






# Graph Parsing as Graph Transformation

## Correctness of Predictive Top-Down Parsers

Frank Drewes<sup>1</sup>, Berthold Hoffmann<sup>2</sup>, and Mark Minas<sup>3</sup>

<sup>1</sup> Umeå Universitet, Umeå, Sweden  
drewes@cs.umu.se

<sup>2</sup> Universität Bremen, Bremen, Germany  
hof@uni-bremen.de

<sup>3</sup> Universität der Bundeswehr München, Neubiberg, Germany  
mark.minas@unibw.de

**Abstract.** Hyperedge replacement (HR) allows to define context-free graph languages, but parsing is NP-hard in the general case. Predictive top-down (PTD) is an efficient, backtrack-free parsing algorithm for subclasses of HR and contextual HR grammars, which has been described and implemented in earlier work, based on a representation of graphs and grammar productions as strings. In this paper, we define PTD parsers for HR grammars by graph transformation rules and prove that they are correct.

**Keywords:** Graph transformation · Hyperedge replacement · Parsing · Correctness

## 1 Introduction

Hyperedge replacement (HR, [8]) is one of the best-studied mechanisms for generating graphs. Being context-free, HR grammars inherit most of the favorable structural and computational properties of context-free string grammars. Unfortunately, simplicity of parsing is not one of these, as there are NP-complete HR languages [1, 14]. Hence, efficient parsing can only be done for suitable subclasses. The authors have devised predictive top-down (PTD, [4]) and predictive shift-reduce (PSR, [6]) parsing for subclasses of HR grammars and, in fact, for subclasses of contextual HR grammars (CHR grammars, [2, 3]), which are a modest extension of HR grammars that allows to overcome some of the structural limitations of HR languages.

Although the concepts and implementation of PTD parsers have been described at depth in [4], their correctness has not yet been formally established. We show in this paper how PTD parsing can be defined by graph transformation rules and use this in order to prove the correctness of PTD parsers. Our experience with the correctness proof for PSR parsing in [6] seems to indicate that a graph- and rule-based definition of parsers can make this task easier.

Related work on using graph transformation for defining parsers has dealt with LR string grammars [11] and two-level string grammars [12]. For a broader

discussion of related work on parsing algorithms for graph grammars in general we refer to [6, Sect. 10.1].

The paper is structured as follows. After recalling graph transformation concepts (Sect. 2) and HR grammars (Sect. 3), we introduce threaded HR grammars (Sect. 4), which impose a total order on the edges of their derived graphs, which in turn induces a dependency relation on their nodes. In Sect. 5, we define a general top-down parser for HR grammars that respects edge order and node dependencies, and prove it correct. Since this parser is nondeterministic and hence inefficient, we introduce properties that make the parser predictive, and backtrack-free (Sect. 6) and show that this yields correct parsers that terminate for grammars without left recursion.<sup>1</sup> We conclude the paper by indicating some future work (Sect. 7).

## 2 Preliminaries

In this paper,  $\mathbb{N}$  denotes the set of non-negative integers and  $[n]$  denotes  $\{1, \dots, n\}$  for all  $n \in \mathbb{N}$ .  $A^*$  denotes the set of all finite sequences over a set  $A$ ; the empty sequence is denoted by  $\varepsilon$ , and the length of a sequence  $\alpha$  by  $|\alpha|$ . As usual,  $\rightarrow^+$  and  $\rightarrow^*$  denote the transitive and the transitive reflexive closure of a binary relation  $\rightarrow$ . For a function  $f: A \rightarrow B$ , its extension  $f^*: A^* \rightarrow B^*$  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  $a_1, \dots, a_n \in A$ . The *composition* of functions  $f: A \rightarrow B$  and  $g: B \rightarrow C$  is denoted as  $g \circ f$  and defined by  $(g \circ f)(x) = g(f(x))$  for  $x \in A$ . The restriction of  $f$  to some subset  $X \subseteq A$  is denoted as  $f|_X$ .

