Rule-Based Top-Down Parsing for Acyclic 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. Contextual hyperedge replacement (CHR) strengthens the generative power of hyperedge replacement (HR) significantly, thus increasing its usefulness for practical modeling. We define top-down parsing for CHR grammars by graph transformation, and prove that it is correct as long as the generation and use of context nodes in productions does not create cyclic dependencies. An efficient predictive version of this algorithm can be obtained as in the case of HR grammars.

Keywords: Graph transformation · Hyperedge replacement · Contextual hyperedge replacement · Parsing · Correctness

1 Introduction

Contextual hyperedge replacement (CHR, [4,5]) strengthens the generative power of hyperedge replacement (HR, [13]) significantly, by productions with context nodes that refer to nodes which are not connected to the edge being replaced. Unfortunately, both HR and CHR grammars can generate NP-complete graph languages [16]. The authors have therefore devised efficient parsers for subclasses of HR and CHR grammars, implementing so-called predictive top-down (PTD) parsing. Although the concepts and implementation of these parsers have been described at depth in [6], their correctness has only recently been formally confirmed, based on the specification of parsers by means of graph transformation rules, and only for HR grammars [8]. Here we extend the parsers and their correctness proof to CHR grammars. It turns out that a CHR grammar Γ can be turned into a HR grammar generating graphs where the nodes that were context nodes in Γ are *borrowed*, i.e., generated like ordinary nodes. From this graph, the one generated by Γ can be obtained by *contraction*, i.e., merging borrowed nodes with other nodes. We show that this is correct provided that the generation and use of context nodes in CHR productions does

F. Gadducci and T. Kehrer (Eds.): ICGT 2021, LNCS 12741, pp. 164–184, 2021.
https://doi.org/10.1007/978-3-030-78946-6_9

not lead to cyclic dependencies. In this paper, we concentrate on describing a non-deterministic parser, and sketch only briefly how this parser can be made predictive (and efficient), since the latter is completely analoguous to the HR case [8].

The remainder of this paper is structured as follows. After recalling some basic concepts of graph transformation (Sect. 2), we define CHR grammars and their corresponding borrowing HR grammars, and we discuss the requirement of acyclicity (Sect. 3). In Sect. 4, we define a top-down parser for acyclic CHR grammars that processes edges in a linear order (corresponding to leftmost derivations in string grammars), and prove it correct. In the conclusions (Sect. 5), we discuss related and future work.

2 Preliminaries

The set of non-negative integers is denoted by $\mathbb{N}$, and $[n]$ denotes $\{1, \ldots, n\}$ for all $n \in \mathbb{N}$. A^* denotes the set of all finite sequences over a set A; the empty sequence is denoted by ε, and the length of a sequence α by $|\alpha|$. As usual, $\to^+$ and $\to^*$ denote the transitive and the transitive reflexive closure of a binary relation $\to$. For a 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 $n \in \mathbb{N}$ and $a_1, \ldots, a_n \in A$.

2.1 Graphs

The graphs considered in this paper have labeled nodes and edges that may have parallel edges carrying the same label. We also generalize edges to hyperedges, which may connect any number of nodes, not just two.

Throughout the paper, let $\mathbb{L}$ be a global set of labels which is partitioned into two infinite subsets $\dot{\mathbb{L}}$ and $\bar{\mathbb{L}}$, and let $\mathit{arity}\colon \bar{\mathbb{L}} \to \mathbb{N}$ be a function that associates an *arity* with every label in $\bar{\mathbb{L}}$. Elements of $\dot{\mathbb{L}}$ and $\bar{\mathbb{L}}$ will be used to label nodes and hyperedges, respectively. A finite set $\Sigma \subseteq \mathbb{L}$ is an *alphabet*, and we let $\dot{\Sigma} = \Sigma \cap \dot{\mathbb{L}}$ and $\bar{\Sigma} = \Sigma \cap \bar{\mathbb{L}}$.

Definition 1 (Hypergraph). A *hypergraph* over an alphabet Σ is a tuple $G = (\dot{G}, \bar{G}, \mathit{att}, \mathit{lab})$, where $\dot{G}$ and $\bar{G}$ are disjoint finite sets of *nodes* and *hyperedges*, respectively, the function $\mathit{att}\colon \bar{G} \to \dot{G}^*$ attaches hyperedges to sequences of nodes, and the function $\mathit{lab}\colon \dot{G} \cup \bar{G} \to \Sigma$ maps $\dot{G}$ to $\dot{\Sigma}$ and $\bar{G}$ to $\bar{\Sigma}$ in such a way that $|\mathit{att}(e)| = \mathit{arity}(\mathit{lab}(e))$ for every edge $e \in \bar{G}$.

For brevity, we omit the prefix "hyper" in the sequel. A node is *isolated* if no edge is attached to it. An edge carrying a label $\sigma \in \Sigma$ is a *σ-edge*, and the *Σ'-edges* of a graph are those labeled with symbols from $\Sigma' \subseteq \bar{\Sigma}$. G° denotes the discrete subgraph of a graph G, which is obtained by removing all edges. We sometimes write $X(G)$ to denote the set $\dot{G}$ of nodes of G, and instead of "$x \in \dot{G}$ or $x \in \bar{G}$", we may write "$x \in G$". We denote the third and fourth component of a graph G by att_G and lab_G. $\mathcal{G}_\Sigma$ denotes the class of graphs over Σ; a graph

$G \in \mathcal{G}_\Sigma$ is called a *handle* (over Σ) if G has a single edge e and each node of G is attached to e. We denote the set of all handles over Σ by $\mathcal{H}_\Sigma$.

Graphs with unlabeled nodes or edges are a special case obtained by letting $\dot{\Sigma}$ contain the "invisible" label $\sqcup$, or letting $\bar{\Sigma}$ contain an invisible label $\sqcup_i$ per arity i. We call a graph *unlabeled* if both nodes and edges are unlabeled. In this case, we omit the labeling in the definition and drawing of the graph.

A set of edges $E \subseteq \bar{G}$ *induces* the subgraph consisting of these edges and their attached nodes. Given graphs $G_1, G_2 \in \mathcal{G}_\Sigma$ with disjoint edge sets, a graph $G = G_1 \cup G_2$ is called the *union* of G_1 and G_2 if G_1 and G_2 are subgraphs of G, $\dot{G} = \dot{G}_1 \cup \dot{G}_2$, and $\bar{G} = \bar{G}_1 \cup \bar{G}_2$. Note that $G_1 \cup G_2$ exists only if common nodes are consistently labeled, i.e., $lab_{G_1}(v) = lab_{G_2}(v)$ for $v \in \dot{G}_1 \cap \dot{G}_2$.

Definition 2 (Graph morphism). Given graphs G and H, a *morphism* $m\colon G \to H$ is a pair $m = (\dot{m}, \bar{m})$ of functions $\dot{m}\colon \dot{G} \to \dot{H}$ and $\bar{m}\colon \bar{G} \to \bar{H}$ that preserve attachments and labels, i.e., $att_H(\bar{m}(v)) = \dot{m}^*(att_G(v))$, $lab_H(\dot{m}(v)) = lab_G(v)$, and $lab_H(\bar{m}(e)) = lab_G(e)$ for all $v \in \dot{G}$ and $e \in \bar{G}$.

The morphism is *injective* or *surjective* if both $\dot{m}$ and $\bar{m}$ are, and a *subgraph inclusion* of G in H if $m(x) = x$ for every $x \in G$; then we write $G \subseteq H$. If m is surjective and injective, it is called an *isomorphism*, and G and H are called *isomorphic*, written as $G \cong H$.

2.2 Graph Transformation

For transforming graphs, we use the classical double-pushout approach of [9], with injective occurrences of rules in graphs.

Definition 3 (Rule). A *graph transformation rule* $r = (P \supseteq I \subseteq R)$ consists of a *pattern graph* P, a *replacement graph* R, and an *interface graph* I included in both P and R. We briefly call r a *rule*, denote it as $r\colon P \circ\!\!\to R$, and refer to its graphs by P_r, R_r, and I_r if they are not explicitly named.

An injective morphism $m\colon P \to G$ into a graph G defines an *occurrence* with respect to r if it satisfies the following *dangling condition*: if the occurrence $m(v)$ of a node $v \in P \setminus I$ is attached to some edge $e \in G$, then e is also in $m(P)$.

A rule r *transforms* a graph G at an occurrence m to a graph H by (1) removing $m(x)$ from G for every $x \in P \setminus I$, to obtain a graph K, and (2) constructing H from the disjoint union of K and R by merging $m(x)$ with every $x \in I$. Then we write $G \Rightarrow_r^m H$, but may omit m if it is irrelevant, and write $G \Rightarrow_{\mathcal{R}} H$ if $\mathcal{R}$ is a set of rules such that $G \Rightarrow_r H$ for some $r \in \mathcal{R}$.

