}$, $Q_0^\iota \centerdot \varepsilon \models^n \mathcal{S} \centerdot g$ if and only if $\varepsilon \blacklozenge Q_0^\iota \mathrel{|\!\!\approx}^* \varphi \blacklozenge R$ and $\varepsilon \centerdot \varepsilon \vdash^n \varphi \centerdot g$ where $R = \mathit{top}(\mathcal{S})$ and $\varphi = \mathit{graph}(\mathcal{S})$.*

Proof. We prove the proposition by induction on n. For $n = 0$, it immediately follows from the fact that $\mathcal{S} = \mathit{top}(\mathcal{S}) = Q_0^\iota$ and $\mathit{graph}(\mathcal{S}) = g = \varepsilon$.

For the inductive step, let $n \geq 0$. We have to show that the proposition holds for $n + 1$ under the assumption that it holds for all shorter configuration sequences of length up to n. We show the *only-if* and the *if* direction separately.

(1) To show the *only-if* direction, we assume any sequence

$$Q_0^\iota \centerdot \varepsilon \models^n \mathcal{S}' \centerdot g' \models \mathcal{S} \centerdot g.$$

Let $R = \mathit{top}(\mathcal{S}')$ and $\varphi = \mathit{graph}(\mathcal{S}')$. The last move is either a shift move or a reduce move.

(1a) If it is a shift move, there exist a literal $\boldsymbol{e} \in \mathit{Lit}_{\mathcal{T}}$ and a concrete state $T \in \mathcal{Q}_{\mathrm{M}}$ with

$$\varphi \blacklozenge R \mathrel{|\!\!\approx} \varphi \boldsymbol{e} \blacklozenge T \tag{25}$$

$$X(\boldsymbol{e}) \cap X(g) \subseteq X(\varphi) \tag{26}$$

$$\mathcal{S} = \mathcal{S}' \boldsymbol{e} T \tag{27}$$

$$g = g' \boldsymbol{e}. \tag{28}$$

Now, $\varepsilon \blacklozenge Q_0^\iota \mathrel{|\!\!\approx}^* \varphi \blacklozenge R \mathrel{|\!\!\approx} \varphi \boldsymbol{e} \blacklozenge T$ follows from (25) and the induction hypothesis, and $\varepsilon \centerdot \varepsilon \vdash^n \varphi \centerdot g' \vdash_{\mathsf{sh}} \varphi \boldsymbol{e} \centerdot g$ from the induction hypothesis, (26), (28), and Def. 4.4. This proves the proposition because $\mathit{top}(\mathcal{S}') = T$ and $\mathit{graph}(\mathcal{S}') = \varphi \boldsymbol{e}$.

(1b) If the last move is a reduce move, there is a rule $\boldsymbol{A} \to \varrho$, and one can obtain a parse stack $\mathcal{S}''$ by removing $2 \cdot |\varrho|$ elements from the end of $\mathcal{S}'$. Let $\psi = \mathit{graph}(\mathcal{S}'')$ and $Q = \mathit{top}(\mathcal{S}'')$. By Def. 8.1, there is a dCFA state $Q_i \in \mathcal{Q}$ such that Q_i contains a p-item $\langle \boldsymbol{A} \to \varrho \centerdot, \sigma \rangle$ and a concrete state $T \in \mathcal{Q}_{\mathrm{M}}$ with

$$\psi \blacklozenge Q \mathrel{|\!\!\approx} \psi \boldsymbol{A}^{\tau \circ \sigma} \blacklozenge T \tag{29}$$

$$\mathcal{S} = \mathcal{S}'' \boldsymbol{A}^{\tau \circ \sigma} T \tag{30}$$

$$g = g'. \tag{31}$$

Moreover, by Lemma 8.4, there is a graph $\alpha \in \mathcal{G}_\Sigma$ and a concrete state $T' \in \mathcal{Q}_\mathrm{M}$ such that

$$\varphi = \alpha \varrho^{\tau\circ\sigma} \tag{32}$$

$$\varepsilon \blacklozenge Q_0^\iota \mathrel{|\!\!\approx}^* \alpha \boldsymbol{A}^{\tau\circ\sigma} \blacklozenge T' \tag{33}$$

$$X(\alpha) \cap X(\varrho^{\tau\circ\sigma}) \subseteq X(\boldsymbol{A}^{\tau\circ\sigma}). \tag{34}$$

Now, $\alpha = \psi$ follows from the construction of $\mathcal{S}''$ and

$$\varepsilon \blacklozenge Q_0^\iota \mathrel{|\!\!\approx}^* \alpha \boldsymbol{A}^{\tau\circ\sigma} \blacklozenge T' = \psi \boldsymbol{A}^{\tau\circ\sigma} \blacklozenge T$$

from (29), (33), and the fact that the dCFA is deterministic. Finally,

$$\varepsilon \mathbin{\scriptstyle\blacksquare} \varepsilon \mathrel{\vdash\!\!\!-}^n \varphi \mathbin{\scriptstyle\blacksquare} g' = \psi \varrho^{\tau\circ\sigma} \mathbin{\scriptstyle\blacksquare} g \mathrel{\underset{\boldsymbol{A}^{\tau\circ\sigma} \Rightarrow \varrho^{\tau\circ\sigma}}{\vdash\!\!\relbar\!\!\relbar\!\!\relbar\!\!\relbar}} \psi A^{\tau\circ\sigma} \mathbin{\scriptstyle\blacksquare} g$$

using the induction hypothesis, (31), (32), (34) and Def. 4.4. This proves the proposition because *top*$(\mathcal{S}) = T$ and *graph*$(\mathcal{S}) = \psi A^{\tau\circ\sigma}$.

(2) To show the *if* direction, we now assume any sequence

$$\varepsilon \mathbin{\scriptstyle\blacksquare} \varepsilon \mathrel{\vdash\!\!\!-}^n \varphi' \mathbin{\scriptstyle\blacksquare} g' \vdash \varphi \mathbin{\scriptstyle\blacksquare} g. \tag{35}$$

of moves and

$$\varepsilon \blacklozenge Q_0^\iota \mathrel{|\!\!\approx}^* \varphi \blacklozenge R \tag{36}$$

for some concrete state $R \in \mathcal{Q}_\mathrm{M}$. The last move in (35) is either a shift move or a reduce move.

(2a) If it is a shift move, there exists a literal $\boldsymbol{e} \in Lit_\mathcal{T}$ such that

$$\varphi = \varphi' \boldsymbol{e} \tag{37}$$

$$g = g' \boldsymbol{e} \tag{38}$$

$$X(\boldsymbol{e}) \cap X(g') \subseteq X(\varphi') \tag{39}$$

Because of (37), we can write (36) as

$$\varepsilon \blacklozenge Q_0^\iota \mathrel{|\!\!\approx}^* \varphi' \blacklozenge Q \mathrel{|\!\!\approx} \varphi' \boldsymbol{e} \blacklozenge R \tag{40}$$

for some concrete state $Q \in \mathcal{Q}_\mathrm{M}$. Therefore, the induction hypothesis applies and yields $Q_0^\iota \mathbin{\scriptstyle\blacksquare} \varepsilon \models^n \mathcal{S}' \mathbin{\scriptstyle\blacksquare} g'$ with *top*$(\mathcal{S}') = Q$ and *graph*$(\mathcal{S}') = \varphi'$. Finally, because of (38), (39) and (40), there is a shift move

$$\mathcal{S}' \mathbin{\scriptstyle\blacksquare} g' \models \mathcal{S}' \boldsymbol{e} R \mathbin{\scriptstyle\blacksquare} g' \boldsymbol{e} = \mathcal{S} \mathbin{\scriptstyle\blacksquare} g$$

with $\mathcal{S} = \mathcal{S}' \boldsymbol{e} R$ and, therefore, *top*$(\mathcal{S}) = R$ and *graph*$(\mathcal{S}) = \varphi' \boldsymbol{e} = \varphi$ because of (37), which proves the proposition.

