Extending Predictive Shift-Reduce Parsing to Contextual Hyperedge Replacement Grammars

Frank Drewes[1], Berthold Hoffmann[2], and Mark Minas[3](✉)

[1] Umeå universitet, Umeå, Sweden
drewes@cs.umu.se
[2] Universität Bremen, Bremen, Germany
hof@uni-bremen.de
[3] Universität der Bundeswehr München, Neubiberg, Germany
mark.minas@unibw.de

Abstract. Parsing with respect to grammars based on hyperedge replacement (HR) is NP-hard in general, even for some fixed grammars. In recent work, we have devised predictive shift-reduce parsing (PSR), a very efficient algorithm that applies to a wide subclass of HR grammars. In this paper, we extend PSR parsing to contextual HR grammars, a moderate extension of HR grammars that have greater generative power, and are therefore better suited for the practical specification of graph and diagram languages. Although the extension requires considerable modifications of the original algorithm, it turns out that the resulting parsers are still very efficient.

Keywords: Graph grammar · Graph parsing · Contextual hyperedge replacement

1 Introduction

Grammars based on hyperedge replacement (HR) generate a well-studied class of context-free graph languages [16]. However, their generative power is too weak; e.g., their languages are known to have bounded treewidth [16, Thm. IV.3.12(7)]. Since this even excludes a language as simple as that of all graphs, HR grammars cannot reasonably be advocated for specifying graph models in general.

An example illustrating this weakness of hyperedge replacement is provided by "unstructured" flowcharts with jumps (see Sect. 6). Since jumps can target any location in the program, an edge that represents such a jump may point to any arbitrary node (representing a program location). Inserting such edges is beyond what hyperedge replacement can do because it would require nonterminal hyperedges of unbounded arity, such as the adaptive star grammars of [8].

A similar example is Abstract Meaning Representation [1], a representation of the meaning of natural language sentences that is being heavily studied

E. Guerra and F. Orejas (Eds.): ICGT 2019, LNCS 11629, pp. 55–72, 2019.
https://doi.org/10.1007/978-3-030-23611-3_4

in computational linguistics. Coreferences caused by, e.g., pronouns that refer to entities mentioned elsewhere in a sentence, give rise to edges that point to nodes which may be almost anywhere else in the graph. Hence, again, hyperedge replacement is too weak.

Contextual hyperedge replacement (CHR) has been devised as a moderate extension of HR that overcomes such restrictions, while preserving many other formal properties of HR [7,9]. Rather than having left-hand sides consisting of nonterminal edges only, CHR rules can have additional isolated *context nodes* in their left-hand side, to which the right-hand side can attach edges. Hence, CHR can attach edges to already generated nodes elsewhere in the graph, but the gain in power is limited as the mechanism lacks control over which nodes to choose. (The application conditions of [17] are not yet supported by Grappa.)

In recent work, we have devised a very efficient predictive shift-reduce (PSR) parsing algorithm for a subclass of HR grammars. In this paper, we extend this algorithm to contextual HR grammars. Its implementation in the graph-parser distiller Grappa[1] turned out to be smooth, and yields parsers that are as efficient as those for the context-free case. This perhaps surprisingly good result is due to the fact that both parsers consume one edge after another and apply rules backwards until the start symbol is reached. As in the context-free case, the grammar analysis by the distiller ensures that suitable edges can be chosen in constant time, and backtracking is avoided. Hence, the overall running time of the generated parser remains linear.

The rest of this paper is structured as follows. Section 2 introduces CHR grammars. In Sect. 3 we recall PSR parsing and discuss the point where it has to be modified for CHR grammars. A particular normal form needed to parse CHR grammars is introduced in Sect. 4, before we discuss the analysis of lookahead in Sect. 5. In Sect. 6 we discuss a more realistic example grammar (of flowcharts), compare the efficiency of different parsers for this grammar, and evaluate some CHR grammars wrt. PSR-parsability with Grappa. We conclude by summarizing the results obtained so far, and indicate related and future work in Sect. 7. Due to lack of space, our presentation is driven by a small artificial example, and properties like *unique start nodes* and *free edge choice* are not discussed here in order to keep the paper focused. The complete constructions and proofs for the base case, PSR parsing for HR grammars, can be found in [13].

2 Contextual Hyperedge Replacement Grammars

We first compile the basic notions and notation used in this paper. Throughout the paper, $\mathbb{N}$ denotes the non-negative integers and A^* denotes the set of all finite sequences over a set A, with ε denoting the empty sequence.

We let X be a global, countably infinite supply of *nodes* or *vertices*.

Definition 1 (Graph). An *alphabet* is a set Σ of *symbols* together with an *arity function* $arity\colon \Sigma \to \mathbb{N}$. Then a *literal* $\boldsymbol{e} = \mathsf{a}^{x_1 \cdots x_k}$ *over* Σ consists of a

[1] Available from its implementor Mark Minas under www.unibw.de/inf2/grappa.

symbol $\mathsf{a} \in \Sigma$ and an *attachment* $x_1 \cdots x_k$ of $k = \mathit{arity}(\mathsf{a})$ pairwise distinct nodes $x_1, \ldots, x_k \in X$. We 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 graphs are sequences of literals, i.e., two graphs $\langle V, \varphi \rangle$ and $\langle V', \varphi' \rangle$ are considered to differ even if $V = V'$ and φ' is just a permutation of φ. However, such graphs are considered equivalent, denoted by the equivalence relation $\bowtie$. "Ordinary" graphs would rather be represented using multisets of literals. 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. To make sure that this is the case, our parsers always treat the remaining (not yet consumed) edge literals as a multiset rather than a sequence.

An injective function $\varrho\colon X \to X$ is called a *renaming*. Moreover, γ^ϱ denotes the graph obtained by renaming all nodes in γ according to ϱ.

For a graph $\gamma = \langle V, \varphi \rangle$, we use the notations $X(\gamma) = V$ and $\mathit{lit}(\gamma) = \varphi$. We define the *concatenation* of two graphs $\alpha, \beta \in \mathcal{G}_\Sigma$ as $\alpha\beta = \langle X(\alpha) \cup X(\beta), \mathit{lit}(\alpha)\, \mathit{lit}(\beta) \rangle$. 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 attached nodes.

We now recall contextual hyperedge replacement from [7,9]. To keep the technicalities simple, we omit node labels. Adding them does not pose any technical or implementational difficulties. Node labels are actually available in Grappa, but discussing them here would only complicate the exposition.[2]

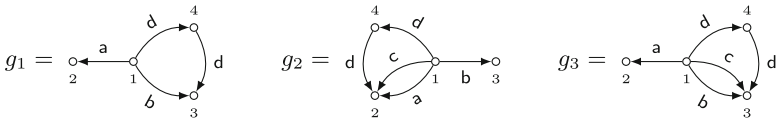
Definition 2 (CHR Grammar). Let the alphabet Σ be partitioned into disjoint subsets $\mathcal{N}$ and $\mathcal{T}$ of *nonterminals* and *terminals*, respectively. A *contextual hyperedge replacement rule* $r = (\alpha \to \beta)$ (a *rule* for short) has a graph $\alpha \in \mathcal{G}_\Sigma$ with a single literal $\mathit{lit}(\alpha) = \boldsymbol{A} \in \mathit{Lit}_\mathcal{N}$ as its *left-hand side*, and a graph $\beta \in \mathcal{G}_\Sigma$ with $X(\alpha) \subseteq X(\beta)$ as its *right-hand side*. The nodes in $X(\alpha) \setminus X(\boldsymbol{A})$ are called *context nodes* of r. A rule without context nodes is called *context-free*.

Consider a graph $\gamma = \delta \boldsymbol{A'} \delta' \in \mathcal{G}_\Sigma$ and a rule r as above. A renaming μ is a *match* (of r to γ) if $\boldsymbol{A}^\mu = \boldsymbol{A'}$ and[3]

$$X(\gamma) \cap X(\beta^\mu) \subseteq X(\alpha^\mu) \subseteq X(\gamma). \tag{1}$$

[2] Contextual hyperedge replacement with application conditions, as originally introduced in [17], would require a more significant extension the difficulties of which we have not yet studied. Investigating this will be a topic of future work; cf. Sect. 7.