**Definition 1 (Hypergraph).** An *alphabet*  $\Sigma$  is a finite set of symbols that comes with an *arity function*  $\text{arity}: \Sigma \rightarrow \mathbb{N}$ . A *hypergraph* (over  $\Sigma$ ) is a tuple  $G = (\dot{G}, \bar{G}, \text{att}, \text{lab})$ , where  $\dot{G}$  and  $\bar{G}$  are finite sets of *nodes* and *hyperedges*, respectively, the function  $\text{att}: \bar{G} \rightarrow \dot{G}^*$  attaches hyperedges to sequences of nodes, and the function  $\text{lab}: \bar{G} \rightarrow \Sigma$  labels hyperedges so that  $|\text{att}(e)| = \text{arity}(\text{lab}(e))$  for every  $e \in \bar{G}$ , i.e., the number of attached nodes of hyperedges is dictated by the arity of their labels.

$\mathcal{G}_\Sigma$  denotes the class of hypergraphs over  $\Sigma$ ;  $\langle \rangle$  denotes the *empty hypergraph*, with empty sets of nodes and hyperedges. A set of hyperedges  $E \subseteq \bar{G}$  *induces* the subgraph consisting of these hyperedges and their attached nodes.

For brevity, we omit the prefix “hyper” in the sequel. Instead of “ $x \in \dot{G}$  or  $x \in \bar{G}$ ”, we often write “ $x \in G$ ”. We often refer to the functions of a graph  $G$  by  $\text{att}_G$  and  $\text{lab}_G$ . An edge carrying a label in an alphabet  $\Sigma$  is also called a  $\Sigma$ -edge. And a node is called *isolated* if no edge is attached to it.

**Definition 2 (Graph Morphism).** Given graphs  $G$  and  $H$ , a *graph morphism* (morphism, for short)  $m: G \rightarrow H$  is a pair  $m = (\dot{m}, \bar{m})$  of functions  $\dot{m}: \dot{G} \rightarrow \dot{H}$

<sup>1</sup> Since this paper is dedicated to proving the correctness of PTD parsers, and it has been established in [4] that they run in quadratic time at worst, we shall not dwell on issues of efficiency here.

and  $\bar{m}: \bar{G} \rightarrow \bar{H}$  that preserve attachments and labels, i.e.,  $att_H \circ \bar{m} = \bar{m}^* \circ att_G$  and  $lab_H \circ \bar{m} = lab_G$ . The morphism is *injective* or *surjective* if both  $\bar{m}$  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  $G \cong H$ .

For transforming graphs, we use the classical approach of [7], with injective matching and non-injective rules [9], but without rules that delete nodes.

**Definition 3 (Rule).** A *graph transformation rule*  $r = (P, R, r^\circ)$  consists of a *pattern graph*  $P$ , a *replacement graph*  $R$ , and a mapping  $r^\circ: \dot{P} \rightarrow \dot{R}$ .<sup>2</sup> We briefly call  $r$  a *rule* and denote it as  $r: P \circ \rightarrow R$ . An injective morphism  $m: P \rightarrow G$  into a graph  $G$  is a *match* of  $r$ , and  $r$  *transforms*  $G$  at  $m$  to a graph  $H$  as follows:

- Remove all edges  $m(e)$ ,  $e \in \bar{P}$ , from  $G$  to obtain a graph  $K$ .
- Construct  $H$  from the disjoint union of  $K$  and  $R$  by identifying  $m(x)$  with  $r^\circ(x)$  for every  $x \in \dot{P}$ .

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  $r \in \mathcal{R}$ .

Sometimes it is necessary to restrict the application of a rule by requiring the existence or non-existence of certain graphs in the context of its match. Our definition of application conditions is based on [10].