(2b) If the last move is a reduce move, there is a rule $\boldsymbol{A} \to \varrho$, a match $\mu \colon X \to X$, and a graph $\alpha \in \mathcal{G}_\Sigma$ such that

$$\varphi' = \alpha \varrho^\mu \tag{41}$$

$$\varphi = \alpha \boldsymbol{A}^\mu \tag{42}$$

$$g = g' \tag{43}$$

$$X(\alpha) \cap X(\varrho^\mu) \subseteq X(\boldsymbol{A}^\mu) \tag{44}$$

and (36) can be written as

$$\varepsilon\blacklozenge Q_0^\iota \mathrel{|\!\approx}^* \alpha\boldsymbol{A}^\mu\blacklozenge R. \tag{45}$$

The graph $\varphi = \alpha\boldsymbol{A}^\mu$ is a viable prefix because of (36), Thm. 6.12, and Thm. 7.13. Therefore, $\varphi' = \alpha\varrho^\mu$ is also a viable prefix because of $\varphi = \alpha\boldsymbol{A}^\mu \underset{\mathrm{rm}}{\Longrightarrow} \alpha\varrho^\mu = \varphi'$. Since the grammar is reduced, there must be concrete states $Q, Q' \in \mathcal{Q}_{\mathrm{M}}$ such that

$$\varepsilon\blacklozenge Q_0^\iota \mathrel{|\!\approx}^* \alpha\blacklozenge Q' \mathrel{|\!\approx}^* \alpha\varrho^\mu\blacklozenge Q = \varphi'\blacklozenge Q. \tag{46}$$

Because of (35), there is also a sequence $\varepsilon \centerdot \varepsilon \vdash^k \alpha \centerdot g''$ for some prefix g'' of $g = g'$ and $k \leq n$. Therefore, the induction hypothesis applies, and we can conclude

$$Q_0^\iota \centerdot \varepsilon \models^k \mathcal{S}'' \centerdot g''$$

for a parse stack $\mathcal{S}''$ with $\mathit{top}(\mathcal{S}'') = Q'$ and $\mathit{graph}(\mathcal{S}'') = \alpha$. Using the same argument, we can also conclude

$$Q_0^\iota \centerdot \varepsilon \models^n \mathcal{S}' \centerdot g'$$

for a parse stack $\mathcal{S}'$ with $\mathit{top}(\mathcal{S}') = Q$ and $\mathit{graph}(\mathcal{S}') = \alpha\varrho^\mu = \varphi'$. Let us assume that $\mathcal{S}''$ is not a prefix of $\mathcal{S}'$. There must be a parse stack $\hat{\mathcal{S}}$, literal $\boldsymbol{l}$ and concrete states $P', P'' \in \mathcal{Q}_{\mathrm{M}}$, $P' \neq P''$, such that $\hat{\mathcal{S}}\boldsymbol{l}P'$ is a prefix of $\mathcal{S}'$ and $\hat{\mathcal{S}}\boldsymbol{l}P''$ a prefix of $\mathcal{S}''$. Let $\psi = \mathit{graph}(\hat{\mathcal{S}}\boldsymbol{l}P') = \mathit{graph}(\hat{\mathcal{S}}\boldsymbol{l}P'')$. We can conclude $\varepsilon\blacklozenge Q_0^\iota \mathrel{|\!\approx}^* \psi\blacklozenge P'$ and $\varepsilon\blacklozenge Q_0^\iota \mathrel{|\!\approx}^* \psi\blacklozenge P''$ using Fact 8.2, and $P' = P''$ using the fact that the dCFA is deterministic, contradicting our assumption. $\mathcal{S}''$ is thus a prefix of $\mathcal{S}'$, and $\mathcal{S}''$ can be obtained from $\mathcal{S}'$ by removing $2 \cdot |\varrho|$ elements from its end. Because of (44), (45), (46), and Lemma 8.4, there is a state $Q_i \in \mathcal{Q}$, a node mapping $\tau\colon \mathit{params}(Q_i) \to X$ and an item $\langle \boldsymbol{A} \to \varrho \centerdot, \sigma\rangle \in Q_i$ such that

$$Q = Q_i^\tau \tag{47}$$

$$\mu = \tau \circ \sigma. \tag{48}$$

Moreover, we know that

$$\alpha\blacklozenge Q' \mathrel{|\!\approx} \alpha\boldsymbol{A}^\mu\blacklozenge R$$

by (45) and (46), using the fact that the dCFA is deterministic. Therefore, using Def. 8.1,

$$Q_0^\iota \centerdot \varepsilon \models^n \mathcal{S}' \centerdot g' \models \mathcal{S}''\boldsymbol{A}^\mu R \centerdot g'.$$

This proves the proposition because of (43) and (48), choosing $\mathcal{S} = \mathcal{S}''\boldsymbol{A}^{\tau\circ\sigma}R$. ☐

We are now ready to prove the correctness of the ASR parser.

Theorem 8.6. *Let* $g \in \mathcal{G}_\mathcal{T}$*. The ASR parser accepts* g *if and only if* $\boldsymbol{Z} \underset{\mathrm{rm}}{\Longrightarrow}^* g$*. Moreover, for every reachable configuration* $\mathcal{S} \mathbin{\blacksquare} g$*, there is a graph* $g' \in \mathcal{G}_\mathcal{T}$ *and an accepting configuration* $\mathcal{S}' \mathbin{\blacksquare} gg'$ *such that* $\mathcal{S} \mathbin{\blacksquare} g \models^* \mathcal{S}' \mathbin{\blacksquare} gg'$*.*

Proof. Consider any graph $g \in \mathcal{G}_\mathcal{T}$.

For the first part of the theorem, by Thm. 4.9 it holds that $\boldsymbol{Z} \underset{\mathrm{rm}}{\Longrightarrow}^* g$ if and only if $\varepsilon \mathbin{\blacksquare} \varepsilon \vdash^* \boldsymbol{Z} \mathbin{\blacksquare} g$. By Lemma 8.5, the latter is the case if an only if $Q_0^\iota \mathbin{\blacksquare} \varepsilon \models^* Q_0^\iota \boldsymbol{Z} Q_\mathsf{A} \mathbin{\blacksquare} g$, because the dCFA approves the viable prefix $\boldsymbol{Z}$ via $\varepsilon \blacklozenge Q_0 \mathrel{|\!\approx}^* \boldsymbol{Z} \blacklozenge Q_\mathsf{A}$.

To prove the second part of the theorem, consider any configuration $\mathcal{S} \mathbin{\blacksquare} g$ with $Q_0^\iota \mathbin{\blacksquare} \varepsilon \models^* \mathcal{S} \mathbin{\blacksquare} g$. By Lemma 8.5, Thm. 6.12, and Thm. 7.13, $\mathit{graph}(\mathcal{S})$ is a viable prefix. Moreover, $\varepsilon \mathbin{\blacksquare} \varepsilon \vdash^* \mathit{graph}(\mathcal{S}) \mathbin{\blacksquare} g$. By Lemma 5.5, there is a graph $g' \in \mathcal{G}_\mathcal{T}$ such that $\mathit{graph}(\mathcal{S}) \mathbin{\blacksquare} g \vdash^* \boldsymbol{Z} \mathbin{\blacksquare} gg'$. Thus, the same argument as above yields $\mathcal{S} \mathbin{\blacksquare} g \models^* Q_0^\iota \boldsymbol{Z} Q_\mathsf{A} \mathbin{\blacksquare} gg'$, i.e., the ASR parser accepts gg'. ☐

It is worthwhile pointing out that the ASR parser is still nondeterministic, despite the fact that it is "assisted" by a dCFA. In fact, there are two sources of nondeterminism. First, the state on top of the stack may contain several items that fulfill the conditions of shift or reduce moves and thus enable several possible moves. There may be items leading to shifts of different literals, items that result in reductions according to different rules, and items of which one triggers a shift move whereas the other triggers a reduce move. For example, in state Q_1 of the dCFA in Fig. 14, the parser may choose among three shift moves.

The second source of nondeterminism lies in the choice of the edge to be consumed by a shift move, as there may be several literals $\boldsymbol{e}$ in the input graph that fulfill the conditions.

Naturally, the "right" choice must be made in order to ensure that the parser accepts a given input graph. Note that this does not contradict Thm. 8.6 which states that, regardless of the choice made, there exists a possible rest graph with which the parser can reach an accepting configuration. Clearly, that rest graph can differ from the actual rest graph in the input. Looking at the ASR parser, this observation should not come as a surprise, because the parser does not inspect the rest graph in any way (except for selecting a literal to be shifted whenever a shift move is made). The extension of the ASR parser by an appropriate inspection of the rest graph to *predict* the necessary move will be discussed next. It leads to the main notion proposed in this paper, the predictive shift-reduce parser.