[3] This condition makes sure that all nodes that are introduced on the right-hand side β of a rule are renamed so that they are distinct from all nodes that do already occur in graph γ, whereas all other nodes are renamed to nodes that occur in γ.



$$\underline{\mathsf{Z}} \Rightarrow_z \mathsf{A}^1\underline{\mathsf{B}^1} \Rightarrow_{b_2} \mathsf{A}^1\underline{\mathsf{D}^{13}}\mathsf{b}^{13} \Rightarrow_d \underline{\mathsf{A}^1}\mathsf{d}^{14}\mathsf{d}^{43}\mathsf{b}^{13} \Rightarrow_a \mathsf{a}^{12}\mathsf{b}^{13}\mathsf{d}^{14}\mathsf{d}^{43} = g_1 \quad (2)$$

$$\underline{\mathsf{Z}} \Rightarrow_z \mathsf{A}^1\underline{\mathsf{B}^1} \Rightarrow_{b_1} \underline{\mathsf{A}^1}\mathsf{C}^1\mathsf{b}^{13} \Rightarrow_a \mathsf{a}^{12}\underline{\mathsf{C}^1}\mathsf{b}^{13} \Rightarrow_c \mathsf{a}^{12}\underline{\mathsf{D}^{12}}\mathsf{c}^{12}\mathsf{b}^{13} \Rightarrow_d \mathsf{a}^{12}\mathsf{b}^{13}\mathsf{c}^{12}\mathsf{d}^{14}\mathsf{d}^{42} = g_2 \quad (3)$$

$$\underline{\mathsf{Z}} \Rightarrow_z \mathsf{A}^1\underline{\mathsf{B}^1} \Rightarrow_{b_1} \underline{\mathsf{A}^1}\mathsf{C}^1\mathsf{b}^{13} \Rightarrow_a \mathsf{a}^{12}\underline{\mathsf{C}^1}\mathsf{b}^{13} \Rightarrow_c \mathsf{a}^{12}\underline{\mathsf{D}^{13}}\mathsf{c}^{13}\mathsf{b}^{13} \Rightarrow_d \mathsf{a}^{12}\mathsf{b}^{13}\mathsf{c}^{13}\mathsf{d}^{14}\mathsf{d}^{43} = g_3 \quad (4)$$

Fig. 1. Graphs g_1, g_2, g_3 and their derivations in Γ.

A match μ of r *derives* γ to the graph $\gamma' = \delta\beta^\mu\delta'$. This is denoted as $\gamma \Rightarrow_{r,\mu} \gamma'$, or just as $\gamma \Rightarrow_r \gamma'$. We write $\gamma \Rightarrow_{\mathcal{R}} \gamma'$ for a set $\mathcal{R}$ of rules if $\gamma \Rightarrow_r \gamma'$ for some $r \in \mathcal{R}$, and denote the reflexive-transitive closure of $\Rightarrow_{\mathcal{R}}$ by $\Rightarrow^*_{\mathcal{R}}$, as usual.

A *contextual hyperedge replacement grammar* $\Gamma = (\Sigma, \mathcal{T}, \mathcal{R}, \mathsf{Z})$ (*CHR grammar* for short) consists of finite alphabets $\Sigma, \mathcal{T}$ as above, a finite set $\mathcal{R}$ of rules over Σ, and a *start symbol* $\mathsf{Z} \in \mathcal{N}$ of arity 0. Γ generates the language $\mathcal{L}(\Gamma) = \{g \in \mathcal{G}_{\mathcal{T}} \mid \mathsf{Z} \Rightarrow^*_{\mathcal{R}} g\}$ of terminal graphs. We call a graph g *valid* with respect to Γ if $\mathcal{L}(\Gamma)$ contains a graph g' with $g \bowtie g'$.

Context-free rules are in fact hyperedge replacement (HR) rules as defined in [13, Def. 2.2], and thus CHR grammars with context-free rules only are HR grammars. Note, however, that derivations in [13] are always *rightmost* derivations that require $\delta' \in \mathcal{G}_{\mathcal{T}}$ in every derivation step. Example 1 demonstrates why derivations for contextual grammars cannot be restricted to just rightmost ones.

Our running example is an artificial CHR grammar chosen for the purpose of illustration only. More practical grammars are considered in Sect. 6.

Example 1. The CHR grammars Γ has $\mathcal{N} = \{\mathsf{Z}, \mathsf{A}, \mathsf{B}, \mathsf{C}, \mathsf{D}\}$ and $\mathcal{T} = \{\mathsf{a}, \mathsf{b}, \mathsf{c}, \mathsf{d}\}$. Z has arity 0, whereas A, B, and C have arity 1; all other labels are binary. Γ has the following rules:

$$\mathsf{Z} \xrightarrow{z} \mathsf{A}^x\mathsf{B}^x \qquad \mathsf{B}^x \xrightarrow{b_1|b_2} \mathsf{C}^x\mathsf{b}^{xy} \mid \mathsf{D}^{xy}\mathsf{b}^{xy} \qquad \mathsf{D}^{xy} \xrightarrow{d} \mathsf{d}^{xz}\mathsf{d}^{zy}$$

$$\mathsf{A}^x \xrightarrow{a} \mathsf{a}^{xy} \qquad \mathsf{C}^x{+}y \xrightarrow{c} \mathsf{D}^{xy}\mathsf{c}^{xy}$$

In rule c, $\mathsf{C}^x{+}y$ is a shorthand for the graph $\langle\{x, y\}, \mathsf{C}^x\rangle$ with context node y; the other rules are context-free.

Figure 1 illustrates the three graphs g_1, g_2, g_3 that constitute the language generated by Γ (up to renaming).[4] As usual, nodes are drawn as circles whereas (binary) edges are drawn as arrows with their label ascribed.

Moreover, Fig. 1 shows the derivations of g_1, g_2, g_3. Underlined nonterminal literals are those rewritten in the next derivation step. The rightmost derivation (2) uses just context-free rules to derive g_1. Both derivation (3) and (4)

[4] Thus a CHR grammar is in fact not necessary to describe this tiny language.

use the contextual rule c in their fourth step. There is only a subtle difference: (3) uses node 2 as context node, whereas (4) uses node 3. Neither derivation is rightmost. While (4) could be turned into a rightmost one, there is no rightmost derivation for g_2: In (3), A^1 must be rewritten before C^1 because the rule rewriting C^1 uses 2 as a context node, which has to be created by rewriting A^1. □

3 Making Shift-Reduce Parsing Predictive

In this section, we recall how a nondeterministic (and inefficient) shift-reduce parsing is made predictive (and efficient), by using a characteristic finite-state automaton (CFA) for control, and by inspecting lookahead. The algorithm developed for HR grammars [13] carries over to CHR grammars in many cases; an exception that requires some modification is discussed at the end of this section.

Nondeterministic Shift-Reduce Parsers read an input graph, and keep a stack of literals (nonterminals and terminals) that have been processed so far. They perform two kinds of actions. *Shift* reads an unread literal of the input graph and pushes it onto the stack. *Reduce* can be applied if literals on top of the stack form the right-hand side of a rule (after a suitable match of their nodes). Then the parser pops these literals off the stack and pushes the left-hand side of the rule onto it (using the same match). The parser starts with an empty stack and an input graph, and accepts this input if the stack just contains the start graph Z and the input has been read completely.

Table 1 shows a parse of graph g_3 in Example 1. Each row shows the current stack, which grows to the right, the sequence of literals read so far, and the multiset of yet unread literals. Note that the literals of the input graph can be shifted in any order; the parser has to choose the literals so that it can construct a reverse rightmost derivation of the input graph. The last column in

Table 1. Shift-reduce parse of graph g_3.