**Definition 4 (Conditional Rule).** For a graph  $P$ , the set of *conditions over*  $P$  is defined inductively as follows: (i) a subgraph relation  $P \subseteq C$  defines a *basic condition*  $\exists C$  over  $P$ . (ii) if  $c, c'$  are conditions over  $P$ , then  $\neg c$ ,  $(c \wedge c')$ , and  $(c \vee c')$  are conditions over  $P$ .<sup>3</sup>

An injective morphism  $m: P \rightarrow G$  *satisfies* a condition  $c$ , written  $m \models c$ , if

- $c = \exists C$  and there is an injective morphism  $m': C \rightarrow G$  so that  $m'|_P = m$ ;
- $c = \neg c'$  and  $m \not\models c'$ ;
- $c = (c' \wedge c'')$  and both  $m \models c'$  and  $m \models c''$ ;
- $c = (c' \vee c'')$  and  $m \models c'$  or  $m \models c''$ .

A *conditional rule*  $r' = (r, c)$  consists of a rule  $r: P \circ \rightarrow R$  and a condition  $c$  over  $P$ , and is denoted as  $r': c \parallel P \circ \rightarrow R$ . We let  $G \Rightarrow_{r'}^m H$  if  $m \models c$  and  $G \Rightarrow_r^m H$ . Note that each rule  $P \circ \rightarrow R$  without a condition can also be seen as a conditional rule  $\exists P \parallel P \circ \rightarrow R$ . If  $\mathcal{C}$  is a finite set of conditional rules,  $\Rightarrow_{\mathcal{C}}$  denotes the conditional transformation relation using these rules.

Examples of graphs and rules, with and without conditions, will be shown below.

<sup>2</sup> This corresponds to a DPO rule  $P \supseteq I \rightarrow R$ , where the interface  $I$  is the discrete graph with nodes  $\dot{P}$ , and the morphism  $I \rightarrow R$  is given by  $(r^\circ, \emptyset)$ .

<sup>3</sup> We omit *nested conditions* like “ $\forall(C, \exists C' \wedge \neg \exists C'')$ ” since we do not need them.

### 3 Hyperedge Replacement Graph Grammars

We recall graph grammars based on hyperedge replacement [8].<sup>4</sup>

**Definition 5 (Hyperedge Replacement Grammar).** Consider a finite alphabet  $\Sigma$  and a subset  $\mathcal{N} \subseteq \Sigma$  of *nonterminals*. Edges with labels in  $\mathcal{N}$  are accordingly *nonterminal edges*; those with labels in  $\Sigma \setminus \mathcal{N}$  are *terminal edges*.

A rule  $p: P \multimap R$  is a *hyperedge replacement production* (*production*, for short) over  $\Sigma$  if the pattern  $P$  consists of a single edge  $e$  and its attached nodes, where  $lab_P(e) \in \mathcal{N}$ , and the mapping  $p^\circ: \dot{P} \rightarrow \dot{R}$  is injective.

A *hyperedge-replacement grammar* (*HR grammar*)  $\Gamma = \langle \Sigma, \mathcal{N}, \mathcal{P}, Z \rangle$  consists of  $\Sigma$  and  $\mathcal{N} \subseteq \Sigma$  as above, a finite set  $\mathcal{P}$  of productions over  $\Sigma$ , and a start graph  $Z \in \mathcal{G}_\Sigma$ .

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

*Example 1 (HR Grammars for Trees).* As a running example for the constructions in this paper, we use the productions in Fig. 1. They derive  $n$ -ary trees like the one in Fig. 2, if the pattern of production S is the start graph. We draw nodes as circles, and nonterminal edges as boxes that contain their labels. Edges are connected to their attached nodes by lines, called tentacles. Tentacles are ordered counter-clockwise around the edge, starting in the north.

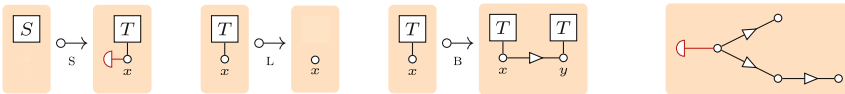


Fig. 1. HR productions for trees

Fig. 2. A tree

For the purpose of this paper, we restrict ourselves to this simple example because illustrations would otherwise become too complex. Further examples of well-known HR languages for which PTD parsers can be built include string graph languages such as palindromes, non-context-free ones like  $a^n b^n c^n$ , arithmetic expressions, and Nassi-Shneiderman diagrams.