Since the interface of a rule is included in its replacement graph, a transformation step can be constructed in such a way that K is included in H.

2.3 Application Conditions

Sometimes it is necessary to restrict the applicability of a rule by requiring the existence or non-existence of certain subgraphs in the context of its occurrence. Our definition of application conditions is based on [14], but omits nesting as it will not be needed here.

Definition 4 (Conditional rule). For a graph P, the set of *conditions over* P is defined inductively as follows: (i) an inclusion $P \subseteq C$ defines a *basic condition* over P, denoted by $\exists C$. (ii) if c and c' are conditions over P, then $\neg c$, $(c \wedge c')$, and $(c \vee c')$ are conditions over P.

An injective morphism $m\colon P \to G$ *satisfies* a basic condition $\exists C$ if there is an injective morphism $m'\colon C \to G$ whose restriction to P coincides with m. The semantics of Boolean combinations of application conditions is defined in the obvious way; $m \vDash c$ expresses that m satisfies condition c.

A *conditional rule* r' consists of a rule $r = P \circ\!\!\to R$ and a condition c over P, and is denoted as $r'\colon c[\!] P \circ\!\!\to R$. We let $G \Rightarrow^m_{r'} H$ or simply $G \Rightarrow_{r'} H$ if $m \vDash c$ and $G \Rightarrow^m_r H$. Note that rules without conditions can also be seen as conditional rules with the neutral condition $c = \exists P$. For a set $\mathcal{C}$ of conditional rules, $\Rightarrow_{\mathcal{C}} = \bigcup_{r \in \mathcal{C}} \Rightarrow_r$.

Examples of graphs and rules, with and without conditions, will be shown in the following sections.

3 Contextual Hyperedge Replacement

We recall graph grammars based on contextual hyperedge replacement [4,5], which include hyperedge replacement grammars [13] as a special case.

Definition 5 (Contextual hyperedge replacement). Let Σ be an alphabet and $\mathcal{N} \subseteq \bar{\Sigma}$ a set of *nonterminal edge labels* (*nonterminals*, for short). The *terminal edge labels* (*terminals*) are those in $\mathcal{T} = \bar{\Sigma} \setminus \mathcal{N}$. Accordingly, edges with labels in $\mathcal{N}$ and $\mathcal{T}$ are *nonterminal* and *terminal edges*, respectively.

A rule $p\colon P \circ\!\!\to R$ is a *hyperedge replacement production* over Σ (*production*, for short) if the pattern P contains a single edge, which is labeled with a nonterminal, and the interface graph I_p is the discrete subgraph P° consisting of all nodes of P. Isolated nodes in the pattern of p are called *context nodes*; p is called *contextual* if such context nodes exist, and *context-free* otherwise.

A *contextual hyperedge replacement grammar* $\Gamma = \langle \Sigma, \mathcal{N}, \mathcal{P}, Z \rangle$ (*CHR grammar* for short) consists of alphabets Σ and $\mathcal{N} \subseteq \bar{\Sigma}$ as above, a finite set $\mathcal{P}$ of productions over Σ, and a start graph $Z \in \mathcal{G}_\Sigma$. Γ is a (context-free) *hyperedge replacement grammar* (*HR grammar*) if all productions in $\mathcal{P}$ are context-free.

The *language* generated by Γ is given as $\mathcal{L}(\Gamma) = \{G \in \mathcal{G}_{\Sigma \setminus \mathcal{N}} \mid Z \Rightarrow^*_{\mathcal{P}} G\}$.

We use a simple but non-context-free running example because illustrations of parsers would otherwise become too big and complex.

Example 1 (Linked trees). Figure 1 shows our running example, and introduces our conventions for drawing graphs and productions. Nodes are circles, nonterminal edges are rectangular boxes containing the corresponding labels, and terminal edges are shapes like ⧫, ▷, ▶. (In this example, all nodes are unlabeled.) Edges are connected to their attached nodes by lines, called tentacles, which are ordered counter-clockwise around the edge, starting at noon. For productions

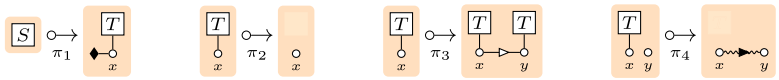


Fig. 1. Productions for linked trees

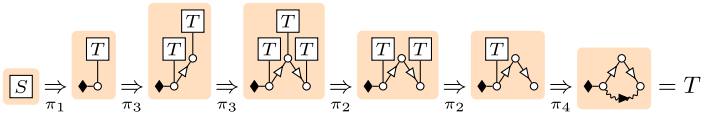


Fig. 2. A derivation of a linked tree T

(and in other rules), we just draw their pattern P and their replacement graph R, and specify the inclusion of the interface nodes by ascribing the same identifier to them in P and R, like x and y in Fig. 1.

Figure 1 defines the productions π_1, π_2, π_3, and π_4 of the CHR grammar Δ. S is the nullary symbol labeling the start graph, and T is a unary nonterminal. A unary ⧫-edge is attached to the root of a tree, binary ▷-edges connect nodes to their children, and binary ▶-edges (drawn with curly tentacles) represent links between nodes. Δ generates trees where every node may have a link to any other node, see Fig. 2.

Assumption 1 (CHR grammar). In the sequel, we assume that CHR grammars $\Gamma = \langle \Sigma, \mathcal{N}, \mathcal{P}, Z \rangle$ satisfy the following conditions:

1. The node sequences attached to nonterminal edges are free of repetitions.
2. The start graph Z consists of a single nonterminal edge of arity 0. This nonterminal symbol does not occur in right-hand sides of productions.
3. Γ is reduced, i.e., every production occurs in a derivation of a graph in $\mathcal{L}(\Gamma)$.
4. $\mathcal{L}(\Gamma)$ does not contain graphs with isolated nodes.

These assumptions are made without loss of generality: in [13, Sect. I 4], it is described how HR grammars can be transformed to satisfy Assumptions 1.1–1.2; these results can directly be lifted to CHR grammars. How to attain Assumption 1.3 for CHR grammars is shown in [4, Sect. 3.4]. Assumption 1.4 is made to simplify the technicalities of parsing. To ensure it, unary *virtual edges* can be attached to isolated nodes in the productions and in the graphs generated by the grammar. In Example 1, e.g., the ⧫-edge avoids that the grammar generates a single isolated node.

We now recall the well-known notion of derivation trees, which reflect the context-freeness of HR grammars [3, Definition 3.3]. Here we use a slightly modified version that represents derivations of concrete graphs:

Definition 6 (Derivation tree). Let $\Gamma = \langle \Sigma, \mathcal{N}, \mathcal{P}, Z \rangle$ be a HR grammar. The set $\mathbb{T}_\Gamma$ of *derivation trees* over Γ and the mappings *root*: $\mathbb{T}_\Gamma \to \mathcal{H}_\Sigma$ as well as *result*: $\mathbb{T}_\Gamma \to \mathcal{G}_\Sigma$ are inductively defined as follows:

- Each handle G is in $\mathbb{T}_\Gamma$, and $root(G) = result(G) = G$.
- A triple $t = \langle G, p, c\rangle$ consisting of a nonterminal handle G, a production $p \in \mathcal{P}$, and a sequence $c = t_1 t_2 \cdots t_n \in \mathbb{T}_\Gamma^*$ is in $\mathbb{T}_\Gamma$ if the union graphs $G' = G^\circ \cup \bigcup_{i=1}^n root(t_i)$ and $G'' = G^\circ \cup \bigcup_{i=1}^n result(t_i)$ exist, $G \Rightarrow_p G'$, and $X(result(t_i)) \cap X(result(t_j)) = X(root(t_i)) \cap X(root(t_j))$ for all distinct $i, j \in [n]$. Furthermore, we let $root(t) = G$ and $result(t) = G''$.

An example of a derivation and its derivation tree is shown in Fig. 5.

The ordering of derivation trees in $c = t_1 t_2 \cdots t_n$ within a derivation tree $t = \langle G, p, c\rangle$ will become relevant when edges in right-hand sides of rules are ordered, which will be the case in Sect. 4. Then we require that $root(t_1)\, root(t_2) \cdots root(t_n)$ corresponds to the edge ordering in production p.