9. Predictive Shift-Reduce Parsing

Intuitively, an appropriate move of the parser is a move that keeps it on its way towards accepting the input graph g, provided that g is valid. (Naturally, if g is not valid, every possible move is appropriate as g will eventually be rejected anyway.) To identify such a move, the parser needs criteria that it can check by inspecting the rest graph. These criteria should preferably only require a fixed number of patterns to be checked, in order to ensure that an appropriate move can be selected in constant or nearly constant time. While the desired patterns will obviously have to depend on Γ, they should be computable as static information by the parser generator. Similarly to the string case, this is only possible if Γ is conflict-free in a sense to be made precise in this section. Thus, in contrast to the pure ASR parser, which works for every HR grammar, the resulting *predictive shift-reduce parser* exists only for a subset of all HR grammars, i.e., the parser generator may fail to construct a parser, reporting the existence of a conflict instead.

For the following considerations, suppose that the ASR parser is in the process of parsing a valid input graph g and has reached a configuration $\mathcal{S}\,\boldsymbol{.}\,g'$, but has not yet processed the rest graph r of g where $g \bowtie g'r$.[7] The top of $\mathcal{S}$ is $top(\mathcal{S}) = Q^{\tau}$ with a CFA state $Q \in \mathcal{Q}$ and a node mapping τ.

The parser must now choose between shift and reduce moves until the input graph has been accepted or no further move is possible. Shift moves are caused by transitions leaving Q, and reduce moves by items within Q with a dot at the end of their right-hand side. Let us call such an item a *reduce item.* Each transition and each reduce item is called a *trigger* that causes the corresponding move. Note that acceptance is also caused by a reduce item, which is the only item in the accepting state Q_{A}.

We now describe an effective decision procedure, which inspects the rest graph r to select the trigger that causes the "right" move, i.e., a move which turns the parser into a new configuration from which it can still reach an accepting configuration by consuming the remaining rest graph. Let us call a sequence of moves that ends in an accepting configuration a *successful sequence*, even if it does not process the entire rest graph. Thm. 8.6 states that such a sequence always exists when the parser has reached $\mathcal{S}\,\boldsymbol{.}\,g'$. The decision procedure must thus select a trigger that causes a successful sequence (by causing the first move of this sequence) that processes the entire rest graph.

The idea for selecting the right trigger is as follows:

Suppose that the rest graph r is not yet empty. The procedure now checks for each trigger whether r contains any literal that will be processed next by any successful sequence caused by this trigger. There must be a trigger with this property because g is valid. If this trigger is the only possible one, this trigger must be the one causing the right move; the parser thus selects this trigger. If, however, two or more triggers have this property, our procedure fails; it cannot *predict* the right move.

Let us consider more closely when a literal is processed next by a successful sequence caused by a trigger. If the trigger is a transition, this literal is just the one that is processed by the corresponding shift move. If the trigger, however, is a reduce item, it must be the one processed by the first shift move in the move

---

[7] Note that we can represent the rest graph by any permutation of r because none of its literals have been processed by the parser yet.

sequence following the reduce move. This shift move may of course not be the first move of the sequence, as it can be preceded by further reduce moves.

Suppose now that the parser has processed the input graph entirely, i.e. the rest graph r is empty. The procedure then checks for each reduce item whether there is a successful sequence that consists of reduce moves only. The parser then selects any reduce item that causes such a successful sequence.

We will now discuss the decision procedure more precisely. To this end, we consider all successful sequences caused by a trigger. Recall that we suppose that the parser has reached configuration $\mathcal{S} \bullet g'$ with $\mathit{top}(\mathcal{S}) = Q^\tau$.

Suppose the trigger is a transition $\mathit{tr} = (Q \xrightarrow{(\boldsymbol{e},\mu)} Q')$ of the dCFA. Def. 8.1 implies that the shift move induced by tr is $\mathcal{S} \bullet g' \underset{\mathit{tr}}{\models} \mathcal{S}\boldsymbol{f}Q'' \bullet g'\boldsymbol{f}$ for an appropriate literal $\boldsymbol{f} \in \mathit{Lit}_{\mathcal{T}}$ and concrete state $Q'' \in \mathcal{Q}_{\mathrm{M}}$. And by Thm. 8.6, there is a graph $v \in \mathcal{G}_{\mathcal{T}}$ such that $\mathcal{S}\boldsymbol{f}Q'' \bullet g'\boldsymbol{f} \models \mathcal{S}_{\mathrm{A}} \bullet g'\boldsymbol{f}v$ with $\mathit{top}(\mathcal{S}_{\mathrm{A}}) = Q_{\mathsf{A}}$. This means that the parser accepts $g'\boldsymbol{f}v$ or, in other words, $\boldsymbol{f}v$ is the graph processed by this successful sequence. Let us denote the set of all graphs processed by any successful sequence caused by tr as $\mathit{Success}(Q^\tau, g', \mathit{tr})$.

Suppose now that the trigger is a reduce item $\mathit{it} = \langle \boldsymbol{A} \to \varrho \bullet, \sigma \rangle \in Q$. Def. 8.1 implies that the reduced move induced by it is $\mathcal{S} \bullet g' \underset{\mathit{it}}{\models} \mathcal{S}'\boldsymbol{A}^{\tau\circ\sigma}Q' \bullet g'$ with an appropriate parse stack $\mathcal{S}'$ and concrete state $Q' \in \mathcal{Q}_{\mathrm{M}}$. And by Thm. 8.6, there is a graph $g'' \in \mathcal{G}_{\mathcal{T}}$ such that $\mathcal{S}'\boldsymbol{A}^{\tau\circ\sigma}Q' \bullet g' \models \mathcal{S}_{\mathrm{A}} \bullet g'g''$ with $\mathit{top}(\mathcal{S}_{\mathrm{A}}) = Q_{\mathsf{A}}$. This means that g'' is the graph processed by this successful sequence. Let us denote the set of all graphs processed by any successful sequence caused by it as $\mathit{Success}(Q^\tau, g', \mathit{it})$.

Before utilizing the sets $\mathit{Success}(Q^\tau, g', t)$ for a trigger t, let us introduce some terminology first. For a graph $h = \boldsymbol{e}_1 \cdots \boldsymbol{e}_n$ with $n > 0$ literals, let $\mathit{First}(h) = \boldsymbol{e}_1$ be the first literal of h. In the special case $n = 0$, we let $\mathit{First}(\varepsilon) = \$$ where the special symbol \$ indicates that there are no literals at all. For a set $S \subseteq \mathcal{G}_\Sigma$ of graphs, let $\mathit{First}(S) = \{\mathit{First}(h) \mid h \in S\}$.

For a trigger t, now consider the set

$$\mathit{First}(\mathit{Success}(Q^\tau, g', t)).$$

This set contains all literals that can be processed next by any successful sequence caused by t, and it contains \$ if there is a successful sequence caused by t without any shift move. The decision procedure, whose idea has been outlined above, thus has to select the trigger t such that $\mathit{First}(\mathit{Success}(Q^\tau, g', t))$ contains any literal of the rest graph, or \$ if $r = \varepsilon$. However, this does not make a practical decision procedure because these sets are in general infinite. We turn these sets into finite sets by mapping their members to *pseudo-literals* as described next.