In our running example, edges of shape  $\lhd$  with  $arity(\lhd) = 1$  designate root nodes, whereas edges of shape  $\triangleright$  with  $arity(\triangleright) = 2$  connect parent nodes to their children.

In productions (and later in other rules), nodes of the pattern  $P$  have the same identifier ascribed in  $P$  as their images in  $R$  under  $p^\circ$ , like  $x$  in our example. In the following, the letters S, L, and B under the arrows in Fig. 1 are used as identifiers that refer to the corresponding production.

<sup>4</sup> In contrast to [8] and [4], “merging rules”, with a non-injective node mapping, are prohibited here as they complicate the following formal discussion considerably.

**Assumption 1.** Throughout the remainder of this paper, we consider only HR grammars  $\Gamma = \langle \Sigma, \mathcal{N}, \mathcal{P}, Z \rangle$  that satisfy the following conditions:

1.  $Z$  consists of a single edge  $e$  of arity 0.
2.  $\mathcal{L}(\Gamma)$  does not contain graphs with isolated nodes.

These assumptions imply no loss of generality: a new initial nonterminal with a single start production according to Assumption 1 can be added easily. A grammar that violates Assumption 1 and produces isolated nodes can be transformed easily into an equivalent grammar that attaches virtual unary edges to those nodes.

## 4 Threaded HR Grammars

We now prepare HR grammars for parsing. The edges in graphs, productions and derivations will be ordered linearly with the idea that the parser is instructed to process the symbols of a grammar in this order when it attempts to construct a derivation for a given input graph. The edge order induces a dependency relation between nodes of a graph as follows: for an edge, an attached node is “known” if it is also attached to some preceding edge, which will be processed earlier by the parser; it is “unknown” otherwise. This defines what we call the profile of an edge: a node is classified as incoming if it is known, and as outgoing otherwise.

Technically, edge order and profiles are represented by extending the structure and labels of a graph: Every edge is equipped with two additional tentacles by which edges are connected to a thread, and the label  $\ell$  of an edge is equipped with a profile  $\nu \subseteq \mathbb{N}$  indicating the positions of its incoming nodes. Unary hyperedges labeled with a fresh symbol distinguish thread nodes from kernel nodes of a graph.

**Definition 6 (Threaded Graph).** The *profiled alphabet* of an alphabet  $\Sigma$  is  $\tilde{\Sigma} = \{\ell^\nu \mid \ell \in \Sigma, \nu \subseteq [\text{arity}(\ell)]\} \cup \{\bullet\}$  with  $\text{arity}(\ell^\nu) = \text{arity}(\ell) + 2$  and  $\text{arity}(\bullet) = 1$ . The *profile* of an edge labelled by  $\ell^\nu$  is  $\nu$ .

Let  $G \in \mathcal{G}_{\tilde{\Sigma}}$ . A node  $v \in \tilde{G}$  is called a *thread* node if a  $\bullet$ -edge is attached to it and a *kernel* node otherwise.  $\check{G}$  and  $\dot{G}$  denote the sets of all kernel nodes and thread nodes of  $G$ , respectively. An edge  $e \in \tilde{G}$  is a *profiled edge* if  $\text{lab}_G(e) \neq \bullet$ . The set of all profiled edges of  $G$  is denoted by  $pe(G)$ . The profile  $\nu$  divides the set of attached kernel nodes of  $e$  into sets  $\text{in}_G(e) = \{v_i \mid i \in \nu\}$  and  $\text{out}_G(e) = \{v_i \mid i \in [\text{arity}(\text{lab}_G(e))] \setminus \nu\}$  of *incoming* and *outgoing* nodes, respectively.