Let t, t' be any derivation trees. We call t' a child tree of t, written $t' \prec t$, if $t = \langle G, p, t_1 t_2 \cdots t_n\rangle$ and $t' = t_i$ for some i, and we call t' a subtree of t if $t' = t$ or $t = \langle G, p, t_1 t_2 \cdots t_n\rangle$ and t' is a subtree of t_i for some i. A derivation tree t *introduces* a node u (at its root) if $t = \langle G, p, t_1 t_2 \cdots t_n\rangle$ and $u \in X(root(t_i)) \setminus \dot{G}$ for some i. The set of all these nodes is denoted by $intro(t)$. We define the pre-order traversal $pre(t) \in \mathbb{T}_\Gamma^*$ of a derivation tree t recursively by $pre(t) = t$ if $t \in \mathcal{H}_\Sigma$ and $pre(t) = t\, pre(t_1)\, pre(t_2) \cdots pre(t_n)$ if $t = \langle G, p, t_1 t_2 \cdots t_n\rangle$.

The following theorem is equivalent to Theorem 3.4 in [3]:

Theorem 1. *Let $\Gamma = \langle \Sigma, \mathcal{N}, \mathcal{P}, Z\rangle$ be a HR grammar, $H \in \mathcal{H}_\Sigma$ a handle and $G \in \mathcal{G}_\Sigma$ a graph. There is a derivation tree $t \in \mathbb{T}_\Gamma$ with $root(t) = H$ and $result(t) = G$ iff $H \Rightarrow_\mathcal{P}^* G$.*

Note that derivation trees are defined only for HR grammars as in the contextual case, any properly labeled node can be used as a context node as long as it has been created earlier in a derivation. This fact produces dependencies between derivation steps which do not exist in HR derivations. It will turn out in the following that there is a close relationship between a CHR grammar Γ and its so-called borrowing HR grammar $\hat{\Gamma}$: every graph $H \in \mathcal{L}(\Gamma)$ is a "contraction" of a graph $G \in \mathcal{L}(\hat{\Gamma})$. Moreover, the converse is also true as long as Γ is acyclic, a notion to be defined later.

In the following, we assume that $\mathcal{T}$ contains two auxiliary edge labels that are not used elsewhere in Γ: edges carrying the unary label $\circledcirc$ will mark borrowed nodes, and binary edges labeled $\neq$ will connect borrowed nodes to all nodes that should be kept separate to them, i.e., not be contracted later on.

Definition 7 (Borrowing grammar). Let $\Gamma = \langle \Sigma, \mathcal{N}, \mathcal{P}, Z\rangle$ be a CHR grammar. For $(p\colon P \circ\!\!\to R) \in \mathcal{P}$, its *borrowing production* $\hat{p}\colon \hat{P} \to \hat{R}$ is obtained by (a) removing every context node from $\hat{P}$ and $I_{\hat{p}}$ and (b) constructing $\hat{R}$ from R as follows: for every context node v of p, attach a new $\circledcirc$-edge to v, and add $\neq$-edges from v to every other node with the label $lab_P(v)$. The *borrowing grammar* $\hat{\Gamma} = \langle \Sigma, \mathcal{N}, \hat{\mathcal{P}}, Z\rangle$ of Γ is given by $\hat{\mathcal{P}} = \{\hat{p} \mid p \in \mathcal{P}\}$.

Note that $\hat{p} = p$ if p is context-free.

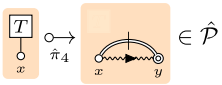


Fig. 3. Borrowing link production

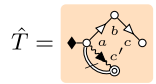
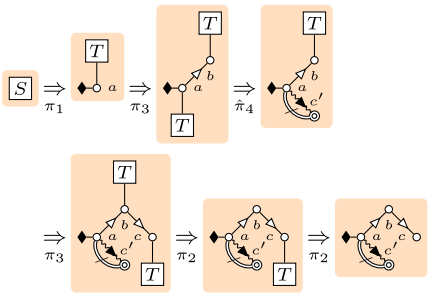


Fig. 4. A linked tree with a detached link



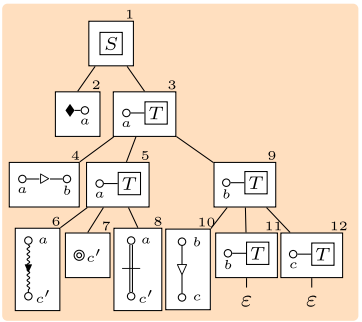


Fig. 5. A derivation of $\hat{T}$ (cf. Fig. 4) and its derivation tree

Definition 8 (Contraction). For a graph G let

$$\dot{G}_{\circledcirc} = \{v \in \dot{G} \mid v = att_G(e) \text{ for a } \circledcirc\text{-edge } e \in \bar{G}\} \text{ and}$$
$$\neq_G = \{(u, v) \in \dot{G} \times \dot{G} \mid uv = att_G(e) \text{ for a } \neq\text{-edge } e \in \bar{G}\}.$$

A morphism $\mu\colon G \to H$ is called a *joining morphism* for G if $\dot{H} = \dot{G} \setminus \dot{G}_{\circledcirc}$, $\bar{H} = \bar{G}$, $\bar{\mu}$ and the restriction of $\dot{\mu}$ to $\dot{G} \setminus \dot{G}_{\circledcirc}$ are inclusions, and $(v, \dot{\mu}(v)) \notin \neq_G$ for every $v \in \dot{G}_{\circledcirc}$. The graph *core*$(H)$ obtained from H by removing all edges with labels $\circledcirc$ and $\neq$ is called the *μ-contraction* of G or just *a contraction* of G.

Example 2 (Trees with detached links). Figure 3 shows the borrowing link production $\hat{\pi}_4$, and Fig. 4 shows a tree with a detached link that can be generated by a borrowing grammar $\hat{\Delta}$ with productions $\{\pi_1, \pi_2, \pi_3, \hat{\pi}_4\}$. (Here, $\circledcirc$-edges are depicted by drawing the attached node accordingly.) Obviously, the linked tree $\hat{T}$ in Fig. 4 can be generated by the derivation in Fig. 5, and contracted to the linked tree T generated in Fig. 2. Figure 5 also shows the derivation tree of $\hat{T}$. Only the root handles of trees and subtrees are shown, and ε is used to indicate an empty sequence of child trees, thus distinguishing this case from the one where a subtree is a single handle. The numbers on top of the handles illustrate the ordering of the corresponding subtrees in a pre-order traversal of the entire tree.

Definition 9 (Borrowing version of a derivation). Let $\Gamma = \langle \Sigma, \mathcal{N}, \mathcal{P}, Z \rangle$ be a CHR grammar and $\hat{\Gamma}$ its borrowing HR grammar. A derivation

$$Z \Rightarrow^{\hat{m}_1}_{\hat{p}_1} H_1 \Rightarrow^{\hat{m}_2}_{\hat{p}_2} H_2 \Rightarrow^{\hat{m}_3}_{\hat{p}_3} \cdots \Rightarrow^{\hat{m}_n}_{\hat{p}_n} H_n$$

in $\hat{\Gamma}$ is a *borrowing version* of a derivation

$$Z \Rightarrow^{m_1}_{p_1} G_1 \Rightarrow^{m_2}_{p_2} G_2 \Rightarrow^{m_3}_{p_3} \cdots \Rightarrow^{m_n}_{p_n} G_n$$

in Γ if the following hold, for $i = 1, 2, \ldots, n$ and $p_i \colon P \hookrightarrow R$:

1. $\hat{p}_i$ is the borrowing production of p_i,
2. if $\bar{P} = \{e\}$ then $\hat{m}_i(e) = m_i(e)$, and
3. for every $x \in \bar{R} \cup (\dot{R} \setminus \dot{P})$, the images of x in G_i and H_i are the same.

By a straightforward induction, it follows that every derivation in Γ has a borrowing version in $\hat{\Gamma}$, and G_i is the μ_i-contraction of H_i for $i \in [n]$, where the joining morphism μ_i is uniquely determined by $\bar{\mu}_i(e) = e$ for all $e \in \bar{H}_i$.

Theorem 2 will show that the converse is also true, i.e., that every contraction of a graph in $\mathcal{L}(\hat{\Gamma})$ can also be derived in Γ, provided that Γ is *acyclic* (Definition 10). Informally, Γ is cyclic if there is a derivation of a graph G in $\hat{\Gamma}$ and a contraction H of G so that there is a cyclic dependency between derivation steps that create nodes and derivation steps that use them as context nodes. These cyclic dependencies then result in derivations of graphs G in $\hat{\Gamma}$ having a contraction H that cannot be derived in Γ because there is no reordering of the derivation steps that yields a valid derivation in Γ.

Cyclic dependencies caused by a joining morphism μ for a derivation tree $t \in \mathbb{T}_{\hat{\Gamma}}$ are formalized using the relation $\sqsubset_\mu$ on subtrees of t. Informally, $t' \sqsubset_\mu t''$ means that t' describes a derivation step (the topmost one that transforms the root handle of t'), which creates a node used as a context node in the corresponding topmost contextual derivation step described by t''. This, together with the definition of acyclic CHR grammars, is defined next.