Note first that every node visited by any literal in any of these sets falls into one of three categories: It is either (1) a node assigned by τ to a parameter of Q, (2) a node not occurring in $X(g')$, or (3) any node in $X(g')$ not assigned to a parameter by τ. We now define the following function that maps nodes of category (1) to their corresponding parameter, nodes of category (2) to '–', and all others to ‘$\bullet$’.

$$f_Q^{\tau,g'}(x) = \begin{cases} y & \text{if there exists } y \in \mathit{params}(Q) \text{ such that } \tau(y) = x \\ - & \text{if } x \notin X(g') \\ \bullet & \text{otherwise} \end{cases}$$

We extend function $f_Q^{\tau,g'}$ to literals and sets of literals in the obvious way. Literals are thus turned into *pseudo-literals*, which are similar to literals, but may be attached to ‘–’ and ‘$\bullet$’ instead of nodes.[8]

Function $f_Q^{\tau,g'}$ applied to $\mathit{First}(\mathit{Success}(Q^\tau, g', t))$ turns this set into a finite set. This is so because the number of terminal labels and the number of parameters in Q are finite. But one cannot compute this set statically because it depends on g'. Recall that the node mapping τ is uniquely determined by g' because the dCFA approves g' by $\varepsilon \blacklozenge Q_0^\iota \mathrel{|\!\!\approx}^* g' \blacklozenge Q^\tau$ and the dCFA is deterministic. To simplify things, let us define the finite set

$$\mathit{Follow}(Q, t) := \bigcup_{g' \in \mathcal{G}_\Sigma} f_Q^{\tau,g'}(\mathit{First}(\mathit{Success}(Q^\tau, g', t))) \tag{49}$$

by building the union over all terminal graphs g'. Clearly, only the graphs g' approved by the dCFA as mentioned above contribute to this set. This set just depends on the state Q and one of its triggers t, and is thus static information independent of the input graph. While $\mathit{Follow}(Q, t)$ cannot directly be computed using (49), one can compute it by analyzing the dCFA in a very similar way as one can compute the *follower* symbols for string grammars (Sect. 3).

Example 9.1. Consider the dCFA for the tree-generating grammar shown in Fig. 10 and in particular state Q_4, which has two triggers: Trigger tr is the transition from Q_4 to Q_3, and trigger it is the reduce item $\langle T(y) \to T(y)\, e(y,z)\, T(z) \bullet, [y/x, z/a]\rangle$. Q_4 has the parameters x and a. Function $f_{Q_4}^{\tau,g'}$, when applied to nodes, thus maps into the set $\{-, \bullet, x, a\}$. In fact

$$\begin{aligned} \mathit{Follow}(Q_4, \mathit{tr}) &= \{e(a, -)\} \\ \mathit{Follow}(Q_4, \mathit{it}) &= \{e(x, -), e(\bullet, -), \$\} \end{aligned}$$

It is clear that any successful sequence caused by transition tr must begin with a shift move and that the consumed literal must match edge $e(a, b)$, which is ascribed to the transition. However, the “new” parameter b is mapped to $-$.

Reduce item it can cause a successful sequence without any shift move, indicated by \$. To see this, consider, e.g., a parse stack $\mathcal{S}$ with $\mathit{top}(\mathcal{S}) = Q_4^\tau$ and $\mathit{graph}(\mathcal{S}) = \mathit{root}(1)\, T(1)\, e(1,2)\, T(2)$. The reduce move will yield a stack $\mathit{root}(1)\, T(1)$, which can be further reduced to $\mathit{Start}()$.

Moreover, $e(x, -)$ and $e(\bullet, -)$ indicate that the literal consumed next must be an e-literal attached to node $\tau(x)$ or any node that has been processed already, but that is not kept track of by a parameter in Q_4, indicated by $\bullet$, and a node that has not yet been processed, indicated by $-$. ☐

---

[8] Note that these pseudo-literals are a generalized version of those introduced in Sect. 6.

Now let $\boldsymbol{e}$ be a literal of the rest graph r. The definition of $\mathit{Follow}(Q,t)$ implies that $f_Q^{\tau,g'}(\boldsymbol{e}) \in \mathit{Follow}(Q,t)$ is a necessary (but not always sufficient) and easily verifiable condition for $\boldsymbol{e}$ to be a literal that can be processed next by a successful sequence caused by trigger t. Similarly, $\$ \in \mathit{Follow}(Q,t)$ can be used to check whether t can cause a successful sequence without any shift move. A naïve procedure may thus try to identify the "right" trigger in the following way: If $r \neq \varepsilon$, it looks for any trigger t of Q such that r contains a literal $\boldsymbol{e}$ with $f_Q^{\tau,g'}(\boldsymbol{e}) \in \mathit{Follow}(Q,t)$. If it can identify a unique trigger with this property, this trigger is selected. However, this procedure fails if it cannot determine a unique trigger that way.

Let us now examine a way in which such a procedure can uniquely select the trigger causing the right move, even if the naïve procedure would identify two or more candidates for the "right" trigger. For that, assume that there are two different triggers t, t' such that r contains literals $\boldsymbol{e}, \boldsymbol{e}'$ satisfying

$$f_Q^{\tau,g'}(\boldsymbol{e}) \in \mathit{Follow}(Q,t) \tag{50}$$

$$f_Q^{\tau,g'}(\boldsymbol{e}') \in \mathit{Follow}(Q,t'). \tag{51}$$

Recall that function $f_Q^{\tau,g'}$ may map many literals to the same pseudo-literal. Moreover, $\mathit{Follow}(Q,t)$ contains the pseudo-literals of any literal that may be consumed next in some successful sequence, not necessarily only those that process the rest graph r entirely. As a consequence, (50) and (51) do in fact not imply that $\boldsymbol{e}$ or $\boldsymbol{e}'$ will be processed next when t or t', respectively, is selected. However, if we notice somehow—and additionally to (50)—that $\boldsymbol{e}$ can never be processed by any successful sequence caused by t', we can eliminate t' from the candidates for the right trigger, even if (51) is satisfied. This observation leads the way to an effective procedure for selecting the right trigger.

Let us determine which literals can be processed by a successful sequence caused by a trigger t. We are not only interested in the literals that are processed first, but also in those literals that are processed *eventually.* Instead of a function First, we will use a function Any which is defined as follows: For a graph $h = \boldsymbol{e}_1 \cdots \boldsymbol{e}_n$ with $n > 0$ literals, let $\mathit{Any}(h) = \{\boldsymbol{e}_1, \ldots, \boldsymbol{e}_n\}$ the set of all of its literals. For the empty graph, let $\mathit{Any}(\varepsilon) = \{\$\}$. For a set $S \subseteq \mathcal{G}_\Sigma$ of graphs, let $\mathit{Any}(S) = \bigcup_{h \in S} \mathit{Any}(h)$. We then define the finite set

$$\mathit{Follow}^*(Q,t) := \bigcup_{g' \in \mathcal{G}_\Sigma} f_Q^{\tau,g'}(\mathit{Any}(\mathit{Success}(Q^\tau, g', t))). \tag{52}$$

Note the close resemblance to (49); the only difference is the use of Any instead of First, i.e., $\mathit{Follow}^*(Q,t)$ contains the $f_Q^{\tau,g'}$-images of all literals that occur eventually in some graph processed by a successful sequence caused by t, and it contains $\$$ if there is a successful sequence caused by t that does not contain any shift move.

Again, this definition cannot be used for computing $\mathit{Follow}^*(Q,t)$ directly, but one can compute it by analyzing the dCFA in a similar way as for $\mathit{Follow}(Q,t)$.

Example 9.2. We continue Example 9.1 and consider again the dCFA for the tree-generating grammar shown in Fig. 10 and in particular state Q_4 with its two triggers tr and it. In addition to

$$\begin{aligned} \mathit{Follow}(Q_4, tr) &= \{e(a,–)\} \\ \mathit{Follow}(Q_4, it) &= \{e(x,–), e(\bullet,–), \$\} \end{aligned}$$

we have

$$\begin{aligned} \mathit{Follow}^*(Q_4, tr) &= \{e(a,–), e(\bullet,–), e(–,–)\} \\ \mathit{Follow}^*(Q_4, it) &= \{e(x,–), e(\bullet,–), e(–,–), \$\} \end{aligned}$$

We can see that any literal that matches the only pseudo-literal $e(a,–)$ in $\mathit{Follow}(Q_4, tr)$ can never be processed in any successful sequence caused by it, even if the rest graph contains literals matching the pseudo-literals $e(x,–)$ or $e(\bullet,–)$, which are members of $\mathit{Follow}(Q_4, it)$. This can be concluded from the fact that $e(a,–)$ does not occur in $\mathit{Follow}^*(Q_4, it)$. As a consequence, it cannot be the right trigger if we find a literal that matches $e(a,–)$.

However, we can see that a literal that matches $e(\bullet,–) \in \mathit{Follow}(Q_4, it)$ may indeed be processed later when transition tr is chosen. The existence of a literal matching any pseudo-literal in $\mathit{Follow}(Q_4, it)$ does thus not help to eliminate tr from the candidates of right triggers.