A graph  $G \in \mathcal{G}_{\tilde{\Sigma}}$  is *threaded* if the following hold:

1. Each node of  $G$  has at most one attached  $\bullet$ -edge.
2. For every  $e \in pe(G)$  with  $\text{lab}_G(e) = \ell^\nu$  and  $\text{att}_G(e) = v_1 \dots v_k v_{k+1} v_{k+2}$ , the nodes  $v_1, \dots, v_k$  are kernel nodes of  $G$  and  $v_{k+1}, v_{k+2}$  are thread nodes of  $G$ . (Hence,  $\text{in}_G(e)$  and  $\text{out}_G(e)$  partition the kernel nodes of  $e$  into incoming and outgoing nodes.)

3. The profiled edges and thread nodes of  $G$  can be ordered as  $pe(G) = \{e_1, \dots, e_n\}$  and  $\tilde{G} = \{v_0, \dots, v_n\}$  so that, for  $i \in [n]$ ,
- (a)  $att_G(e_i)$  ends in  $v_{i-1}v_i$  and
  - (b) no edge  $e_j$  with  $j \in [i - 1]$  is attached to any node in  $out_G(e_i)$ .

We call  $v_0$  the *first* and  $v_n$  the *last* thread node of  $G$ , and define furthermore  $in(G) = \tilde{G} \setminus \bigcup_{i \in [n]} out_G(e_i)$ .

The *kernel graph* of  $G$  is the graph  $\underline{G} \in \mathcal{G}_\Sigma$  obtained by removing the profiles of edge labels, the  $\bullet$ -edges, the thread nodes and their attached tentacles.  $\tilde{\mathcal{G}}_{\tilde{\Sigma}}$  denotes the set of threaded graphs over  $\tilde{\Sigma}$ ;  $\langle \bullet \rangle$  denotes the *empty threaded graph* that consists of a single thread node with its attached  $\bullet$ -edge.

*Remark 1* It is important to note that the profiles of the (profiled) edges of a threaded graph  $G$  are uniquely determined by  $in(G)$  and the structure of  $G$ . To see this, let  $pe(G) = \{e_1, \dots, e_n\}$ , threaded in this order. For every  $v \in \underline{G}$ , let

$$first(v) = \begin{cases} 0 & \text{if } v \in in(G) \\ i & \text{if } v \notin in(G) \text{ and } i = \min\{j \in [n] \mid att_G(e_j) \text{ contains } v\}. \end{cases}$$

Then  $v \in in_G(e_i)$  if  $v \in att_G(e_i)$  and  $first(v) < i$ .

Let the *concatenation*  $H = G \circ G'$  of two threaded graphs  $G$  and  $G'$  with  $\bar{G} \cap \bar{G}' = \tilde{G} \cap \tilde{G}' = \emptyset$  be the threaded graph  $H$  that is constructed from the union of  $G$  and  $G'$  by identifying the last thread node of  $G$  with the first thread node of  $G'$  (and removing one of their attached  $\bullet$ -edges). Note that kernel nodes of  $G$  may also occur in  $G'$ .

**Definition 7 (Threaded Production and HR grammar).** A rule  $p: P \circ \rightarrow R$  is a *threaded production* if  $P$  and  $R$  are threaded and the following conditions are satisfied:

1. the rule  $\underline{p}: \underline{P} \circ \rightarrow \underline{R}$ , where  $\underline{p}^\circ$  is the restriction of  $p^\circ$  to  $\underline{P}$ , is a production, called *kernel production* of  $p$ ,
2.  $p^\circ$  maps the first and last thread nodes of  $P$  onto the first and last thread nodes of  $R$ , respectively, and
3.  $p^\circ(in(P)) = in(R)$ .

An application  $G \Rightarrow_p^m H$  of a threaded production  $p$  to a threaded graph  $G$  is called *leftmost*, written  $G \Rightarrow_{lm}^m H$ , if it replaces the first nonterminal on the thread of  $G$ .

A HR grammar  $\tilde{\Gamma} = \langle \tilde{\Sigma}, \tilde{\mathcal{N}}, \tilde{\mathcal{P}}, \tilde{\mathcal{Z}} \rangle$  over a profiled alphabet  $\tilde{\Sigma}$  is *threaded* if all its productions are threaded.