Definition 10 (Acyclic CHR grammar). Let Γ be a CHR grammar.

For any two subtrees t', t'' of a derivation tree $t \in \mathbb{T}_{\hat{\Gamma}}$, we let $t' \sqsubset_\mu t''$ iff there is a node $u \in \mathit{intro}(t'')$ so that $\dot{\mu}(u) \neq u$ and $\dot{\mu}(u) \in \mathit{intro}(t')$.

Γ is *acyclic* if $(\succ \cup \sqsubset_\mu)^+$ is irreflexive for all derivation trees $t \in \mathbb{T}_{\hat{\Gamma}}$ over $\hat{\Gamma}$ and all joining morphisms μ for $\mathit{result}(t)$. Otherwise, Γ is *cyclic*.

Theorem 2. *Let Γ be a CHR grammar and $\hat{\Gamma}$ its borrowing HR grammar. For every graph $H \in \mathcal{L}(\Gamma)$, there is a graph $G \in \mathcal{L}(\hat{\Gamma})$ so that H is a contraction of G. Moreover, every contraction of a graph in $\mathcal{L}(\hat{\Gamma})$ is in $\mathcal{L}(\Gamma)$ if Γ is acyclic.*

Proof. Let Γ be a CHR grammar and $\hat{\Gamma}$ its borrowing HR grammar. The first part of the theorem is a direct consequence of the observation made after Definition 9. To prove the second part of the theorem, let Γ be acyclic and H the μ-contraction of a graph $G \in \mathcal{L}(\hat{\Gamma})$, i.e., $G = \mathit{result}(t)$ for some derivation tree

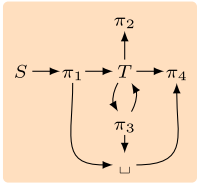


Fig. 6. Grammar graph of Δ

t over $\hat{\Gamma}$. Since $(\succ \cup \sqsubset_\mu)^+$ is irreflexive, one can order the subtrees of t topologically according to $\succ \cup \sqsubset_\mu$, obtaining a sequence $t_1, \ldots, t_n$ of derivation trees so that $1 \leq i < j \leq n$ implies that neither $t_i \prec t_j$ nor $t_j \sqsubset_\mu t_i$. This sequence (when considering just those subtrees that are of the form $\langle G, p, c\rangle$) determines a derivation of G in $\hat{\Gamma}$, and every node u with $\dot{\mu}(u) \neq u$ is created in a later derivation step than $\dot{\mu}(u)$. The derivation is therefore a borrowing version of a derivation of H in Γ. In particular, no node is used as a context node before it has been created. □

Definition 10 does not provide effective means to check whether a CHR grammar is acyclic. However, we can make use of the fact that joining morphisms may only map a borrowed node to a node with the same label. A CHR grammar cannot be cyclic if we can make sure that no nodes with any label can ever be part of a cyclic dependency. The grammar graph of a CHR grammar allows for such reasoning. It describes which rules can create nodes with which labels or use them as context nodes, and it relates nonterminal labels with productions in the sense that productions are applied to nonterminal edges, producing new nonterminal edges. (The creation of terminal edges is irrelevant here.)

Definition 11 (Grammar graph). The *grammar graph* of a CHR grammar $\Gamma = \langle \Sigma, \mathcal{N}, \mathcal{P}, Z\rangle$ is the unlabeled graph G_Γ such that $\dot{G}_\Gamma = \mathcal{P} \cup \mathcal{N} \cup \dot{\Sigma}$ and for each rule $p\colon P \hookrightarrow R$, say with $\bar{P} = \{l\}$, $\bar{G}$ contains binary edges from the node $lab(l)$ to p and from p to each node in $\{lab(x) \mid x \in R\} \setminus \mathcal{T}$, as well as an edge from $lab(u)$ to p for every context node u of p.

Example 3 (Acyclicity of the linked tree grammar). Figure 6 shows the grammar graph for Δ. (Recall that "$\sqcup$" is the otherwise omitted invisible node label.) Lemma 1 will reveal that Δ has only harmless dependencies since the cycle $T \to \pi_3 \to T$ in its grammar graph does not contain $\sqcup$.

The next lemma provides a sufficient criterion to check whether a CHR grammar is acyclic, and therefore, whether Theorem 2 can be exploited:

Lemma 1. *The grammar graph of every cyclic CHR grammar has a cycle that contains a node in $\dot{\Sigma}$.*

Proof. Let $\Gamma = \langle \Sigma, \mathcal{N}, \mathcal{P}, Z\rangle$ be a cyclic CHR grammar, $\hat{\Gamma}$ its borrowing HR grammar, G_Γ the grammar graph of Γ, $t \in \mathbb{T}_{\hat{\Gamma}}$ a derivation tree over $\hat{\Gamma}$, and μ

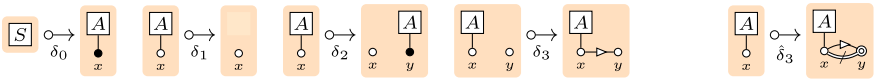


Fig. 7. Productions for generating dags, with borrowing production $\hat{\delta}_3$

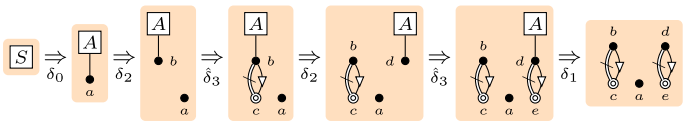


Fig. 8. A derivation with $\hat{\Lambda}$

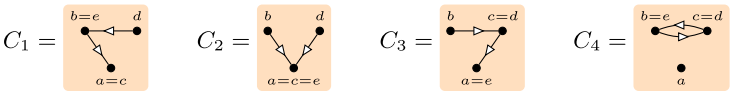


Fig. 9. The four contractions of the graph derived in Fig. 8

a joining morphism for $\mathit{result}(t)$ so that there is a sequence $t_1, \ldots, t_n$ of $n > 2$ subtrees of t so that $t_1 = t_n$ and $t_i \,(\succ \cup \sqsubset_\mu)\, t_{i+1}$ for $1 \leq i < n$. By the definition of $\succ$ and $\sqsubset_\mu$, t_i is of the form $\langle G_i, p_i, c_i\rangle$ for every i. If $t_i \succ t_{i+1}$ and $\bar{G}_{i+1} = \{e\}$, G_Γ contains edges from p_i to $\mathit{lab}(e)$ and from $\mathit{lab}(e)$ to p_{i+1}. If $t_i \sqsubset_\mu t_{i+1}$, there is a node $u \in \mathit{intro}(t_{i+1})$ so that $\dot{\mu}(u) \neq u$ and $\dot{\mu}(u) \in \mathit{intro}(t_i)$, which means that u is the image of a context node of p_{i+1} and the image of a created node of p_i, i.e., G_Γ contains edges from p_i to $\mathit{lab}(u)$ and from $\mathit{lab}(u)$ to p_{i+1}. Hence, G_Γ contains a cycle. Moreover, there must be at least one i so that $t_i \sqsubset_\mu t_{i+1}$ because $\succ$ is irreflexive, proving that the cycle contains a node in $\dot{\Sigma}$. □

The following example demonstrates that directed acyclic graphs (dags) can be generated by a CHR grammar. However, this gramnmar is cyclic and thus has a grammar graph with a cycle that contains a node in $\dot{\Sigma}$.

Example 4 (CHR grammar for dags). Consider nonterminals S (of arity 0), A (of arity 1), and terminals $\bullet$ (of arity 1) and $\triangleright$ (of arity 2). Figure 7 shows productions δ_0 to δ_3 over these symbols, where nodes attached to a $\bullet$-edge are just drawn as $\bullet$. (These edges are introduced to meet Assumption 1.4 that generated graphs do not contain isolated nodes.) The CHR grammar Λ with these productions and with an S-edge as a start graph generates all unlabeled dags with at least one node. In its borrowing HR grammar $\hat{\Lambda}$, the contextual production δ_3 is replaced by the context-free production $\hat{\delta}_3$ shown on the right of Fig. 7.

A derivation with $\hat{\Lambda}$ is shown in Fig. 8; the resulting terminal graph can be contracted in four possible ways, to the graphs $C_1, \ldots, C_4$ shown in Fig. 9. The contraction C_4 is cyclic, and is the only one that cannot be generated with the productions of the CHR grammar Λ.