As a consequence, a procedure can reliably predict the next move in state Q_4 by first checking whether there is a rest graph literal $\boldsymbol{e}$ with $f^{\tau,g'}_{Q_4}(\boldsymbol{e}) = e(a,–)$. If there is such a literal, tr is guaranteed to be the right trigger because $e(a,–) \notin \mathit{Follow}^*(Q_4, it)$. If such a literal, however, does not exist, one can check whether the rest graph contains any literal $\boldsymbol{e}'$ that matches a pseudo-literal of $\mathit{Follow}(Q_4, it)$, i.e., with $f^{\tau,g'}_{Q_4}(\boldsymbol{e}') \in \mathit{Follow}(Q_4, it)$. If there is is such a literal, one chooses the reduce move caused by it. If there is no such $\boldsymbol{e}'$, it is guaranteed that there is no successful sequence caused by it that processes the rest graph entirely, and the parser can terminate with a failure. ☐

This example motivates that one must compare the *Follow* and *Follow** sets of the different triggers and that one must determine which trigger should be considered first when looking for rest graph literals that match any pseudo-literals in the *Follow* set of this trigger:

Definition 9.3. A trigger t *precedes* a trigger t', written $t \prec t'$, if t and t' are triggers of the same state $Q \in \mathcal{Q}$, $t \neq t'$, and $\mathit{Follow}^*(Q, t) \cap \mathit{Follow}(Q, t') \neq \varnothing$.

Note that $\prec$ is not an ordering because it is in general not transitive. But $t \prec t'$ indicates that one must check t prior to t'. However, $t \prec t'$ does not help to find an order if there is a $\prec$-chain $t \prec t' \prec \cdots \prec t$. This motivates the definition of *conflicting* triggers. We will see in the following that an effective decision procedure for identifying the right trigger requires conflict-freeness:

Definition 9.4. Let $Q \in \mathcal{Q}$ be a state and T_Q the set of its triggers. A subset $T \subseteq T_Q$ is *in conflict* if there is a sequence $t_1 \prec t_2 \prec \cdots \prec t_k \prec t_1$ with $T = \{t_1, t_2, \ldots, t_k\}$. Q is *conflict-free* if no subset of its triggers is in conflict.

If a state is conflict-free, one can sort the triggers such their order reflects $\prec$, which will be necessary for the effective decision procedure:

Lemma 9.5. *For every conflict-free state $Q \in \mathcal{Q}$, there is an ordered sequence $t_1, \ldots, t_n$ of its triggers such that $\mathit{Follow}(Q, t_i) \cap \mathit{Follow}^*(Q, t_j) = \varnothing$ for every pair of indices i, j with $i < j$.*

Proof. Let T_Q be the set of triggers of Q. T_Q can be considered as a directed graph with triggers acting as nodes and having an edge from t to t' iff $t \prec t'$. A cycle in T_Q would indicate a conflict of the members of the cycle. Therefore, one can sort the transitions topologically into an ordered sequence $t_1, \ldots, t_k$ such that $T_Q = \{t_1, \ldots, t_k\}$ and $t_i \prec t_j$ implies $i < j$ for every pair of indices i, j. As a consequence, $j < i$ implies $t_i \not\prec t_j$, which is equivalent to $\mathit{Follow}(Q, t_j) \cap \mathit{Follow}^*(Q, t_i) = \varnothing$. $\square$

Algorithm SelectTrigger shows the pseudo-code of the effective decision procedure that reliably identifies the unique right trigger when the ASR parser has reached configuration $\mathcal{S} \centerdot g'$ with $\mathit{top}(\mathcal{S}) = Q^\tau$ for a state $Q \in \mathcal{Q}$, node mapping τ, and rest graph g''. SelectTrigger is called with the current state Q, its node mapping τ, and the graphs g' as well as g'' as parameters. The procedure returns '*failure*' if it is guaranteed that there is no successful sequence processing g'' entirely. Note that the procedure requires an ordered sequence of all triggers as described in Lemma 9.5, i.e., it does not work if a state has conflicting triggers.

The following lemma states that SelectTrigger can reliably identify the unique right trigger:

Lemma 9.6. *Let $\mathcal{S} \centerdot g'$ be any configuration reached by the ASR parser and $\mathit{top}(\mathcal{S}) = Q^\tau$ where $Q \in \mathcal{Q}$ is a state and τ a node mapping.*

For every graph $g'' \in \mathcal{G}_\mathcal{T}$ such that $\mathcal{S} \centerdot g' \models^* \mathcal{S}_\mathrm{A} \centerdot g'g''$ with $\mathit{top}(\mathcal{S}_\mathrm{A}) = Q_\mathrm{A}$, SelectTrigger, when called with parameters (Q, τ, g', g''), returns a trigger t of Q with the following properties:

```
Procedure SelectTrigger(Q,$\tau$,$g'$,$g''$)
  Input  : State $Q \in \mathcal{Q}$, node mapping $\tau$,
           processed graph $g'$, rest graph $g''$
  Output: a trigger $t$ of $Q$ or 'failure'
  let $t_1, \ldots, t_n$ be a sequence of triggers of $Q$ as in Lemma 9.5
  for $i \leftarrow 1$ to $n$ do
      if $g'' \neq \varepsilon$ then
          look for a literal $\boldsymbol{e}$ of $g''$ such that $f_Q^{\tau,g'}(\boldsymbol{e}) \in \mathit{Follow}(Q, t_i)$
          if $\boldsymbol{e}$ exists then return $t_i$
      else if $\$ \in \mathit{Follow}(Q, t_i)$ then return $t_i$
  return 'failure'
```

- *If* $t = \langle \boldsymbol{Start} \to \boldsymbol{Z}\,\centerdot, \iota\rangle$ *is the reduce item causing acceptance, then* $\mathcal{S} = \mathcal{S}_\mathrm{A}$ *and* $g'' = \varepsilon$.
- *If* $t \neq \langle \boldsymbol{Start} \to \boldsymbol{Z}\,\centerdot, \iota\rangle$ *is any other reduce item, there is a stack* $\mathcal{S}'$ *with*

$$\mathcal{S} \centerdot g' \underset{t}{\models} \mathcal{S}' \centerdot g' \models^* \mathcal{S}_\mathrm{A} \centerdot g'g''. \tag{53}$$

- *If* t *is a transition, then* g'' *contains a literal* $\boldsymbol{e}'$ *such that there is a graph* h *and a concrete state* $Q' \in \mathcal{Q}_\mathrm{M}$ *with* $g'' \bowtie \boldsymbol{e}'h$ *and*

$$\mathcal{S} \centerdot g' \underset{t}{\models} \mathcal{S}eQ' \centerdot g'\boldsymbol{e}' \models^* \mathcal{S}_\mathrm{A} \centerdot g'\boldsymbol{e}'h. \tag{54}$$

Proof. Let $\mathcal{S} \centerdot g'$, Q, τ, and g'' be as in the lemma. We distinguish three cases:

(1) $g'' = \varepsilon$ and $Q = Q_\mathsf{A}$. Q_A consists of just the reduce item $it = \langle \boldsymbol{Start} \to \boldsymbol{Z}\,\centerdot, \iota\rangle$ and $\$ \in \mathit{Follow}(Q_A, it)$. Thus SelectTrigger returns it, and the parser terminates by accepting g'.

(2) $g'' = \varepsilon$ and $Q \neq Q_\mathsf{A}$. There must be a nonempty sequence of reduce moves leading to $\mathcal{S}_\mathrm{A} \centerdot g'$. Any reduce item $it \in Q$ with $\$ \in \mathit{Follow}(Q, it)$ causes such a successful sequence. The procedure, therefore, returns such an item it. It cannot return a transition because no successful sequence caused by a transition can process the empty graph.