As in the case of context-free string grammars, the context-freeness of hyper-edge replacement implies that derivations can be restricted to leftmost ones:

**Fact 1.** For every threaded HR grammar  $\tilde{\Gamma} = \langle \tilde{\Sigma}, \tilde{\mathcal{N}}, \tilde{\mathcal{P}}, \tilde{\mathcal{Z}} \rangle$  and every  $G \in \mathcal{L}(\tilde{\Gamma})$ , there is a leftmost derivation  $\tilde{\mathcal{Z}} \Rightarrow_{lm}^* G$ , i.e., a derivation in which all applications of productions are leftmost.

This fact will be important, as top-down parsers for HR grammars attempt to construct leftmost derivations of a graph.

It follows from Remark 1 and condition 3 of Definition 7 that the profiles of edges in the replacement graph of a threaded production are uniquely determined by the profile of the pattern. Hence, given a HR grammar  $\Gamma = \langle \Sigma, \mathcal{N}, \mathcal{P}, Z \rangle$  and an order on  $\tilde{R}$  for each of its productions  $p: P \circ \rightarrow R$ , a unique threaded version  $\tilde{\Gamma}$  of  $\Gamma$  is obtained as follows:

1. The threaded start graph  $\tilde{Z}$  of  $\tilde{\Gamma}$  is given by  $\tilde{Z} = Z$  (recall that  $arity(Z) = 0$ ).
2. Every production  $p: P \circ \rightarrow R$  of  $\Gamma$  is turned into all threaded productions  $\tilde{p}: \tilde{P} \circ \rightarrow \tilde{R}$  where  $\tilde{P} = P$ ,  $\tilde{R} = R$ , and the edges of  $\tilde{R}$  are threaded according to the chosen order on  $\tilde{R}$  (which defines the profiles of edges in  $\tilde{R}$  uniquely).

While the procedure above creates an exponential number of profiles and thus productions, in most cases many of them will be useless. A more efficient way of constructing  $\tilde{\Gamma}$  is thus to choose the threading order and then construct the useful threaded productions inductively. The procedure would initially construct the threaded start production (in which  $in(P) = \emptyset$ ) and then, as long as a replacement graph of one of the constructed productions contains a hitherto unseen profiled nonterminal, continue by constructing the threaded productions for this nonterminal. This leads to the following definition:

**Definition 8 (Threaded Version of a HR Grammar).** Let  $\Gamma = \langle \Sigma, \mathcal{N}, \mathcal{P}, Z \rangle$  be a HR grammar. A *threaded version* of  $\Gamma$  is a threaded grammar  $\tilde{\Gamma} = \langle \Sigma, \tilde{\mathcal{N}}, \tilde{\mathcal{P}}, \tilde{Z} \rangle$ , such that

1.  $\mathcal{P} = \{ \underline{p} \mid p \in \tilde{\mathcal{P}} \}$  and  $Z = \tilde{Z}$ ,
2. all threaded productions with the same kernel production  $p: P \circ \rightarrow R$  order the edges of  $R$  identically, and
3.  $\tilde{\Gamma}$  is reduced, i.e., every production  $p \in \tilde{\mathcal{P}}$  can participate in the generation of a graph in  $\mathcal{L}(\tilde{\Gamma})$ : there is a derivation  $\tilde{Z} \xrightarrow[\tilde{p}]{*} G \Rightarrow G' \xrightarrow[p]{*} H$  such that  $H \in \mathcal{L}(\tilde{\Gamma})$ .