#	Stack	Read literals	Unread literals	Action
0	ε	ε	$\{\mathsf{a}^{12}, \mathsf{b}^{13}, \mathsf{c}^{13}, \mathsf{d}^{14}, \mathsf{d}^{43}\}$	shift a^{12}
1	$\underline{\mathsf{a}^{12}}$	a^{12}	$\{\mathsf{b}^{13}, \mathsf{c}^{13}, \mathsf{d}^{14}, \mathsf{d}^{43}\}$	reduce a
2	A^1	a^{12}	$\{\mathsf{b}^{13}, \mathsf{c}^{13}, \mathsf{d}^{14}, \mathsf{d}^{43}\}$	shift d^{14}
3	$\mathsf{A}^1\mathsf{d}^{14}$	$\mathsf{a}^{12}\mathsf{d}^{14}$	$\{\mathsf{b}^{13}, \mathsf{c}^{13}, \mathsf{d}^{43}\}$	shift d^{43}
4	$\mathsf{A}^1\underline{\mathsf{d}^{14}\mathsf{d}^{43}}$	$\mathsf{a}^{12}\mathsf{d}^{14}\mathsf{d}^{43}$	$\{\mathsf{b}^{13}, \mathsf{c}^{13}\}$	reduce d
5	$\mathsf{A}^1\mathsf{D}^{13}$	$\mathsf{a}^{12}\mathsf{d}^{14}\mathsf{d}^{43}$	$\{\mathsf{b}^{13}, \mathsf{c}^{13}\}$	shift c^{13}
6	$\mathsf{A}^1\underline{\mathsf{D}^{13}\mathsf{c}^{13}}$	$\mathsf{a}^{12}\mathsf{d}^{14}\mathsf{d}^{43}\mathsf{c}^{13}$	$\{\mathsf{b}^{13}\}$	reduce c
7	$\mathsf{A}^1\mathsf{C}^1$	$\mathsf{a}^{12}\mathsf{d}^{14}\mathsf{d}^{43}\mathsf{c}^{13}$	$\{\mathsf{b}^{13}\}$	shift b^{13}
8	$\mathsf{A}^1\underline{\mathsf{C}^1\mathsf{b}^{13}}$	$\mathsf{a}^{12}\mathsf{d}^{14}\mathsf{d}^{43}\mathsf{c}^{13}\mathsf{b}^{13}$	$\varnothing$	reduce b_1
9	$\underline{\mathsf{A}^1\mathsf{B}^1}$	$\mathsf{a}^{12}\mathsf{d}^{14}\mathsf{d}^{43}\mathsf{c}^{13}\mathsf{b}^{13}$	$\varnothing$	reduce z
10	Z	$\mathsf{a}^{12}\mathsf{d}^{14}\mathsf{d}^{43}\mathsf{c}^{13}\mathsf{b}^{13}$	$\varnothing$	accept

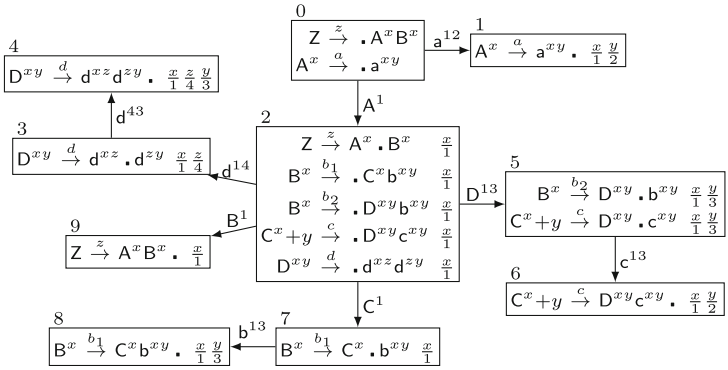


Fig. 2. Characteristic items and states for the steps of the parse of graph g_3 in Table 1.

the table indicates the parser action that yields the situation shown in the next row. Underlined literals on the stack are those popped off the stack by the next reduction step.

We have shown in [13, Sect. 4] that such a parser can find an accepting parse if and only if the input graph is valid. But this parser is highly nondeterministic. For instance, it could start with shifting any literal in step 0, but only shifting a^{12} leads to a successful parse. So it must employ expensive backtracking whenever it runs into a dead end.

Characteristic Finite-State Automata (CFAs) are used to reduce the nondeterminism in shift-reduce parsers. This concept has been transferred from LR parsing for context-free string grammars in [13, Sect. 5–7]. The CFA records *items* of the grammar which the parser is processing, where an item is a rule with a dot in its right-hand side that indicates how far processing has advanced. Figure 2 shows the characteristic items for the steps of the parse in Table 1. The numbering of item sets corresponds to the steps in Table 1. Each of these sets corresponds to a state of the CFA, with a particular renaming of nodes.[5]

In step 0 of the parse, the parser starts with the item $\mathsf{Z} \xrightarrow{z} \mathbf{.}\mathsf{A}^x\mathsf{B}^x$, where the dot at the beginning indicates that nothing has been processed so far. As the dot is before the nonterminal literal A^x, which can be replaced using rule a, the corresponding configuration in Fig. 2 also contains the item $\mathsf{A}^x \xrightarrow{a} \mathbf{.}\mathsf{a}^{xy}$. So the parser can read a literal a^{xy} (with a suitable match of x and y). It cannot process the nonterminal A^x, as only terminals can be shifted; so shifting a^{12} is the only action fitting the grammar in step 0. As a consequence, step 1 is characterized by the sole item $\mathsf{A}^x \xrightarrow{a} \mathsf{a}^{xy}\mathbf{.}\,\underline{x}\,\underline{y}$, which indicates that the right-hand side of rule a has been processed completely and x and y have been matched to nodes 1 and 2, respectively. This implies that a^{12} can be reduced to A^1, which turns item $\mathsf{Z} \xrightarrow{z} \mathbf{.}\mathsf{A}^x\mathsf{B}^x$ of step 0 into item $\mathsf{Z} \xrightarrow{z} \mathsf{A}^x\mathbf{.}\mathsf{B}^x\,\underline{x}$ in step 2. Step 2 contains further items because the dot is in front of nonterminal B^x.

[5] The complete CFA for Example 1 will only be presented in the next section.

Table 2. Wrong shift-reduce parse of graph g_4. Steps 0–4 are essentially the same as in Table 1

#	Stack	Read literals	Unread literals	Action
0	ε	ε	$\{\mathsf{a}^{12}, \mathsf{b}^{12}, \mathsf{d}^{14}, \mathsf{d}^{42}\}$	shift a^{12}
1	$\underline{\mathsf{a}^{12}}$	a^{12}	$\{\mathsf{b}^{12}, \mathsf{d}^{14}, \mathsf{d}^{42}\}$	reduce a
2	A^{1}	a^{12}	$\{\mathsf{b}^{12}, \mathsf{d}^{14}, \mathsf{d}^{42}\}$	shift d^{14}
3	$\mathsf{A}^{1}\mathsf{d}^{14}$	$\mathsf{a}^{12}\mathsf{d}^{14}$	$\{\mathsf{b}^{12}, \mathsf{d}^{42}\}$	shift d^{42}
4	$\mathsf{A}^{1}\underline{\mathsf{d}^{14}\mathsf{d}^{42}}$	$\mathsf{a}^{12}\mathsf{d}^{14}\mathsf{d}^{42}$	$\{\mathsf{b}^{12}\}$	reduce d
5	$\mathsf{A}^{1}\mathsf{D}^{12}$	$\mathsf{a}^{12}\mathsf{d}^{14}\mathsf{d}^{42}$	$\{\mathsf{b}^{12}\}$	shift b^{12}
6	$\mathsf{A}^{1}\underline{\mathsf{D}^{12}\mathsf{b}^{12}}$	$\mathsf{a}^{12}\mathsf{d}^{14}\mathsf{d}^{42}\mathsf{b}^{12}$	$\varnothing$	reduce b_2
7	$\underline{\mathsf{A}^{1}\mathsf{B}^{1}}$	$\mathsf{a}^{12}\mathsf{d}^{14}\mathsf{d}^{42}\mathsf{b}^{12}$	$\varnothing$	reduce z
8	Z	$\mathsf{a}^{12}\mathsf{d}^{14}\mathsf{d}^{42}\mathsf{b}^{12}$	$\varnothing$	accept $\lightning$

Transitions in the CFA move the dot across a literal in some of its items, and match nodes accordingly. The transitions in Fig. 2 are labeled with these literals. Note that the sequence of literals along any path starting in state 0 equals the stack of the step that is reached by the path.