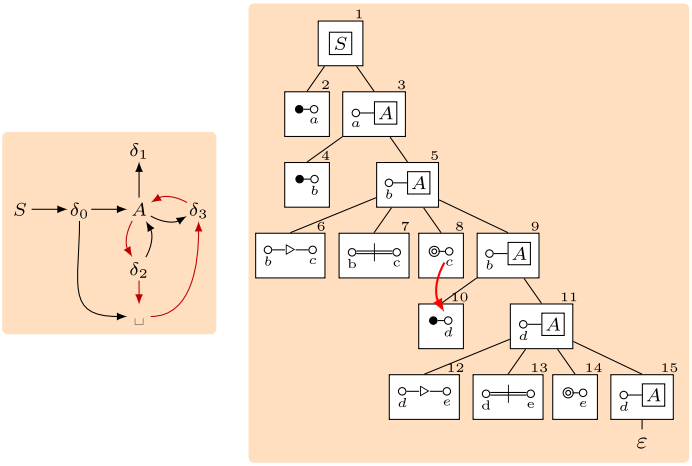


Fig. 10. Grammar graph (left) and derivation tree (right) of grammar $\hat{\Lambda}$

Figure 10 shows the cyclic grammar graph for Λ on the left, and the derivation graph of the derivation in Fig. 8. Here the illegal contraction leading to the cyclic graph C_4 is indicated by the thick bent arrow between nodes 8 and 10.

The fact that the contextual production δ_3 can be applied only after its context node has been generated with production δ_0 or δ_2 makes sure that no cyclic graphs can be generated. This indicates that cyclic CHR grammars are strictly more powerful than acyclic ones.

While the absence of a cycle containing a node in $\dot{\Sigma}$ in the grammar graph implies that a CHR grammar is acyclic, the converse is unfortunately not true: there are acyclic grammars whose grammar graphs have such cycles. A criterion to characterize acyclic CHR grammars needs to be determined in future work.

4 Top-Down Parsing for Acyclic CHR Grammars

We define top-down parsers for acyclic CHR grammars by parsing the corresponding borrowing HR grammars with stack automata that take the necessary merging of borrowed nodes with other nodes into account. The stack automata perform transitions of states that are called configurations. Configurations are represented as graphs, and transitions are described by graph transformation rules. This definition is more precise than the original definition of top-down parsing in [6], but avoids technical complications occurring in [7], where graphs are represented textually as sequences of literals transformed by parser actions. In particular, no explicit substitution and renaming operations on node identifiers are required. Further, this approach simplifies handling the borrowing and merging required for CHR parsing.

For ease of presentation, we consider an arbitrary but fixed CHR grammar $\Gamma = \langle \Sigma, \mathcal{N}, \mathcal{P}, Z \rangle$ throughout the rest of this paper, and let $\hat{\Gamma} = \langle \Sigma, \mathcal{N}, \hat{\mathcal{P}}, Z \rangle$ be its borrowing HR grammar according to Definition 7.

Top-down parsers attempt to construct a derivation of a graph that matches a given input graph. Our top-down parser processes the edges of an input graph G in a nondeterministically chosen linear order when it attempts to construct a derivation for G. Technically, this order is represented by equipping derived edges with two additional tentacles by which they are connected to nodes labeled with a fresh symbol $\bullet$ to form a linear thread.

The definition of the top-down parser for borrowing HR grammars differs from that of HR grammars as follows: in [8], the binding of stack to input nodes and edges was represented by identifying them; here we connect matched nodes with binding edges, and remove matched edges. This results in a more elegant construction, yields more intuitive parses, and simplifies the correctness proof.

Definition 12 (Threaded graph). The *threaded alphabet* $\Sigma^\bullet$ of Σ is given by $\dot{\Sigma}^\bullet = \dot{\Sigma} \cup \{\bullet\}$ and $\bar{\Sigma}^\bullet = \mathcal{T} \cup \{\ell^\bullet \mid \ell \in \bar{\Sigma} \setminus \{\circledcirc, \neq\}\}$ with $arity(\ell^\bullet) = arity(\ell) + 2$; $\mathcal{N}^\bullet = \{\ell^\bullet \mid \ell \in \mathcal{N}\}$ denotes the set of *threaded nonterminals*.

Let $G \in \mathcal{G}_{\Sigma^\bullet}$. A node $v \in \dot{G}$ is a *thread node* if $lab_G(v) = \bullet$, and a *kernel node* otherwise. $\lceil \dot{G} \rfloor$ and $\dot{G}^\bullet$ denote the sets of all kernel nodes and thread nodes of G, respectively. An edge is *threaded* if its label is of the form $\ell^\bullet$, and *unthreaded* otherwise.

A graph $G \in \mathcal{G}_{\Sigma^\bullet}$ is *threaded* if all of its edges except the $\circledcirc$- and $\neq$-edges are threaded and the following additional conditions hold:

1. For every threaded edge $e \in \bar{G}$ with $att_G(e) = u_1 \dots u_k u_{k+1} u_{k+2}$, the nodes $u_1, \dots, u_k$ are kernel nodes of G and u_{k+1}, u_{k+2} are thread nodes of G.
2. $\dot{G}^\bullet$ can be ordered as $\dot{G}^\bullet = \{v_0, \dots, v_n\}$ for some $n \in \mathbb{N}$ such that, for every $i \in [n]$, $\bar{G}$ contains exactly one threaded edge e so that $att_G(e)$ ends in $v_{i-1} v_i$, and there are no further threaded edges than these.

We call v_0 the *first* and v_n the *last* thread node of G.

The *kernel graph* of G is the graph $\lceil G \rfloor \in \mathcal{G}_\Sigma$ obtained by replacing every edge label $\ell^\bullet$ by ℓ, and removing the thread nodes and their attached tentacles. In this case, G is a *threading* of $\lceil G \rfloor$. Note that a threading of a graph in $\mathcal{G}_\Sigma$ is uniquely determined by an ordering of its non-$\{\circledcirc, \neq\}$-edges and vice versa.

We use a set $\Sigma_{\text{aux}} = \{\otimes, \mathsf{bind}\}$ of *auxiliary edge labels*, disjoint with Σ, where $\otimes$ is unary and labels edges that mark *unmatched nodes* in a configuration, whereas bind is binary, and represents the *binding* of a stack node to a node in the input.

Definition 13 (Configuration). A graph C over $\Sigma^\bullet \cup \mathcal{T} \cup \Sigma_{\text{aux}}$ is a *configuration* if

- only $\Sigma^\bullet$-edges are attached to thread nodes and
- the subgraph $stack(C)$ of C induced by its $\Sigma^\bullet$-edges and thread nodes is a threaded graph.

The edge attached to the first thread node is said to be *topmost* on the stack. The *input* of C is the subgraph *input*(C) induced by the (unthreaded) $\mathcal{T}$-edges. C is called

- *initial* if *stack*(C) is a handle of $Z^\bullet$, $\otimes$-edges are attached to all other nodes of C, and there are no bind-edges in C;
- *accepting* if *stack*(C) is an isolated node, all further nodes are attached to bind-edges, and all other edges in C are labeled with $\circledcirc$ or $\neq$.

Definition 14 (Top-down parser). Let $\mathcal{R}$ be a set of conditional rules. A derivation $C \Rightarrow^*_{\mathcal{R}} C'$ is a *parse* if C is an initial configuration. A parse $C \Rightarrow^*_{\mathcal{R}} C'$ is *successful* if C' is an accepting configuration. $\mathcal{R}$ is a *top-down parser* for Γ if, for each initial configuration C, *input*$(C) \in \mathcal{L}(\Gamma)$ if and only if there is a successful parse $C \Rightarrow^*_{\mathcal{R}} C'$.

We define two kinds of top-down parsing rules operating on the topmost edge e on the stack:

- If e is nonterminal, expand pops e from the stack, and pushes the right-hand side of a production for e onto the stack;
- If e is terminal, e is matched with a corresponding unthreaded edge e' in the input; then e is popped from the stack, and e' is removed.

For a match rule to match e to e', we must have $lab(e) = lab(e')^\bullet$, and each pair u, v of corresponding attached nodes must either already be bound to each other by a bind-edge, or there must be $\otimes$-edges attached to both, which indicates that they are still unprocessed, or u must have both a $\otimes$-edge and a $\circledcirc$-edge attached to it. The latter covers the case where u is a still unprocessed borrowed node which can thus be bound to a node already bound earlier. To make sure that the borrowed node is not bound to a node of the same right-hand side, an application condition checking for the absence of a $\neq$-edge is needed. Also, the condition forbids that borrowed nodes are treated as ordinary ones.

Definition 15 (Expand and match rules). For every borrowing production $(p\colon P \hookleftarrow R) \in \hat{\mathcal{P}}$, the *expand rule* $t_p\colon P' \hookleftarrow R'$ is given as follows:

- P' and R' are threadings of P and R, respectively, where every node introduced by p has a $\otimes$-edge attached in R' (and no others have).
- The interface I_{t_p} is I_p plus the last thread node of P'.