(3) $g'' \neq \varepsilon$. There is a successful sequence since

$$\mathcal{S} \centerdot g' \models^* \mathcal{S}_\mathrm{A} \centerdot g'g'', \tag{55}$$

which contains at least one shift move. Let $\boldsymbol{f}$ be the literal processed by the first shift move in this particular sequence (55). Then $f_Q^{\tau,g'}(\boldsymbol{f}) \in \mathit{Follow}(Q, t_j)$ where t_j is the transition causing sequence (55). Therefore, the procedure will return a trigger, although not necessarily t_j. Let t_i be the first trigger in $t_1, \ldots, t_n$ such that there is a literal $\boldsymbol{e}$ in g'' with

$$f_Q^{\tau,g'}(\boldsymbol{e}) \in \mathit{Follow}(Q, t_i). \tag{56}$$

We can conclude that no trigger t_j with $j < i$ can cause a successful sequence that processes g'' because its first shift move cannot process any literal of g''. We can also conclude that

$$f_Q^{\tau,g'}(\boldsymbol{e}) \notin \mathit{Follow}^*(Q, t_j) \tag{57}$$

for every $j > i$ by the construction of sequence $t_1, \ldots, t_n$. Assume that t_i does not trigger the "right" move, but any trigger t_j with $j > i$. But (57) yields that literal $\boldsymbol{e}$ cannot be processed by any successful sequence caused by t_j, contradicting (55). Hence, t_i must cause a successful sequence s that processes g'' entirely. If t_i is a reduce item, s has the form (53). Otherwise, t_i is a transition. (56) has shown that any literal $\boldsymbol{e}'$ with $f_Q^{\tau,g'}(\boldsymbol{e}') = f_Q^{\tau,g'}(\boldsymbol{e})$ can be processed in a successful sequence caused by t_i, and we know by (55) and by t_i being the "right" choice, that s has the form (54) for at least one of these literals $\boldsymbol{e}'$.[9] $\square$

SelectTrigger can now be used to predict the next move in every configuration reachable by the ASR parser. This leads to the *predictive shift-reduce (PSR)* parser, the main notion proposed in this paper, which is in fact the ASR parser equipped with SelectTrigger for deterministically selecting the next move:

Definition 9.7 (PSR Parser). A *(PSR parser) configuration* $\mathcal{S}\,\blacksquare\,g|r$ is an ASR parser configuration $\mathcal{S}\,\blacksquare\,g$ together with a rest graph $r \in \mathcal{G}_{\mathcal{T}}$. $\mathcal{S}\,\blacksquare\,g|r$ is *accepting* if $r = \varepsilon$ and $\mathcal{S}\,\blacksquare\,g$ is an accepting ASR parser configuration.

A *PSR move* turns $\mathcal{S}\,\blacksquare\,g|r$ into $\mathcal{S}'\,\blacksquare\,g'|r'$, written $\mathcal{S}\,\blacksquare\,g|r \models \mathcal{S}'\,\blacksquare\,g'|r'$, if *SelectTrigger*$(Q, \tau, g, r)$ returns trigger t, $\mathcal{S}\,\blacksquare\,g \underset{t}{\models} \mathcal{S}'\,\blacksquare\,g'$ is an ASR move, and $gr \bowtie g'r'$.

A PSR parser configuration $\mathcal{S}\,\blacksquare\,g'|g''$ is *reachable* if $Q_0^\iota\,\blacksquare\,\varepsilon, g \models^* \mathcal{S}\,\blacksquare\,g'|g''$. A PSR parser *accepts* a graph $g \in \mathcal{G}_{\mathcal{T}}$ if there is a reachable accepting configuration $\mathcal{S}\,\blacksquare\,g|\varepsilon$.

Theorem 9.8. *Let $g \in \mathcal{G}_{\mathcal{T}}$. The PSR parser accepts g if and only if $\boldsymbol{Z} \underset{\mathrm{rm}}{\Longrightarrow}^* g$.*

Moreover, in every step the trigger used to select the next move of the PSR parser is uniquely determined by the current configuration.

Proof. The first part of the theorem is an immediate consequence of Thm. 8.6 and Lemma 9.6. SelectTrigger chooses the trigger that causes the next move taken by the PSR parser in a deterministic way, yielding the second part of the theorem. $\square$

Note, however, that the parser is still nondeterministic, despite the fact that it chooses the trigger causing the next parser move for every configuration deterministically. The reason is that a transition, if it is chosen as a trigger, does not uniquely determine the literal to be processed by the shift move to be made. For instance, the ASR parser moves shown in Fig. 15, which are also valid PSR parser moves, choose edge e^{12} in the third move, but could have chosen e^{13} instead, keeping e^{12} for later. There are thus two different sequences of parser moves that both prove the validity of the given input graph, i.e., the PSR parser is nondeterministic. However, this nondeterminism is harmless as it does not make a difference when it comes to acceptance.

The *Grappa* tool implemented by the author Mark Minas generates PSR parsers based on the construction of the dCFA and the analysis of the three criteria outlined above. Table 1 summarizes test results for some HR grammars. The columns under "Grammar" indicate the size of the grammar in terms of

---

[9] When we further assume that the grammar has the *free edge choice property* [12], s has the form (54) for every literal $\boldsymbol{e}'$ with $f_Q^{\tau,g'}(\boldsymbol{e}') = f_Q^{\tau,g'}(\boldsymbol{e})$. A discussion of this property is out of scope of the present paper, however.

the maximal arity of nonterminals (A), number of nonterminals (N), number of terminals (T) and number of rules (R). The columns under "dCFA" indicate the size of the generated dCFA in terms of the number of states (S), the overall number of items (P) and the number of transitions (T). The number of conflicting sets in the dCFA is shown in the column "Conflicts". Note that the PSR parser can successfully be generated for the grammars without any conflicts. For the others, the parser generator fails with a message pointing out that the grammar is not conflict-free. We refer the reader to [24, Sect. 6] for runtime measurements of PSR parsers that confirm that it runs in linear time, for all practical purposes.

Table 1: Test results of some HR grammars.

Example	Grammar				dCFA			Conflicts
	A	N	T	R	S	P	T	
Persuade (Example 2.8)	4	1	3	5	9	36	20	–
Trees (Example 4.2)	1	2	1	3	4	10	4	–
$a^n b^n c^n$ [12]	4	3	3	5	14	22	14	–
Nassi-Shneiderman diagrams [30]	4	3	3	6	12	78	59	–
Palindromes (Cor. 10.5)	2	2	2	7	12	32	19	–
Arithmetic expressions	2	4	5	7	12	34	22	–
Series-parallel graphs	2	2	1	4	7	63	32	3
Flowcharts (Example 7.14)	2	3	4	6	14	75	50	4

10. Comparison with String Parsing and Top-Down Graph Parsing

PSR parsing can be compared with SLR(1) string parsing if we represent strings as graphs, and context-free string grammars as HR grammars.

The *chain graph* $w^\bullet$ of a string $w = a_1 \cdots a_n \in A^*$ (of length $n \geq 0$) consists of n edge literals $a_i(x_{i-1}, x_i)$ over $n+1$ distinct nodes $x_0, \ldots, x_n$. (The empty string ε is represented by an isolated node.)

The HR rule of a context-free rule $A \to \alpha$ (where $A \in \mathcal{N}$ and $\alpha \in \Sigma^*$) is $\mathcal{A}^\bullet \to \alpha^\bullet$. For the purpose of this section, we represent an ε-rule $A \to \varepsilon$ by a rule that maps both nodes of $A^\bullet$ to the only node in $\varepsilon^\bullet$. Such rules are called "*merging*" in [12].

For technical simplicity, our definition of hyperedge replacement does not include merging rules. However, while context-free grammars and HR grammars can be *cleaned*, i.e., transformed into equivalent grammars with neither ε-rules nor merging rules, this process may destroy their SLL(1) and PTD property, respectively. Thus, for the sake of generality it is useful to deal with such grammars as they are.

Definition 10.1 (Chain Graph Grammar). The *chain graph grammar* of a context-free grammar G with a finite set $\mathcal{P}$ of rules and a start symbol Z is the

HR grammar $G^\bullet = (\Sigma, \mathcal{T}, \mathcal{P}^\bullet, Z')$ with the rules $\mathcal{P}^\bullet = \{Z'() \to Z^\bullet)\} \cup \{A^\bullet \to \alpha^\bullet \mid (A \to \alpha) \in \mathcal{P}\}$, where $Z' \in \mathcal{N}$ does not occur in $\mathcal{P}$.