Every shift-reduce parser can be controled by the CFA so that it will only choose actions in accordance with the grammar, which are promising to find a successful parse if the input graph is valid [13, Sect. 9]. Still some sources of nondeterminism remain, which have to be resolved by different means.

Lookahead may be used when a set of items allows several promising actions. For instance, consider step 5 in Fig. 2, where the dot is in front of b^{13} and c^{13} (under the match $\frac{x}{1}\,\frac{y}{3}$), which both occur in the unread input. Only shifting c^{13} leads to a successful parse of g_3. If the parser shifted b^{13} instead, the next steps would reduce for rules b_2 and z, yielding a stack Z, leaving c^{13} unread so that the parse would fail. In such a situation, one must check which literals may follow later when either of the actions is selected. An analysis of grammar Γ (prior to parsing) reveals that selecting c^{13} will allow to shift b^{13} later, whereas selecting b^{13} will never allow to shift c^{13} later. So the predictive shift-reduce (PSR) parser must shift c^{13} in step 5. In general (not for Γ), a HR grammar may have states with *conflicts* where the lookahead does not determine which shift or reduction should be done in some state. Then the grammar is not PSR-parsable [13, Sect. 9]. We will discuss the analysis of lookahead in Sect. 5.

Context Nodes require a modification of the PSR parsing algorithm. For a HR grammar, a PSR parser can always continue its parse to a successful one (for *some* remaining input) as long as all actions comply with the CFA. This does not necessarily hold for CHR grammars.

For instance, consider the invalid graph $g_4 = \mathsf{a}^{12}\mathsf{d}^{14}\mathsf{d}^{42}\mathsf{b}^{12}$. Its parse, shown in Table 2, starts with the same actions 0–5 as for g_3 in Table 1 and Fig. 2. However, only b^{12} is unread in step 5, which is then shifted. Therefore, the parser will eventually accept g_4 as all literals have been read, although g_4 is invalid! In fact, the reduction in step 6 is wrong, because condition (1) in Definition 2 is violated:

The reduce action is the reverse of the derivation $\mathsf{A}^1\mathsf{B}^1 \Rightarrow_{b_2} \mathsf{A}^1\mathsf{D}^{12}\mathsf{b}^{12}$ that creates node 2, which is also created when deriving A^1. But the error happened already in step 5 with its characteristic items $\mathsf{B}^x \xrightarrow{b_2} \mathsf{D}^{xy}\,\centerdot\,\mathsf{b}^{xy}\frac{x}{1}\frac{y}{2}$ and $\mathsf{C}^x{+}y \xrightarrow{c} \mathsf{D}^{xy}\,\centerdot\,\mathsf{c}^{xy}\frac{x}{1}\frac{y}{2}$. Since node 2 has been reused for y, it must be a context node, i.e., the first of these items, which is based on the context-free rule b_2, is not valid in step 5. Apparently, the CFA does not treat context nodes correctly. We shall see in the following section that a CHR grammar like Γ has to be transformed before parsing in order to solve the problem.

4 IOC Normalform

The problem described in the previous section could have been avoided if the parser would "know" in step 5 that node 2 of the nonterminal literal D^{12} is a context node. This would be easy if it held for all occurrences of a D-literal in Γ, but this is not true for the occurrence of D^{xy} in b_2. In the following, we consider only CHR grammars where such situations cannot occur. We require that the label of a nonterminal literal always determines the *roles* of its attached nodes. We then say that a CHR grammar is in *IOC normalform*. Fortunately, every CHR grammar can be transformed into an equivalent IOC normalform. Before we define the IOC normalform in a formal manner, we discuss roles of nodes and profiles of nonterminals.

Nonterminal literals are produced by reduce actions, i.e., if the dot is moved across the nonterminal literal within a rule. For instance, consider literal D^{xy} in item $\mathsf{B}^x \to \centerdot\,\mathsf{D}^{xy}\mathsf{b}^{xy}\frac{x}{1}$ in Fig. 2. Node x is already bound to node 1 before the dot is moved across D^{xy}, whereas y is unbound before, but bound afterwards (to node 3 in step 5). Nodes x and y act like in and out parameters, respectively, of a procedure; we say that x has role I (for "in") and y has role O (for "out"). By combining I and O for x and y, we say that D^{xy} has *profile IO* in this particular situation. However, the situation is different for D^{xy} in item $\mathsf{C}^x{+}y \to \centerdot\,\mathsf{D}^{xy}\mathsf{c}^{xy}\frac{x}{1}$. Node x again has role I, and y is again not bound before moving the dot across D^{xy}. But y must then be bound to the context node of this rule; we say that y has role C (for "context"), and D^{xy} has *profile IC* in this situation. So the profile of D^{xy} is not determined by its label D. This must not happen for a CHR grammar in IOC normalform.

Definition 3. A CHR grammar $\Gamma = (\Sigma, \mathcal{T}, \mathcal{R}, \mathsf{Z})$ is in *IOC normalform* if there is a function $P\colon \mathcal{N} \to \{I, O, C\}^*$ so that, for every rule $(\alpha \to \beta) \in \mathcal{R}$ and every nonterminal literal $\boldsymbol{B} = B^{y_1 \cdots y_m} \in \mathit{Lit}_{\mathcal{N}}$ occurring in β, i.e., $\beta = \delta\boldsymbol{B}\delta'$ for some $\delta, \delta' \in \mathit{Lit}^*_{\Sigma}$, $P(B) = p_1 \cdots p_m$ with

$$p_i = \begin{cases} I & \text{if } y_i \in X(\delta) \\ O & \text{if } y_i \notin X(\delta) \wedge y_i \notin X(\alpha) \\ C & \text{if } y_i \notin X(\delta) \wedge y_i \in X(\alpha) \setminus X(\boldsymbol{A}) \\ p'_j & \text{if } y_i \notin X(\delta) \wedge y_i = x_j \end{cases} \qquad (\text{for } i = 1, \ldots, m)$$

where $\mathit{lit}(\alpha) = \boldsymbol{A} = A^{x_1 \cdots x_k}$ and $P(A) = p'_1 \cdots p'_k$.

Example 2. CHR grammar Γ of Example 1 can be turned into a CHR grammar Γ' in IOC normalform by splitting up the nonterminal label D into D_1 and D_2 and using the following rules:

$$\mathsf{Z} \xrightarrow{z} \mathsf{A}^x\mathsf{B}^x \qquad \mathsf{B}^x \xrightarrow{b_1|b_2} \mathsf{C}^x\mathsf{b}^{xy} \mid \mathsf{D}_1^{xy}\mathsf{b}^{xy} \qquad \mathsf{D}_1^{xy} \xrightarrow{d_1} \mathsf{d}^{xz}\mathsf{d}^{zy}$$
$$\mathsf{A}^x \xrightarrow{a} \mathsf{a}^{xy} \qquad \mathsf{C}^x{+}y \xrightarrow{c} \mathsf{D}_2^{xy}\mathsf{c}^{xy} \qquad \mathsf{D}_2^{xy} \xrightarrow{d_2} \mathsf{d}^{xz}\mathsf{d}^{zy}$$

It can easily be verified that the function P defined by $\mathsf{Z} \mapsto \varepsilon$, $\mathsf{A} \mapsto O$, $\mathsf{B} \mapsto I$, $\mathsf{C} \mapsto I$, $\mathsf{D}_1 \mapsto IO$, and $\mathsf{D}_2 \mapsto IC$ satisfies the conditions of Definition 3. In particular, every D_1-literal has profile IO, and every D_2-literal has profile IC.

The general construction is straightforward: the simplest method is to create, for every nonterminal label B, all copies $B_{P(B)}$ in which B is indexed with its $3^{arity(B)}$ possible profiles. Each rule for B is thus turned into $3^{arity(B)}$ rules, and the nonterminal literals in the right-hand sides are annotated according to Definition 3. Grappa turns this procedure around to avoid the exponential blow-up in most practically relevant cases, as follows. In all rules for a nonterminal label B, assume first that the profile of the left-hand side is $I^{arity(B)}$, and annotate it accordingly. Then annotate the nonterminal labels in all right-hand sides, again following Definition 3. This may give rise to a number of yet unseen annotated nonterminal labels. Create rules for them by copying the respective original rules as before, and repeat until no more new annotations are encountered.