A conditional rule $t\colon c[\![P \hookleftarrow R$ with configurations P and R is a *match rule* for a terminal symbol $a \in \mathcal{T} \setminus \{\circledcirc, \neq\}$ if the following holds:

- *stack*(P) is a handle of $a^\bullet$, say with attached nodes $u_1 \cdots u_k u_{k+1} u_{k+2}$.
- *input*(P) is a handle of a, say with attached nodes $v_1 \cdots v_k$.
- For every pair (u_i, v_i), $i \in [k]$, precisely one of the following conditions holds:
  - P contains a bind-edge with attachment $u_i v_i$,
  - a $\otimes$-edge is attached to both u_i and v_i, or
  - both a $\otimes$-edge and a $\circledcirc$-edge is attached to u_i.

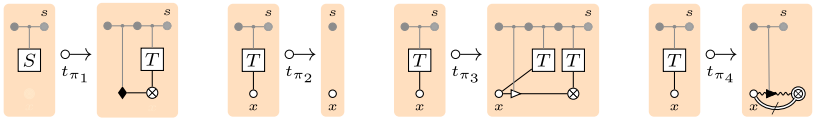


Fig. 11. Expand rules of the top-down parser for linked trees

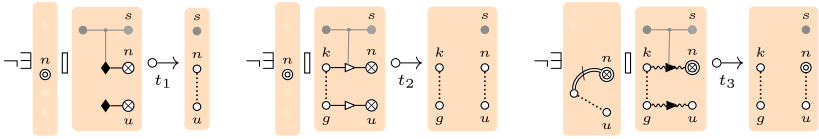


Fig. 12. Match rules for the top-down parser for linked trees

- I_t is P without the a-edge, the $a^\bullet$-edge, and the first thread node.
- R consists of the nodes $u_1, \ldots, u_{k+2}, v_1, \ldots, v_k$, the last thread node of $\mathit{stack}(P)$, and a bind-edge from u_i to v_i for every $i \in [k]$.
- For every $i \in [k]$, the application condition c requires the following: Let $m\colon P \to G$ be the occurrence. If $u_i \in \dot{P}_\circledcirc$, then there is no $z \in \dot{G}$ with $(m(u_i), z) \in \neq_G$ such that z has a bind-edge to $m(v_i)$. If $u_i \notin \dot{P}_\circledcirc$, then c requires that $m(u_i) \notin \dot{G}_\circledcirc$.

We let $\mathcal{R}_{\Gamma^\bullet}$ denote the set of all expand and match rules of $\Gamma^\bullet$.

Example 5 (Top-down parser for linked trees). For the expand rules of the top-down parser for linked trees with detached links (Fig. 11) we have threaded the right-hand sides so that terminal edges come first, and nonterminals attached to the source node of a terminal edge next. Nodes attached to a $\otimes$-edge or a $\circledcirc$-edge are drawn as $\otimes$ and $\circledcirc$ respectively, rather than drawing the edge separately. A node attached to both a $\otimes$-edge and a $\circledcirc$-edge, is drawn as $\circledcirc$. We draw bind-edges as dotted lines (in downward direction), and $\neq$-edges as double lines with a vertical bar.

In general, the terminal symbols of Δ lead to four match rules for $\blacklozenge$, and nine rules for $\triangleright$ and $\blacktriangleright$ each. However, inspection of Δ reveals that just three match rules are needed by the parser, which are shown in Fig. 12: A $\blacklozenge$-edge is matched when its attached node is unbound, and $\triangleright$-edges and $\blacktriangleright$-edges are matched when their first attached node is already bound. The application conditions for the match rules t_1 and t_2 are actually not needed; analysis of Δ reveals that the unbound nodes of these edges will never be attached to $\circledcirc$-edges.

Figure 13 shows a successful parse of the linked tree T derived in Fig. 2 with these rules. We have left shades of the matched edges in the configurations to illustrate how the parse constructs the derivation with the borrowing HR grammar $\hat{\Gamma}$ in Fig. 5, which corresponds to the derivation with Γ in Fig. 2.

Note that match rules consume the thread and the matched edges. Expand rules do not modify the input, but just replace the first nonterminal on the thread by the replacement graph of a threaded production for this nonterminal.

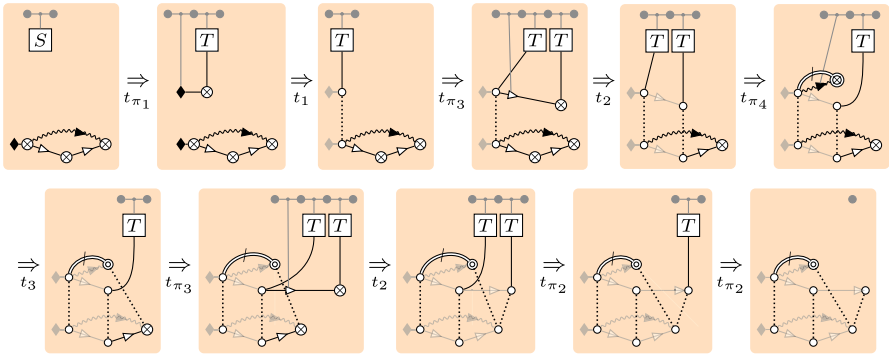


Fig. 13. A parse of the linked tree T generated in Fig. 2

In the following, we prove formally that $\mathcal{R}_{\Gamma^\bullet}$ is indeed a top-down parser for Γ, provided that Γ is acyclic.

Fact 1 (Invariants of configurations)

1. The bind-edges define an irreflexive partial function
$$\mapsto = \{(x, y) \mid e \in \bar{C}, lab_C(e) = \mathsf{bind}, att_C(e) = xy\},$$
between non-thread nodes, called *binding*, such that $x \mapsto y$ implies that (1) x is not in $input(C)$, and (2) y is not in $stack(C)$.
2. No node is attached to several $\otimes$-edges.
3. A kernel node of a threaded edge in $stack(C)$ is attached to a $\otimes$-edge if and only if it is not the source of a bind-edge.
4. Every node of $input(C)$ that is not the target of a bind-edge is attached to a $\otimes$-edge. (The converse is not true because a $\otimes$-edge may be attached to the target of a bind-edge if the source of that bind-edge is in $\dot{C}_\otimes$; see Fig. 13.)

We now consider the first direction of the correctness of the parser.

Lemma 2. *Every graph $G \in \mathcal{L}(\Gamma)$ has a successful parse $C \Rightarrow^*_{\mathcal{R}_{\Gamma^\bullet}} C'$, where C is the initial configuration with $input(C) = G$.*

The proof relies on the following construction to obtain a successful parse.

Construction 1. Let $Z \Rightarrow_{p_1} G_1 \Rightarrow_{p_2} \cdots \Rightarrow_{p_n} G_n$ with $p_i \in \mathcal{P}$ for $i \in [n]$ be a derivation for $G = G_n$. Consider a borrowing version $Z \Rightarrow_{\hat{p}_1} H_1 \Rightarrow_{\hat{p}_2} \cdots \Rightarrow_{\hat{p}_n} H_n$ of the derivation, where G is the μ-contraction of $G' = H_n$. Let $t \in \mathbb{T}_{\hat{\Gamma}}$ be the derivation tree of $Z \Rightarrow_{\hat{p}_1} H_1 \Rightarrow_{\hat{p}_2} \cdots \Rightarrow_{\hat{p}_n} H_n$, where the ordering of subtrees corresponds to the edge ordering in rules of Γ.[1] Let $t_1 t_2 \cdots t_n$ be the sequence

[1] Note that, while t is a derivation tree over the borrowing HR grammar $\hat{\Gamma}$, the ordering of right-hand sides of productions in $\hat{\Gamma}$ ignores edges that are labeled with $\neq$ and $\circledcirc$, so that it is the same as in Γ and provides the required edge ordering.

of subtrees obtained from the pre-order traversal $pre(t)$ of t by keeping only the trees with terminal and nonterminal root handles.

For $k \in \{0, \ldots, n\}$, construct the sequence $T_k \in \mathcal{H}_\Sigma^*$ from $t_1 t_2 \cdots t_k$ by keeping only the trees that are terminal handles. Let N_k be the set of nodes occurring in T_k. By definition, $N_k \subseteq \dot{G}'$. As the edges of terminal handles in T_k are exactly those in G', and $\bar{G} = \bar{G}'$, each handle in T_k identifies a unique edge in G.

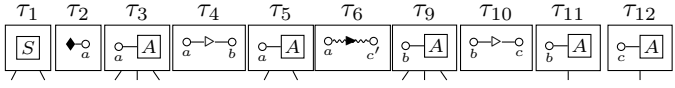
Let $S_k \in \mathcal{H}_\Sigma^*$ be the sequence obtained from $t_{k+1} t_{k+2} \cdots t_n$ by removing each tree t_i that is a subtree of a tree t_j where $k < j < i$, and replacing each remaining tree t_i by its root handle $root(t_i)$. Moreover, let L_k be the subset of those handles in $\{root(t') \mid t' \prec t_i \text{ for some } i < k\}$ whose edges are labeled with $\neq$ or $\circledcirc$.