It is easy to see that the HR language of $G^\bullet$ is $\mathcal{L}(G^\bullet) = \{w^\bullet \mid w \in \mathcal{L}(G)\}$. For the discussion of generative power, let $\mathrm{SLR}^\bullet(1)$ denote the chain graph grammars of SLR(1) string grammars, and $\mathrm{PSR}^\bullet$ the class of PSR chain graph grammars. The following can easily be shown by inspection of the automata of string and HR grammars.

Proposition 10.2. *For every* SLR(1) *grammar G without ε-rules, $G^\bullet \in \mathrm{PSR}^\bullet$.*

In recent work [12], the authors have lifted simple deterministic top-down string parsing using one symbol of lookahead, known as SLL(1) parsing, to *predictive top-down parsing* (PTD) for HR grammars. Let $\mathrm{SLL}^\bullet(1)$ denote the chain graph HR grammars of SLL(1)-parsable string grammars, and $\mathrm{PTD}^\bullet$ the class of PTD chain graph grammars. The following relation has been established in that paper:

Theorem 10.3 ([12, Thm. 2]). $\mathrm{SLL}^\bullet(1) \subseteq \mathrm{PTD}^\bullet$.

A recent result of [23] concerning SLL(1) and SLR(1) string grammars allows to establish a relation between $\mathrm{SLL}^\bullet(1)$ and $\mathrm{SLR}^\bullet(1)$ chain graph grammars.

Theorem 10.4. *The cleaned version of a grammar in* $\mathrm{SLL}^\bullet(1)$ *is in* $\mathrm{PSR}^\bullet$.

Proof. By [23, Thm. 7] the cleaned version $\tilde{G}$ of a grammar $G \in \mathrm{SLL}(1)$ is in SLR(1). It is easy to check that the cleaned version of $G^\bullet$ coincides with $\tilde{G}^\bullet$. Since $\tilde{G}$ is SLR(1), Prop. 10.2 establishes that $\tilde{G}^\bullet \in \mathrm{PSR}^\bullet$. ☐

The grammar classes $\mathrm{PTD}^\bullet$ and $\mathrm{PSR}^\bullet$ are strictly more powerful than $\mathrm{SLR}^\bullet(1)$.

Corollary 10.5. *There are chain graph languages that cannot be generated by any* $\mathrm{SLR}^\bullet(1)$ *grammar, but have grammars in both* $\mathrm{PTD}^\bullet$ *and* $\mathrm{PSR}^\bullet$.

Proof. The language of *palindromes* over $V = \{a, b\}$, i.e., all strings reading the same backward as forward, can be generated by the unambiguous grammar G with rules $Z \to P$ and $P \to a \mid aa \mid aPa \mid b \mid bb \mid bPb$. Since the language cannot be recognized by a deterministic stack automaton [33, Prop. 5.10], this language neither has an LL(k) parser, nor an LR(k) parser. However, $G \in \mathrm{PTD}^\bullet$ by [12, Theorem 2] and $G \in \mathrm{PSR}^\bullet$, see Table 1. ☐

Fig. 16 summarizes the relations between HR chain graph grammars. We conjecture that Thm. 10.4 can be lifted to the general case, along the lines of the proof of [23, Thm. 7], but this will be rather tedious, as it involves many details of the construction of PTD and PSR parsers. The “proof” of this result given in [14], where it was formulated as Theorem 1, is wrong.

$$\begin{tikzcd}
\mathrm{PTD} \arrow[r, dashed, "\textit{future work}"] & \mathrm{PSR} \\
\mathrm{PTD}^\bullet \arrow[u] & \mathrm{PSR}^\bullet \arrow[u] \\
\mathrm{SLL}^\bullet(1) \arrow[u, "\text{Thm. 10.3}", "\text{Cor. 10.5}"'] \arrow[r, "\text{Thm. 10.4}"'] & \mathrm{SLR}^\bullet(1) \arrow[u, "\text{Cor. 10.5}", "\text{Prop. 10.2}"']
\end{tikzcd}$$

Figure 16: Relation of HR chain graph grammars (solid arrows indicate proper inclusions)

11. Conclusions

We have devised a predictive shift-reduce (PSR) parsing algorithm for HR grammars, along the lines of SLR(1) string parsing, thus continuing the work begun in [14] by formalizing the construction of PSR parsers and proving its correctness. For chain graphs, PSR has greater generative power than SLR(1) and predictive top-down (PTD) parsing [12]. Checking PSR-parsability is complicated enough, but easier than for PTD, as we do not need to consider HR rules that merge nodes of their left-hand sides. PSR parsers also work more efficiently than PTD parsers: while PTD parsers require quadratic time in the worst case, PSR parsers run in linear time for all practical purposes. The reader is encouraged to download the *Grappa* generator of PTD and PSR parsers and to conduct own experiments.[9]

Related Work

Much related work on graph parsing has been done for graph grammars based on context-free node replacement [16]. In these grammars, a node v is replaced by a graph R, where embedding instructions specify what happens to the edges incident in v; in general, such an edge can just be deleted, or turned around, or replicated and directed towards different nodes of R. Node replacement has greater generative power, but is difficult to handle for general embedding instructions. So papers on parsing for node replacement graph grammars restrict these instructions. The earliest ones (to our knowledge), by T. Pavlides, T.W. Pratt, and P. Della Vigna and C. Ghezzi, [31, 32, 7], appeared well before visual user interfaces supported input and processing of diagrams by computers. R. Franck [20] has extended precedence string parsing to graphs, in order to implement a "*two-dimensional programming language*" based on Algol-68. W. Kaul corrected and extended this idea of parsing [25]. This parser is linear, and can cope with ambiguous grammars, but fails to parse some languages that are both PSR- and PTD-parsable languages, like the trees of Example 4.2.

---

[9] The *Grappa* tool is available at www.unibw.de/inf2/grappa; the examples mentioned in Table 1 can be found there as well.

A parsing algorithm following the idea of the well-known Cocke-Younger-Kasami algorithm was proposed and investigated by C. Lautemann [27] who gave a sufficient condition under which this algorithm is polynomial. However, even if the condition is met, the degree of the polynomial depends on the grammar. The algorithm was recently refined by D. Chiang et al. [4], making it more practical but without changing its general characteristics. An alternative algorithm developed by W. Vogler in [34] and generalized by F. Drewes in [9] guarantees a cubic running time at the expense of employing a very strong connectedness requirement. Due to this requirement it seems fair to say that this algorithm is mainly of theoretical interest. A promising approach for certain types of applications, especially for graph languages appearing in computational linguistics, has recently been proposed by S. Gilroy, A. Lopez, and S. Maneth [21]. This parsing algorithm applies to Courcelle's "regular" graph grammars [6] and runs in linear time.

Over the years, M. Flasiński and his group have developed top-down and bottom-up parsing techniques for pattern recognition [17, 18, 19]. The graph classes they consider are very restricted: rooted directed acyclic graphs with ordered nodes. Their parsers are also linear, but this is achieved by forbidding all concepts that make graph parsing essentially different from string parsing. According to our knowledge, another early attempt at *LR*-like graph parsers by H.J. Ludwigs [29] has never been completed.

G. Costagliola's positional grammars [5] are used to specify visual languages, but they can also describe certain HR languages. Although they are parsed in an LR-like fashion, many decisions are deferred until the parser is actually executed, in order to avoid complex analyzes of the grammar when the parsers are generated. In contrast, the PSR parser generator implemented in the *Grappa* tool performs an elaborate static analysis of the grammar. It includes the detection of conflicts that prevent the parser from running into situations where, despite the use of a dCFA, a nondeterministic choice must be made (i.e., backtracking must be employed). It also checks and makes use of other properties, such as the so-called free-edge-choice property, and the existence of uniquely determined start nodes. As mentioned before, the precise formalization and discussion of these analysis techniques will be presented in a follow-up paper.

The CYK-style parsers for unrestricted HR grammars (plus edge-embedding rules) implemented in DiaGen [30] work for practical input with hundreds of nodes and edges, although their worst-case complexity is exponential. A closer comparison to PTD and PSR parsers shows its limits with larger input [24, Sect. 6].

Future Work

Like PTD parsing, PSR parsing can be lifted to contextual HR grammars [10, 11], a class of graph grammars that is more relevant for the practical definition of graph languages. This is another part of future work.