Figure 3 shows the CFA of grammar Γ'. The start state is q_0, indicated by the incoming arc out of nowhere. The CFA has been built in essentially the same

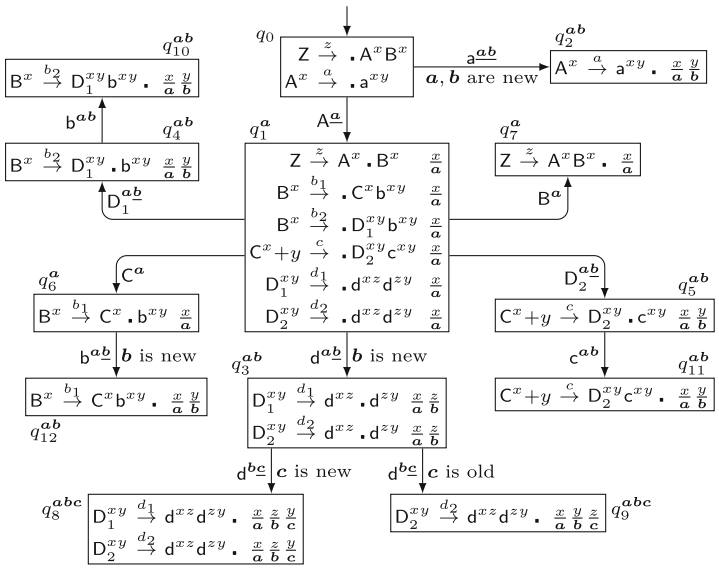


Fig. 3. The characteristic finite automaton of Γ' in Example 2.

way as the characteristic items and states for the steps when parsing the specific graph g_3 in Fig. 2. Instead of the concrete nodes of g_3, we now use *parameters* $\boldsymbol{a}$, $\boldsymbol{b}$, and $\boldsymbol{c}$. They are placeholders, which will be bound to nodes of the particular input graph during parsing. For instance, item $\mathsf{A}^x \xrightarrow{a} \mathsf{a}^{xy} \bullet \frac{x}{1}\frac{y}{2}$ that characterizes step 1 in Fig. 2 corresponds to state $q_2^{\boldsymbol{ab}}$ in Fig. 3 where $\boldsymbol{a}$ and $\boldsymbol{b}$ have been bound to nodes 1 and 2, respectively.

The transitions between states also refer to parameters. For instance, the transition from $q_1^{\boldsymbol{a}}$ to $q_3^{\boldsymbol{ab}}$ means that the dots in the two items $\mathsf{D}_1^{xy} \xrightarrow{d_1} \bullet \mathsf{d}^{xz}\mathsf{d}^{zy}\frac{x}{\boldsymbol{a}}$ and $\mathsf{D}_2^{xy} \xrightarrow{d_2} \bullet \mathsf{d}^{xz}\mathsf{d}^{zy}\frac{x}{\boldsymbol{a}}$ are moved across d^{xz} where x is bound to $\boldsymbol{a}$ and z is yet unbound. The corresponding shift action must select and read a d-edge in the input graph that leaves the node being bound to parameter $\boldsymbol{a}$. The node of the input graph that matches z and is bound to parameter $\boldsymbol{b}$ in state $q_3^{\boldsymbol{ab}}$, also becomes "known" that way. It must not have been read before because d_1 and d_2 are context-free rules. Figure 3 represents the fact that $\boldsymbol{b}$ is bound to the target node of the d-edge by using $\boldsymbol{b}$ as the underlined target node, and the fact that this node has not been read before is indicated by the label "$\boldsymbol{b}$ is new". Using the IOC normalform, this distinction between "new" and "old" nodes makes it possible to handle context nodes correctly (see also the discussion at the end of Sect. 3). It marks the major technical difference between the context-free parser and the contextual one.

The label "$\boldsymbol{c}$ is old" at the transition from $q_3^{\boldsymbol{ab}}$ to $q_9^{\boldsymbol{abc}}$, however, indicates that $\boldsymbol{c}$ is bound to a node of the input graph that has already been read, together with a shifted edge, earlier in the parsing process. This situation can occur although this transition means moving the dot of item $\mathsf{D}_2^{xy} \xrightarrow{d_2} \mathsf{d}^{xz} \bullet \mathsf{d}^{zy}\frac{x}{\boldsymbol{a}}\frac{z}{\boldsymbol{b}}$ across d^{zy} in the context-free rule d_2. Node y of the nonterminal literal corresponds to context node y of the contextual rule c. This is reflected by the profile IC of D_2, which says that the second node of D_2 must be a context node. Further note that $q_8^{\boldsymbol{abc}}$ also contains the same item as $q_9^{\boldsymbol{abc}}$, but $\boldsymbol{c}$ is declared "new" by the corresponding transition. This is so because the context node may be still unread in this situation, and the parser can not yet distinguish whether it is currently processing rule d_2 or d_1, where y is not a context node, indicated by profile IO of D_1. This is demonstrated in the following.

Table 3 shows the only parse that a PSR parser using the CFA in Fig. 3 will try when analyzing graph g_3. Note that this parse corresponds to the unique successful (non-predictive) shift-reduce parse among all possible attempts to parse graph g_3, shown earlier in Table 1. It predicts the unique promising action that keeps it on track towards a successful parse. This is done by keeping, on its stack, an alternating sequence of CFA states and literals processed so far, for instance $q_0\,\mathsf{A}^1\,q_1^1\,\mathsf{d}^{14}\,q_3^{14}\,\mathsf{d}^{43}\,q_8^{143}$ in step 4. The stack contents represent a walk through the CFA from the initial state q_0 to q_8^{143} via q_1^1 and q_3^{14}; the literals between consecutive states correspond to the transitions and their labels. When we ignore the states, the stack equals the stack of the (nondeterministic) shift-reduce parse shown in Table 1. The shift and reduce action of a PSR parser work as follows:

Table 3. PSR parse of g_3 using the CFA in Fig. 3.

#	Stack	Read literals	Unread literals	Action
0	q_0	ε	$\{\mathsf{a}^{12}, \mathsf{b}^{13}, \mathsf{c}^{13}, \mathsf{d}^{14}, \mathsf{d}^{43}\}$	shift a^{12}
1	$q_0\ \underline{\mathsf{a}^{12}\ q_2^{12}}$	a^{12}	$\{\mathsf{b}^{13}, \mathsf{c}^{13}, \mathsf{d}^{14}, \mathsf{d}^{43}\}$	reduce a
2	$q_0\ \mathsf{A}^1\ q_1^1$	a^{12}	$\{\mathsf{b}^{13}, \mathsf{c}^{13}, \mathsf{d}^{14}, \mathsf{d}^{43}\}$	shift d^{14}
3	$q_0\ \mathsf{A}^1\ q_1^1\ \mathsf{d}^{14}\ q_3^{14}$	$\mathsf{a}^{12}\mathsf{d}^{14}$	$\{\mathsf{b}^{13}, \mathsf{c}^{13}, \mathsf{d}^{43}\}$	shift d^{43}
4	$q_0\ \mathsf{A}^1\ q_1^1\ \underline{\mathsf{d}^{14}\ q_3^{14}\ \mathsf{d}^{43}\ q_8^{143}}$	$\mathsf{a}^{12}\mathsf{d}^{14}\mathsf{d}^{43}$	$\{\mathsf{b}^{13}, \mathsf{c}^{13}\}$	reduce d_2
5	$q_0\ \mathsf{A}^1\ q_1^1\ \mathsf{D}_2^{13}\ q_5^{13}$	$\mathsf{a}^{12}\mathsf{d}^{14}\mathsf{d}^{43}$	$\{\mathsf{b}^{13}, \mathsf{c}^{13}\}$	shift c^{13}
6	$q_0\ \mathsf{A}^1\ q_1^1\ \underline{\mathsf{D}_2^{13}\ q_5^{13}\ \mathsf{c}^{13}\ q_{11}^{13}}$	$\mathsf{a}^{12}\mathsf{d}^{14}\mathsf{d}^{43}\mathsf{c}^{13}$	$\{\mathsf{b}^{13}\}$	reduce c
7	$q_0\ \mathsf{A}^1\ q_1^1\ \mathsf{C}^1\ q_6^1$	$\mathsf{a}^{12}\mathsf{d}^{14}\mathsf{d}^{43}\mathsf{c}^{13}$	$\{\mathsf{b}^{13}\}$	shift b^{13}
8	$q_0\ \mathsf{A}^1\ q_1^1\ \underline{\mathsf{C}^1\ q_6^1\ \mathsf{b}^{13}\ q_{12}^{13}}$	$\mathsf{a}^{12}\mathsf{d}^{14}\mathsf{d}^{43}\mathsf{c}^{13}\mathsf{b}^{13}$	$\varnothing$	reduce b
9	$q_0\ \mathsf{A}^1\ q_1^1\ \mathsf{B}^1\ q_7^1$	$\mathsf{a}^{12}\mathsf{d}^{14}\mathsf{d}^{43}\mathsf{c}^{13}\mathsf{b}^{13}$	$\varnothing$	accept