The configuration C_k is then obtained by the following steps:

1. Define C_k^{T} as the threaded graph whose thread contains the (threaded versions) of edges in S_k in this order. Additionally, all nodes that occur in T_k but not in S_k, and all edges in L_k are in C_k^{T}.
2. Replace each kernel node u of C_k^{T} by a fresh copy $copy(u)$. Add a $\otimes$-edge in C_k^{T} to $copy(u)$ if $u \notin N_k$.
3. Obtain C_k^{U} from G by removing all edges occurring in handles in T_k. Add a $\otimes$-edge in C_k^{U} to u if $u \notin N_k$.
4. Let $C_k = C_k^{\mathrm{U}} \cup C_k^{\mathrm{T}}$.
5. For each node $u \in N_k$, add a bind-edge from $copy(u)$ to $\dot{\mu}(u)$ in C_k. □

The following example illustrates this construction:

Example 6. Consider Example 2 again, and the illustration of the derivation tree $t_{\hat{T}}$ in Fig. 5, which results in the graph $\hat{T}$ of Fig. 4. The pre-order traversal of $t_{\hat{T}}$ is $pre(t_{\hat{T}}) = \tau_1 \cdots \tau_{12}$ where τ_i is the subtree of $t_{\hat{T}}$ whose root handle carries i as a small number in Fig. 5. Note that τ_7 and τ_8 are handles whose edges are labeled with $\neq$ or $\circledcirc$. Construction 1 thus ignores them and considers the following sequence $t_1 \cdots t_{10}$ of the remaining ten subtrees (again, only the root handles are depicted):



Construction 1 creates the configurations in the parse $C_0 \Rightarrow^*_{\mathcal{R}_{\Gamma^\bullet}} C_{10}$ shown in Fig. 13 from this sequence. Figure 14 displays the sequences T_k and S_k and the set L_k of handles used for creating each C_k for $k \in \{0, \ldots, 10\}$.

We are now ready to sketch a proof of Lemma 2:

Proof Sketch. We build graphs $C_0, C_1, \ldots, C_n$ following Construction 1. C_0 and C_n are clearly an initial and an accepting configuration, respectively, with $input(C_0) = G$. To see that $C_0 \Rightarrow_{\mathcal{R}_{\Gamma^\bullet}} C_1 \Rightarrow_{\mathcal{R}_{\Gamma^\bullet}} \cdots \Rightarrow_{\mathcal{R}_{\Gamma^\bullet}} C_n$, consider C_{k-1} and C_k for some $k \in [n]$. We can distinguish two cases for $H = root(t_k)$:

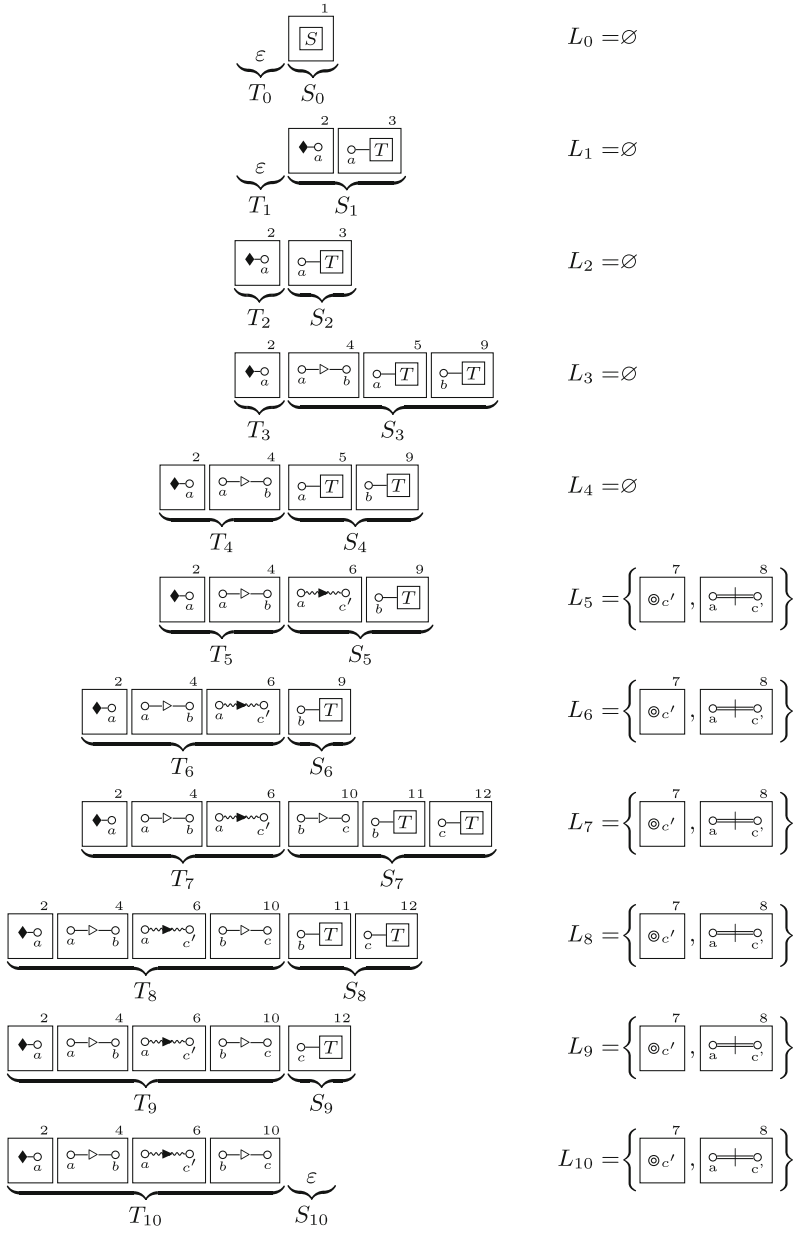


Fig. 14. Steps of Construction 1 for creating the parse in Fig. 13

Case 1: H is terminal.
This implies $T_k = T_{k-1}H$, $S_{k-1} = HS_k$, and $N_k = N_{k-1} \cup \dot{H}$. The reader can confirm by following the steps in Construction 1 that $C_{k-1} \Rightarrow_{\mathcal{R}_{\Gamma^\bullet}} C_k$ using the match rule for the edge label in H.

Case 2: H is nonterminal.
Then $t_{k-1} = \langle H, p, t'_1 t'_2 \cdots t'_l \rangle$ for derivation trees $t'_1 t'_2 \cdots t'_l$, and we have $T_k = T_{k-1}$, $N_k = N_{k-1}$, $S_{k-1} = t_{k-1} R$, and $S_k = root(t'_1)\, root(t'_2) \cdots root(t'_l)\, R$ with $R \in \mathbb{T}^*_\Gamma$. By Definition 6, $H \Rightarrow_p H^\circ \cup \bigcup_{i=1}^{l} root(t'_i)$. Construction 1 makes sure that $C_{k-1} \Rightarrow_{\mathcal{R}_{\Gamma^\bullet}} C_k$ using the expand rule for p. □

The next lemma covers the other direction of the correctness of the parser.

Lemma 3. *If Γ is acyclic, then the existence of a successful parse $C \Rightarrow^*_{\mathcal{R}_{\Gamma^\bullet}} C'$ implies $input(C) \in \mathcal{L}(\Gamma)$.*

Proof. Let $C_0 \Rightarrow_{t_1} C_1 \Rightarrow_{t_2} \cdots \Rightarrow_{t_n} C_n$ be any successful parse with $t_1, \ldots, t_n \in \mathcal{R}_{\Gamma^\bullet}$. Let $G = input(C_0)$. For $i = 0, \ldots, n$, consider the following subgraphs Top_i and Bot_i of C_i: Top_i is the subgraph induced by $\dot{C}_i \setminus \dot{G}$, and Bot_i the subgraph obtained from C_i by deleting all edges in Top_i and all thread nodes. Note that $C_i = Top_i \cup Bot_i$ and, since C_0 is initial, $Top_0 = Z^\bullet$ and $Bot_0 = G$.

Bot_i may contain bind-edges, which define the binding relation $\mapsto$ of Fact 1. For every graph H containing Bot_i as a subgraph (for any i), we obtain $merge(H)$ as the homomorphic image of H without its bind-edges by mapping node x to node y for each $(x, y) \in \mapsto$.