A still open challenge is to find a HR (or contextual HR) language that has a PSR parser, but no PTD parser. The corresponding example for LL(k) and LR(k) string languages exploits that strings are always parsed from left to right—the palindrome example shows that this is not the case for PTD and PSR parsers. Another challenge concerning generative power has already been mentioned in Sect. 10: we are working on a theorem relating the generative power of PTD-parsable and PSR-parsable HR grammars, as it is indicated by the dashed arrow in Fig. 16 above.

References

[1] IJ.J. Aalbersberg, Andrzej Ehrenfeucht, and Grzegorz Rozenberg. On the membership problem for regular DNLC grammars. *Discrete Applied Mathematics*, 13:79–85, 1986.

[2] Alfred V. Aho and Jeffrey D. Ullman. *The Theory of Parsing, Translation, and Compiling*, volume I: Parsing. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1972.

[3] Laura Banarescu, Claire Bonial, Shu Cai, Madalina Georgescu, Kira Griffitt, Ulf Hermjakob, Kevin Knight, Philipp Koehn, Martha Palmer, and Nathan Schneider. Abstract meaning representation for sembanking. In *Proc. 7th Linguistic Annotation Workshop, ACL 2013 Workshop*, 2013.

[4] David Chiang, Jacob Andreas, Daniel Bauer, Karl Moritz Hermann, Bevan Jones, and Kevin Knight. Parsing graphs with hyperedge replacement grammars. In *Proc. 51st Ann. Meeting of the Assoc. for Computational Linguistic (Vol. 1: Long Papers)*, pages 924–932, 2013.

[5] G. Costagliola, A. De Lucia, S. Orefice, and G. Tortora. A parsing methodology for the implementation of visual systems. *IEEE Transactions on Software Engineering*, 23:777–799, 1997.

[6] Bruno Courcelle. The monadic second-order logic of graphs V: on closing the gap between definability and recognizability. *Theoretical Computer Science*, 80:153–202, 1991.

[7] Pierluigi Della Vigna and Carlo Ghezzi. Context-free graph grammars. *Information and Control*, 37(2):207–233, 1978.

[8] Franklin L. DeRemer. Simple LR(k) grammars. *Comm. ACM*, 14(7):453–460, 1971.

[9] Frank Drewes. Recognising k–connected hypergraphs in cubic time. *Theoretical Computer Science*, 109:83–122, 1993.

[10] Frank Drewes and Berthold Hoffmann. Contextual hyperedge replacement. *Acta Informatica*, 52:497–524, 2015.

[11] Frank Drewes, Berthold Hoffmann, and Mark Minas. Contextual hyperedge replacement. In *Proc. Applications of Graph Transformation with Industrial Relevance (AGTIVE'11)*, volume 7233 of *LNCS*, pages 182–197, 2012.

[12] Frank Drewes, Berthold Hoffmann, and Mark Minas. Predictive top-down parsing for hyperedge replacement grammars. In *Proc. 8th Intl. Conf. on Graph Transformation (ICGT 2015)*, volume 9151 of *LNCS*, pages 19–34, 2015.

[13] Frank Drewes, Berthold Hoffmann, and Mark Minas. Approximating Parikh images for generating deterministic graph parsers. In *Software Technologies: Applications and Foundations - STAF 2016 Collocated Workshops: DataMod, GCM, HOFM, MELO, SEMS, VeryComp, Vienna, Austria, July 4-8, 2016, Revised Selected Papers*, volume 9946 of *LNCS*, pages 112–128, 2016.

[14] Frank Drewes, Berthold Hoffmann, and Mark Minas. Predictive shift-reduce parsing for hyperedge replacement grammars. In *Proc. 8th Intl. Conf. on Graph Transformation (ICGT 2017)*, volume 10373 of *Lecture Notes in Computer Science*, pages 106–122, 2017.

[15] Frank Drewes and Anna Jonsson. Contextual hyperedge replacement grammars for abstract meaning representations. In *13th Intl. Workshop on Tree-Adjoining Grammar and Related Formalisms (TAG+13)*, pages 102–111, 2017.

[16] Joost Engelfriet and Grzegorz Rozenberg. Node replacement graph grammars. In Grzegorz Rozenberg, editor, *Handbook of Graph Grammars and Computing by Graph Transformation. Vol. I: Foundations*, chapter 1, pages 1–94. World Scientific, Singapore, 1997.

[17] Mariusz Flasiński. Characteristics of edNLC-graph grammar for syntactic pattern recognition. *Computer Vision, Graphics, and Image Processing*, 47(1):1–21, 1989.

[18] Mariusz Flasiński. Power properties of NLC graph grammars with a polynomial membership problem. *Theor. Comput. Sci.*, 201(1-2):189–231, 1998.

[19] Mariusz Flasiński and Zofia Flasińska. Characteristics of bottom-up parsable edNLC graph languages for syntactic pattern recognition. In Leszek J. Chmielewski, Ryszard Kozera, Bok-Suk Shin, and Konrad W. Wojciechowski, editors, *Computer Vision and Graphics - International Conference, ICCVG 2014, Warsaw, Poland, September 15-17, 2014. Proceedings*, volume 8671 of *Lecture Notes in Computer Science*, pages 195–202. Springer, 2014.

[20] Reinhold Franck. A class of linearly parsable graph grammars. *Acta Informatica*, 10(2):175–201, 1978.

[21] S. Gilroy, A. Lopez, and S. Maneth. Parsing graphs with regular graph grammars. In *Proc. 6th Joint Conf. on Lexical and Computational Semantics (*SEM 2017)*, pages 199–208, 2017.

[22] Annegret Habel. *Hyperedge Replacement: Grammars and Languages*, volume 643 of *LNCS*. 1992.

[23] Berthold Hoffmann. Cleaned SLL(1) grammars are SLR(1). Technical Report 17-1, Studiengang Informatik, Universität Bremen, 2017. `http://www.informatik.uni-bremen.de/~hof/papers/sllr.pdf`.

[24] Berthold Hoffmann and Mark Minas. Generating efficient predictive shift-reduce parsers for hyperedge replacement grammars. In *Proc. 8th International Workshop on Graph Computation Models (GCM 2017), Satellite of ICGT 2017*, 2017. Appears in vol. 10748 of Lecture Notes of Computer Science.

[25] Manfred Kaul. Practical applications of precedence graph grammars. In Hartmut Ehrig, Manfred Nagl, Grzegorz Rozenberg, and Azriel Rosenfeld, editors, *Graph-Grammars and Their Application to Computer Science*, volume 291 of *LNCS*, pages 326–342, 1986.

[26] Donald E. Knuth. On the translation of languages from left to right. *Information and Control*, 8(6):607 – 639, 1965.

[27] Clemens Lautemann. The complexity of graph languages generated by hyperedge replacement. *Acta Informatica*, 27:399–421, 1990.

[28] Philip M. Lewis II and Richard Edwin Stearns. Syntax-directed transduction. *J. ACM*, 15(3):465–488, 1968.

[29] Helmut J. Ludwigs. A LR-like analyzer algorithm for graphs. In Reinhard Wilhelm, editor, *GI - 10. Jahrestagung, Saarbrücken, 30. September - 2. Oktober 1980, Proceedings*, volume 33 of *Informatik-Fachberichte*, pages 321–335, 1980.

[30] Mark Minas. Diagram editing with hypergraph parser support. In *Proc. 1997 IEEE Symposium on Visual Languages (VL'97), Capri, Italy*, pages 226–233, 1997.

[31] Theodosios Pavlidis. Linear and context-free graph grammars. *J. ACM*, 19(1):11–22, 1972.

[32] Terrence W. Pratt. Pair grammars, graph languages and string-to-graph translations. *Journal of Computer and System Sciences*, 5:560–595, 1971.

[33] S. Sippu and E. Soisalon-Soininen. *Parsing Theory I: Languages and Parsing*, volume 15 of *EATCS Monographs in Theoretical Computer Science*. 1988.

[34] Walter Vogler. Recognizing edge replacement graph languages in cubic time. In H. Ehrig, H.-J. Kreowski, and G. Rozenberg, editors, *Proc. Fourth Intl. Workshop on Graph Grammars and Their Application to Comp. Sci.*, volume 532 of *LNCS*, pages 676–687. 1991.