Table 4. PSR parse of the invalid graph g_4 using the CFA in Fig. 3.

#	Stack	Read literals	Unread literals	Action
0	q_0	ε	$\{\mathsf{a}^{12}, \mathsf{b}^{12}, \mathsf{d}^{14}, \mathsf{d}^{42}\}$	shift a^{12}
1	$q_0\ \underline{\mathsf{a}^{12}\ q_2^{12}}$	a^{12}	$\{\mathsf{b}^{12}, \mathsf{d}^{14}, \mathsf{d}^{42}\}$	reduce a
2	$q_0\ \mathsf{A}^1\ q_1^1$	a^{12}	$\{\mathsf{b}^{12}, \mathsf{d}^{14}, \mathsf{d}^{42}\}$	shift d^{14}
3	$q_0\ \mathsf{A}^1\ q_1^1\ \mathsf{d}^{14}\ q_3^{14}$	$\mathsf{a}^{12}\mathsf{d}^{14}$	$\{\mathsf{b}^{12}, \mathsf{d}^{42}\}$	shift d^{42}
4	$q_0\ \mathsf{A}^1\ q_1^1\ \underline{\mathsf{d}^{14}\ q_3^{14}\ \mathsf{d}^{42}\ q_9^{142}}$	$\mathsf{a}^{12}\mathsf{d}^{14}\mathsf{d}^{42}$	$\{\mathsf{b}^{12}\}$	reduce d_2
5	$q_0\ \mathsf{A}^1\ q_1^1\ \mathsf{D}_2^{12}\ q_5^{12}$	$\mathsf{a}^{12}\mathsf{d}^{14}\mathsf{d}^{42}$	$\{\mathsf{b}^{12}\}$	failure

A *shift* action corresponds to an outgoing transition of the state which is currently on top of the stack. For instance, in step 3 with topmost state q_3^{14} there are two transitions leaving $q_3^{\boldsymbol{ab}}$. They both look for a d-edge leaving node 4 in g_3. The only such edge is d^{43}. And, the parser must choose the transition to q_8^{143} because node 3 is "new", i.e., has not yet occurred in the parse.

A *reduce* action may be selected if the topmost state on the stack contains an item with the dot at the end. For instance, consider step 4 with topmost state q_8^{143}. This state in fact contains two items with a dot at their ends: The parser may either reduce according to rule d_1 or d_2; the CFA cannot help the parser with this decision. However, further analysis (see the following section) reveals that only reducing d_2 can lead to a successful parse. The parser, therefore, removes the topmost four elements from the stack (the right-hand side of rule d_2 together with the states in between, indicated by the underline), leaving q_1^1 as the intermediate topmost state. It then selects the transition for the obtained nonterminal literal D_2^{13} that leaves q_1^1, i.e., the transition to q_5^{13}.

The PSR parser accepts g_3 in step 9 because all literals of g_3 have been read and the topmost state is q_7^1 which has the dot at the end of rule z, i.e., a last reduction would produce Z.

Finally Table 4 shows that the PSR parser using CHR grammar Γ' in IOC normalform correctly recognizes that graph g_4 is invalid. It fails in step 5 in state q_5^{12} where it looks for an unread literal c^{12}, but only finds b^{12}, which cannot be shifted, in contrast to the situation shown in Table 2.

5 Lookahead Analysis

The previous section revealed that the CFA does not always provide enough information for the parser to unambiguously select the next action. This is in fact unavoidable (at least if P $\neq$ NP) because PSR parsing is very efficient while HR graph languages in general can be NP-complete. The problem is that the CFA may contain states with items that trigger different actions, for instance state $q_8^{\boldsymbol{abc}}$ in Fig. 3, which allows two different reduce actions. Then we must analyze (prior to parsing) which literals may follow or cannot follow (immediately or later) in a correct parse when either of the possible actions is chosen. This may yield two results: The analysis may either identify a fixed number of (lookahead) literals that the parser must find among the unread literals in order to always predict the correct next action, or there is at least one state for which this is not possible. The latter situation is called a *conflict*. A CHR grammar is PSR only if no state has a conflict. Here, we describe this situation and the peculiarities for CHR grammars by means of grammar Γ' and state $q_8^{\boldsymbol{abc}}$.

Consider the situation when the PSR parser has reached $q_8^{\boldsymbol{abc}}$. We wish to know which literals will be read next or later if either "reduce d_1" or "reduce d_2" is chosen, producing $\mathsf{D}_1^{\boldsymbol{ac}}$ or $\mathsf{D}_2^{\boldsymbol{ac}}$. Figure 4 shows the history of the parser that is relevant in this situation. The parser is either in item I_6 or I_7. If it is in I_7, it must also be in I_5 where its literal D_2^{xy} corresponds to the left-hand side of I_7, and so on. B^x in I_1 corresponds to the left-hand side of either I_3 or I_4. Note that the node renamings in $I_2, \ldots, I_5$ reflect the information available when I_6 or I_7 are reduced. For instance, node y of I_5 will be bound to parameter $\boldsymbol{c}$. However, the choice of context node y of I_5 affects these situations. In this small example, y can be bound either to the node which y of I_2 is bound to, or to the node which y of I_3 is bound to. In the former case, y of I_2 is bound to $\boldsymbol{c}$ since we already know that y of I_5 is bound to $\boldsymbol{c}$; then, y of I_3 must be a yet unread node, indicated by $\underline{y}$. Otherwise, y of I_3 must be bound to $\boldsymbol{c}$, and y of I_2 has already been read, but is not stored in any of the parameters, indicated by $\overset{\bullet}{y}$.

It is clear from Fig. 4 that the parser must read a literal $\mathsf{c}^{\boldsymbol{ac}}$ next if it reduces I_7, or more precisely, a c-literal that is attached to the input nodes bound to parameters $\boldsymbol{a}$ and $\boldsymbol{c}$, respectively. This is so because c^{xy} immediately follows D_2^{xy} in I_5 (indicated by a box). And, if the parser reduces I_6, it must read $\mathsf{b}^{\boldsymbol{ac}}$ next. In fact, the parser must check whether there is a yet unread literal $\mathsf{c}^{\boldsymbol{ac}}$. If it finds one, it must reduce d_2, otherwise it must reduce d_1. To see this, assume there is an unread $\mathsf{c}^{\boldsymbol{ac}}$. This can be read when I_7 is reduced, but never if I_6 is reduced because no further literal would be read after $\mathsf{b}^{\boldsymbol{ac}}$. And if there is no $\mathsf{c}^{\boldsymbol{ac}}$, the parser would get stuck after reducing I_7.