Let us now consider a parse step $C_{i-1} \Rightarrow_{t_i} C_i$ for $i \in [n]$. There are two cases:

Case 1: t_i is a match rule.
Then C_i is obtained by deleting a terminal edge as well as its threaded version from C_{i-1}, and by adding some bind-edges. As a consequence, there is a handle Del_i of a terminal edge with $\dot{Del}_i \subseteq X(\lceil Top_{i-1} \rfloor)$ so that

$$\lceil Top_{i-1} \rfloor = \lceil Top_i \rfloor \cup Del_i \quad \text{and} \quad merge(Bot_{i-1}) = merge(Bot_i \cup Del_i).$$

Case 2: t_i is an expand rule for a rule $p_i \in \hat{\mathcal{P}}$.
By the definition of the expand rule we have

$$\lceil Top_{i-1} \rfloor \underset{p_i}{\Rightarrow} \lceil Top_i \rfloor \quad \text{and} \quad Bot_{i-1} = Bot_i.$$

Let $Del = \bigcup_{j=1}^{n} Del_j$ where Del_j is the empty graph if t_j is an expand rule. Making use of the fact that $G = merge(Bot_0)$ and $\lceil Top_0 \rfloor = Z$, a straightforward induction yields

$$(1)\; Z \overset{*}{\underset{\hat{\mathcal{P}}}{\Rightarrow}} \lceil Top_n \rfloor \cup Del \quad \text{and} \quad (2)\; G = merge(Bot_n \cup Del).$$

Let $F = \lceil Top_n \rfloor \cup Del$. Note that for every node $x \in \dot{F}_\circledcirc$ (see Definition 8), there is a unique node $y \in \dot{F} \setminus \dot{F}_\circledcirc$ so that G has a node u with $x \mapsto u$ and $y \mapsto u$ in Bot_n. Now consider the morphism $\mu \colon F \to F'$ where $\bar{\mu}$ and the restriction of $\dot{\mu}$ to $\dot{F} \setminus \dot{F}_\circledcirc$ are inclusions, and $\dot{\mu}$ maps each node $x \in \dot{F}_\circledcirc$ to the corresponding node $y \in \dot{F} \setminus \dot{F}_\circledcirc$ as described above. The application conditions of match rules make sure that $(v, \dot{\mu}(v)) \notin \neq_F$ for every $v \in \dot{F}_\circledcirc$. Hence, μ is a joining morphism, and $core(F') \in \mathcal{L}(\Gamma)$ because of (1) and the assumption that Γ is acyclic. However, $core(F')$ is isomorphic to $merge(Bot_n \cup Del)$, and (2) implies $G \in \mathcal{L}(\Gamma)$. □

Corollary 1. *If Γ is acyclic, then $\mathcal{R}_{\Gamma^\bullet}$ is a top-down parser for Γ.*

5 Conclusions

In this paper, we have shown that our rule-based definition of top-down parsing for CHR grammars is correct if the dependencies arising from the use of context nodes in these grammars are acyclic. This extends our correctness proof for HR grammars in [8, Theorem 2]. Our result can be specialized for predictive top-down parsing by equipping expand rules with application conditions that allow to predict the only promising production for a nonterminal; cf. Theorem 4 of that paper.

The language of all graphs over Σ, unrestricted flowcharts of imperative programs, statecharts [5, Ex. 1, 2 & Sect. 3], and graphs representing object-oriented programs [4] cannot be generated with HR grammars; however, they can be generated with CHR grammars. This indicates that the extension is practically relevant. Moreover, the mentioned CHR grammars are acyclic and PTD-parsable. This suggests that these restrictions will not be too strong in practice.[2] The cyclic grammar Λ for dags in Example 4 is not PTD-parsable. (We conjecture that there is no acyclic CHR grammar for dags at all.)

Much of the related work on parsing for HR grammars follows the well-known Cocke-Younger-Kasami algorithm. An implementation for unrestricted HR grammars (plus edge-embedding rules) in DiaGen [18] works for practical input with hundreds of nodes and edges, although their worst-case complexity is exponential. D. Chiang et al. [1] have implemented a polynomial algorithm for a subclass of HR grammars (based on the work of C. Lautemann [17]). S. Gilroy, A. Lopez, and S. Maneth [12] have proposed a linear parsing algorithm for Courcelle's "regular" graph grammars [2]. Both algorithms apply to graphs as they occur in computational linguistics.

To our knowledge, early approaches to parsing for context-free node replacement grammars [10] like [15,19] are no longer pursued.

Like many scientific efforts, this paper raises more questions than it answers: (*i*) Is there a decidable sufficient and necessary condition for acyclicity? (*ii*) Can parsing for CHR grammars be extended to cyclic CHR grammars? (*iii*) Can PSR parsing [7] be defined by graph transformation in a similar way? (*iv*) Is PSR parsing more powerful than PTD parsing? All this remains for future work.

Acknowledgments. We thank Annegret Habel, Verone Stillger, and the anonymous reviewers for their advice.

[2] The mentioned CHR grammars can be downloaded at www.unibw.de/inf2/grappa. The *graph parser distiller* GRAPPA developed by Mark Minas, generates PTD parsers that run in quadratic time, and often even in linear time [6]. The website also contains specifications of the PTD parser for linked trees with the AGG system [11] along the lines of Example 5.

References

1. 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 Linguistics (vol. 1: Long Papers), Sofia, Bulgaria, pp. 924–932. Association for Computational Linguistics, August 2013
2. 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)
3. Drewes, F., Habel, A., Kreowski, H.J.: Hyperedge replacement graph grammars. In: Rozenberg [20], chapter. 2, pp. 95–162
4. Drewes, F., Hoffmann, B.: Contextual hyperedge replacement. Acta Informatica **52**(6), 497–524 (2015). https://doi.org/10.1007/s00236-015-0223-4
5. 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
6. 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
7. Drewes, F., Hoffmann, B., Minas, M.: Formalization and correctness of predictive shift-reduce parsers for graph grammars based on hyperedge replacement. J. Log. Algebraic Methods Program. (JLAMP) **104**, 303–341 (2019). https://doi.org/10.1016/j.jlamp.2018.12.006
8. Drewes, F., Hoffmann, B., Minas, M.: Graph parsing as graph transformation. In: Gadducci, F., Kehrer, T. (eds.) ICGT 2020. LNCS, vol. 12150, pp. 221–238. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-51372-6_13
9. Ehrig, H., Ehrig, K., Prange, U., Taentzer, G.: Fundamentals of Algebraic Graph Transformation. MTCSAES, Springer, Heidelberg (2006). https://doi.org/10.1007/3-540-31188-2
10. Engelfriet, J., Rozenberg, G.: Node replacement graph grammars. In: Rozenberg [20], chapter. 1, pp. 1–94
11. Ermel, C., Rudolf, M., Gabriele, T.: The AGG approach: language and environment. In: Engels, G., Ehrig, H., Kreowski, H.J., Rozenberg, G. (eds.) Handbook of Graph Grammars and Computing by Graph Transformation, Vol. II: Applications, Languages, and Tools, pp. 551–603. World Scientific, Singapore (1999)
12. Gilroy, S., Lopez, A., Maneth, S.: Parsing graphs with regular graph grammars. In: Proceedings of the 6th Joint Conference on Lexical and Computational Semantics (*SEM 2017), Vancouver, Canada, pp. 199–208. Association for Computational Linguistics, August 2017. https://doi.org/10.18653/v1/S17-1024
13. Habel, A.: Hyperedge Replacement: Grammars and Languages. LNCS, vol. 643. Springer, Heidelberg (1992). https://doi.org/10.1007/BFb0013875
14. Habel, A., Pennemann, K.H.: Correctness of high-level transformation systems relative to nested conditions. Math. Struct. Comput. Sci. **19**(2), 245–296 (2009)
15. Kaul, M.: Practical applications of precedence graph grammars. In: Ehrig, H., Nagl, M., Rozenberg, G., Rosenfeld, A. (eds.) Graph Grammars 1986. LNCS, vol. 291, pp. 326–342. Springer, Heidelberg (1987). https://doi.org/10.1007/3-540-18771-5_62
16. Lange, K.J., Welzl, E.: String grammars with disconnecting or a basic root of the difficulty in graph grammar parsing. Discret. Appl. Math. **16**, 17–30 (1987)

17. Lautemann, C.: The complexity of graph languages generated by hyperedge replacement. Acta Informatica **27**, 399–421 (1990)
18. Minas, M.: Diagram editing with hypergraph parser support. In: Proceedings of 1997 IEEE Symposium on Visual Languages (VL 1997), Capri, Italy, pp. 226–233. IEEE Computer Society Press (1997)
19. Pavlidis, T.: Linear and context-free graph grammars. J. ACM **19**(1), 11–22 (1972). https://doi.org/10.1145/321679.321682
20. Rozenberg, G. (ed.): Handbook of Graph Grammars and Computing by Graph Transformation, Vol. I: Foundations. World Scientific, Singapore (1997)