On the other hand, the parser cannot make a reliable decision based on the existence of just literal $\mathsf{b}^{\boldsymbol{ac}}$, because such a literal can be read by the parser if it

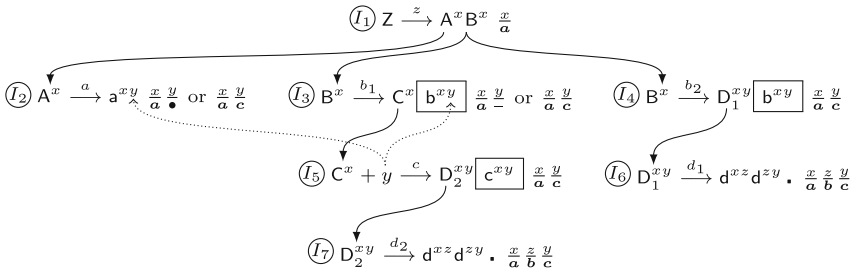


Fig. 4. Lookahead analysis for state q_8^{abc} in Fig. 3.

chooses "reduce d_1", but also if it chooses "reduce d_2". The former is obvious. To see the latter, consider the literal read immediately after c^{ac} when I_7 has been reduced. This must be a literal that corresponds to b^{xy} in I_3, i.e., it is either b^{c-} or b^{ac} according to the possible renamings of I_3. This means, b^{ac} may in fact be read later if I_7 is reduced. Note that this is only possible because the node used as y of I_3 can be the context node used as y of I_5.

6 Realization and Evaluation

PSR parsers for CHR grammars can be generated with the Grappa parser distiller (see footnote 1). Grappa checks whether the CHR grammar has the *free edge choice property*, which is not discussed in this paper. It ensures that, if the parser can end up in a conflict-like situation between shifting alternative edges, the choice will not affect the outcome of the parsing; see [13, Sect. 9] for details.

Parsing of an input graph starts with the identification of unique start nodes, i.e., a place in the graph where parsing has to start. (This is also not considered in this paper; see [11, Sect. 4] for details.) Then the parser uses the concepts outlined in the previous sections to find a shift-reduce parse of the input graph, and finally tries to construct a derivation from this parse so that context nodes are never used before they have been created in the derivation. However, it may turn out in this last step that the input graph is invalid although a parse has been found. This does happen if there are cyclic dependencies between derivation steps that create nodes and those that use such nodes as context nodes [7]. Identification of start nodes and finding a PSR parse are as efficient as in the context-free case. As discussed in [11,13], this means that these steps require linear time in the size of the graph, for all practical purposes. So does creating a derivation from the parse by topological sorting. As a consequence, PSR parsing with respect to CHR grammars runs in linear time in the size of the input graph. However, a more detailed discussion must be omitted here due to lack of space.

We now use the more realistic language of flowcharts to evaluate PSR parsing for CHR grammars. Note that these "unstructured" flowcharts cannot be specified with HR grammars as they have unbounded treewidth.

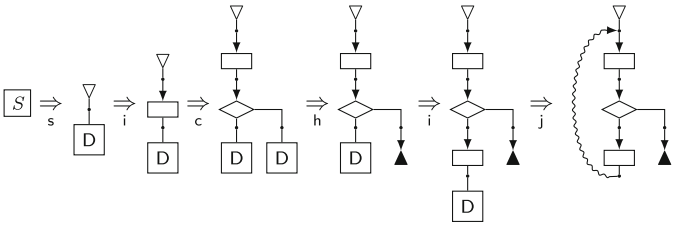


Fig. 5. Derivation of a flowchart

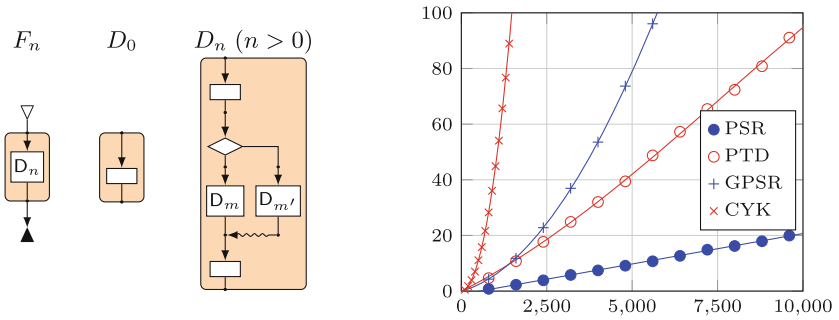


Fig. 6. Definition of flowchart graphs F_n (left) and running time (in milliseconds) of different parsers analyzing F_n for varying numbers of n (right).

Example 3 (Flowcharts). Flowcharts represent the control of low-level imperative programs. In the graphs representing these charts, nodes drawn as small black circles represent program *states*, unary edges labeled with $\triangledown$ and $\blacktriangle$ designate its unique *start state* and its *stop states*, resp., binary edges labeled with $\square$ and $\rightsquigarrow$ represent *instructions* and *jumps*, resp., and ternary edges labeled with $\diamond$ represent *predicates*, which select one of the following branches. (Here, we ignore the texts that usually occur in instructions and predicates.) Flowcharts can be generated by a CHR grammar [7, Ex. 2.6] as follows:

$$S^{\varepsilon} \xrightarrow{\mathsf{s}} \triangledown^{x}\,\mathsf{D}^{x} \qquad \mathsf{D}^{x} \xrightarrow{\mathsf{h}\mid\mathsf{i}\mid\mathsf{c}} \blacktriangle^{x} \mid \square^{xy}\,\mathsf{D}^{y} \mid \diamond^{xyz}\,\mathsf{D}^{y}\,\mathsf{D}^{z} \qquad \mathsf{D}^{x} + y \xrightarrow{\mathsf{j}} \rightsquigarrow^{xy}$$

The context-free rules h, i, and c generate control flow trees of the halt, instruction, and conditional selection, respectively, and the fourth rule, j, which is not context-free, inserts jumps to some program location in the context. Figure 5 shows the diagrams of a derivation of a small flowchart.

Grappa has been used to generate a PSR parser for this grammar. In order to evaluate its efficiency, we have generated three further parsers: a *predictive top-down* (PTD) parser for the same grammar (after extending PTD parsing [10] to CHR grammars), a *Cocke-Younger-Kasami*-style (CYK) parser [20], and finally a *generalized predictive shift-reduce* (GPSR) parser for structured flowcharts [18].

Table 5. Key figures for parsers generated with GRAPPA. "Grammar" columns indicate maximal nonterminal arity (A), nonterminals (N) and terminals (T), context-free (R_{cf}) and contextual (R_{c}) rules, and the maximal length of right-hand sides (L). As the AMR grammar must be transformed into IOC normalform, its key figures are listed under "IOC normalform". "CFA" columns count states (S), items (I), and transitions (Δ) in the CFAs. The last column counts the conflicts in the CFAs.

Example	Grammar						IOC normalform						CFA			Conflicts
	A	N	T	R_{cf}	R_{c}	L	A	N	T	R_{cf}	R_{c}	L	S	I	Δ	
Flowcharts	1	2	5	4	1	3							10	26	30	–
Program graphs	2	12	11	17	4	4							33	80	62	–
AMR	2	9	14	17	11	5	2	13	14	25	11	5	68	211	128	11

GPSR parsing extends PSR parsing to grammars that are not PSR, which is the case for structured flowcharts.

All four parsers have been used to parse flowcharts F_n as defined in Fig. 6 (left), which consist of n predicates, $3n + 1$ instructions, and n jumps. F_n has a subgraph D_n, which, for $n > 0$, contains subgraphs D_m and $D_{m'}$ with $n = m + m' + 1$. Note that the predicates in F_n form a binary tree with n nodes when we ignore instructions. We always choose m and m' such that it is a complete binary tree. Note furthermore that each F_n forms in fact a structured flowchart, which must be built with *jumps* in our flowchart grammar. The GPSR parser has been applied to variations of F_n wherein jumps have been removed, and their source and target nodes have been identified.

Figure 6 shows the running time of each of the four parsers when analyzing graphs F_n with varying values of n. It was measured on a iMac 2017, 4.2 GHz Intel Core i7, Java 1.8.0_202 with standard configuration, and is shown in milliseconds on the y-axis while n is shown on the x-axis. The graphics shows nicely that the PSR parser is linear, and about four times faster than the PTD parser. The GPSR parser is much slower because it deals with conflicts of the CFA by following several parses simultaneously. The CYK parser is the slowest, because it builds up a table of nonterminal edges by dynamic programming.

We have also created parsers for CHR grammars for two additional graph languages: *Program graphs* [7, Ex. 2.7] represent the syntactic structure of object-oriented programming and are used for refactoring. *Abstract Meaning Representations* are widely used in natural language processing to represent sentence meaning. To define the structure of AMRs, CHR grammars are preferable because of their greater capability to cover the set of all AMRs over a given domain. At the same time the grammars become both smaller and simpler. Unfortunately, the example grammar from [14] is not PSR because the CFA has 11 conflicts (see Table 5), but one can employ generalized PSR parsing introduced in [18]. Table 5 lists key figures of the three example languages outlined above.

7 Conclusions

In this paper, we have described how predictive shift-reduce parsing can be extended from HR grammars to CHR grammars. These parsers can be generated with Grappa (see footnote 1), and are as efficient as the context-free version, although they apply to a larger class of languages.

Related Work

Much work has been dedicated to graph parsing. Since space is limited, we mention only results and approaches for HR grammars. Early on, Lautemann [19] identified connectedness conditions which make polynomial parsing of certain HR languages possible, using a generalization of the Cocke-Younger-Kasami (CYK) algorithm for context-free string languages. However, the degree of the polynomial depends on the HR language. Stronger connectedness requirements yield cubic parsing (by a different kind of algorithm), as shown by Vogler and Drewes [6,21]. A CYK parser for general HR grammars (even extended by so-called embedding rules) was implemented by Minas in DiaGen [20]. While this parser takes exponential time in general, it can handle graphs with hundreds of nodes and edges.

The line of work continued in this paper started with the proposal of predictive top-down (PTD) parsers in [10] and continued with the introduction of predictive shift-reduce (PSR) parsers [12,13]. Both apply to suitable classes of HR grammars, while the current paper extends PSR parsers to CHR grammars.

Independently, Chiang et al. [4] improved the parser by Lautemann by making use of tree decompositions, and Gilroy et al. [15] studied parsing for the regular graph grammars by Courcelle [5]. Finally, Drewes et al. [2,3] study a structural condition which enables very efficient uniform parsing.

Future Work

If a grammar has conflicts, a *generalized parser* can pursue all conflicting options in parallel until one of them yields a successful parse. This idea, which has been used for LR string parsing in the first place, has recently been transferred to HR grammars [18]. It turns out that generalized PSR parsing can be further extended to CHR grammars. This way the CHR grammar for abstract meaning representations analyzed in Table 5 can be recognized by a generalized PSR parser generated with Grappa.

So far, the matching of a context node in a host graph depends on the existence of a matching host node. So a contextual rule may causally depend on another rule that generates the required node. Example 1 showed that certain graphs cannot be derived with a rightmost derivation. This complicates the parser, which always constructs rightmost derivations, since it has to check whether causal dependencies have been respected. Since we conjecture that causal dependencies do not really extend the generative power of CHR grammars, we will consider to re-define CHR grammars, without causality. However, for the practical modeling of graph and diagram languages, it should be possible

to express certain conditions that the host node of a context node should fulfill. For instance, one may wish to forbid that the application of a context rule introduces a cycle. This is why the initial version of CHR grammars introduced in [17] features contextual rules with application conditions that can express the existence or absence of certain paths in the host graph. We will investigate the ramifications of application conditions for parsing in the future.

References

1. Banarescu, L., et al.: Abstract meaning representation for sembanking. In: Proceedings of 7th Linguistic Annotation Workshop at ACL 2013 Workshop, pp. 178–186 (2013)
2. Björklund, H., Drewes, F., Ericson, P.: Between a rock and a hard place – uniform parsing for hyperedge replacement DAG grammars. In: Dediu, A.-H., Janoušek, J., Martín-Vide, C., Truthe, B. (eds.) LATA 2016. LNCS, vol. 9618, pp. 521–532. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-30000-9_40
3. Björklund, H., Drewes, F., Ericson, P., Starke, F.: Uniform parsing for hyperedge replacement grammars. UMINF 18.13, Umeå University (2018)
4. Chiang, D., Andreas, J., Bauer, D., Hermann, K.M., Jones, B., Knight, K.: Parsing graphs with hyperedge replacement grammars. In: Proceedings of 51st Annual Meeting of the Association for Computational Linguistic (Vol. 1: Long Papers), pp. 924–932 (2013)
5. Courcelle, B.: The monadic second-order logic of graphs V: on closing the gap between definability and recognizability. Theoret. Comput. Sci. **80**, 153–202 (1991)
6. Drewes, F.: Recognising k-connected hypergraphs in cubic time. Theoret. Comput. Sci. **109**, 83–122 (1993)
7. Drewes, F., Hoffmann, B.: Contextual hyperedge replacement. Acta Informatica **52**, 497–524 (2015)
8. Drewes, F., Hoffmann, B., Janssens, D., Minas, M.: Adaptive star grammars and their languages. Theoret. Comput. Sci. **411**, 3090–3109 (2010)
9. Drewes, F., Hoffmann, B., Minas, M.: Contextual hyperedge replacement. In: Schürr, A., Varró, D., Varró, G. (eds.) AGTIVE 2011. LNCS, vol. 7233, pp. 182–197. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-34176-2_16
10. Drewes, F., Hoffmann, B., Minas, M.: Predictive top-down parsing for hyperedge replacement grammars. In: Parisi-Presicce, F., Westfechtel, B. (eds.) ICGT 2015. LNCS, vol. 9151, pp. 19–34. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-21145-9_2
11. Drewes, F., Hoffmann, B., Minas, M.: Approximating Parikh images for generating deterministic graph parsers. In: Milazzo, P., Varró, D., Wimmer, M. (eds.) STAF 2016. LNCS, vol. 9946, pp. 112–128. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-50230-4_9
12. Drewes, F., Hoffmann, B., Minas, M.: Predictive shift-reduce parsing for hyperedge replacement grammars. In: de Lara, J., Plump, D. (eds.) ICGT 2017. LNCS, vol. 10373, pp. 106–122. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-61470-0_7
13. Drewes, F., Hoffmann, B., Minas, M.: Formalization and correctness of predictive shift-reduce parsers for graph grammars based on hyperedge replacement. J. Logical Algebraic Methods Program. **104**, 303–341 (2019)

14. Drewes, F., Jonsson, A.: Contextual hyperedge replacement grammars for abstract meaning representations. In: 13th International Workshop on Tree-Adjoining Grammar and Related Formalisms (TAG+13), pp. 102–111 (2017)
15. Gilroy, S., Lopez, A., Maneth, S.: Parsing graphs with regular graph grammars. In: Proceedings of 6th Joint Conference on Lexical and Computational Semantics (*SEM 2017), pp. 199–208 (2017)
16. Habel, A.: Hyperedge Replacement: Grammars and Languages. LNCS, vol. 643. Springer, Heidelberg (1992). https://doi.org/10.1007/BFb0013875
17. Hoffmann, B., Minas, M.: Defining models - meta models versus graph grammars. Elect. Commun. EASST **29** (2010). Proceedings of 6th Workshop on Graph Transformation and Visual Modeling Techniques (GT-VMT'10), Paphos, Cyprus
18. Hoffmann, B., Minas, M.: Generalized predictive shift-reduce parsing for hyperedge replacement graph grammars. In: Martín-Vide, C., Okhotin, A., Shapira, D. (eds.) LATA 2019. LNCS, vol. 11417, pp. 233–245. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-13435-8_17
19. Lautemann, C.: The complexity of graph languages generated by hyperedge replacement. Acta Informatica **27**, 399–421 (1990)
20. Minas, M.: Concepts and realization of a diagram editor generator based on hypergraph transformation. Sci. Comput. Program. **44**(2), 157–180 (2002)
21. Vogler, W.: Recognizing edge replacement graph languages in cubic time. In: Ehrig, H., Kreowski, H.-J., Rozenberg, G. (eds.) Graph Grammars 1990. LNCS, vol. 532, pp. 676–687. Springer, Heidelberg (1991). https://doi.org/10.1007/BFb0017421