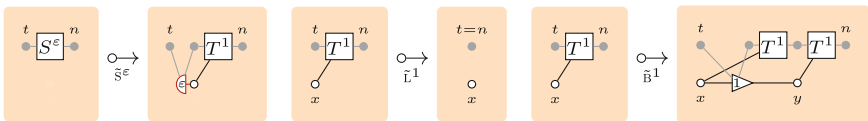


Fig. 3. Threaded tree productions

*Example 2 (Threaded Tree Grammar).* We consider a threaded version of the tree grammar, given by the threaded productions in Fig. 3. In examples such as this one we draw thread nodes in gray and omit the attached  $\bullet$ -edges, and we write profiles as ascending sequences of numbers rather than as sets. The profiles of profiled terminal edges are inscribed into the label symbols, i.e.,  $\triangleright$  for  $\triangleright^1$  and

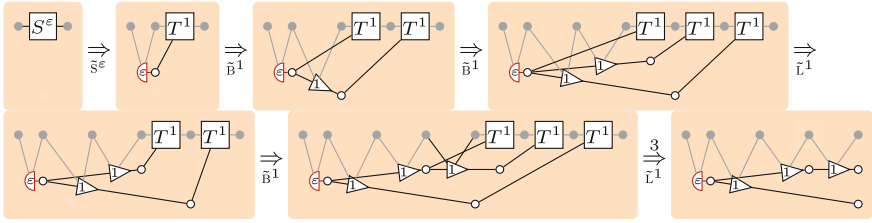


Fig. 4. A leftmost threaded derivation of the tree in Fig. 2

⊞ for ⊞<sup>ε</sup> Moreover, we distinguish threaded productions with the same kernel productions by the profile of the (unique edge in the) pattern in the production name. The profiled symbols  $T^\varepsilon$ ,  $\triangleright^\varepsilon$ ,  $\triangleright^2$ ,  $\triangleright^{12}$ , and  $\triangleright^1$  do not appear as they occur only in useless productions.

It is worthwhile to note that production  $\tilde{l}^1$  merges thread nodes  $t$  and  $n$ , which we indicate in the drawing by annotating the corresponding node in the replacement graph with “ $t=n$ ”.

We arrange thread nodes from left to right and draw thread tentacles in gray so that the kernel graph can be better identified. To make it easier to distinguish incoming from outgoing attached nodes, we draw the former to the left of an edge and the latter to the right of it.

In production  $\tilde{b}^1$ , left-recursion was avoided by choosing the terminal edge to be the first one on the thread. Figure 4 shows a threaded derivation of the tree in Fig. 2, which is leftmost.

Threaded productions derive threaded graphs to threaded graphs.

**Fact 2.** If  $G \Rightarrow_{\tilde{\mathcal{P}}} H$  and  $G$  is a threaded graph,  $H$  is a threaded graph as well, and  $in(H) = in(G)$ .

Threaded derivations and unthreaded ones correspond to each other.

**Lemma 1.** Let  $\Gamma = \langle \Sigma, \mathcal{N}, \mathcal{P}, Z \rangle$  be a HR grammar,  $\tilde{\Gamma} = \langle \tilde{\Sigma}, \tilde{\mathcal{N}}, \tilde{\mathcal{P}}, \tilde{Z} \rangle$  a threaded version of  $\Gamma$ , and  $G$  a threaded graph such that  $\tilde{Z} \Rightarrow_{\tilde{\mathcal{P}}}^* G$ . Then it holds for all graphs  $G'$  that  $\underline{G} \Rightarrow_{\mathcal{P}} G'$  if and only if there is a threaded graph  $H$  with  $\underline{H} = G'$  and  $G \Rightarrow_{\tilde{\mathcal{P}}} H$ .

Thus the threaded and unthreaded version of a HR grammar generate the same language of kernel graphs.

**Theorem 1.** If  $\Gamma = \langle \Sigma, \mathcal{N}, \mathcal{P}, Z \rangle$  is a HR grammar and  $\tilde{\Gamma} = \langle \tilde{\Sigma}, \tilde{\mathcal{N}}, \tilde{\mathcal{P}}, \tilde{Z} \rangle$  is a threaded version of  $\Gamma$ , then  $\mathcal{L}(\Gamma) = \{ \underline{G} \mid G \in \mathcal{L}(\tilde{\Gamma}) \}$ .

*Proof.* Easy induction on the length of derivations, using Lemma 1. □



## 5 General Top-Down Parsing for HR Grammars

We define top-down parsers for HR grammars as stack automata, which perform transitions of configurations that represent the input graph and a stack. Configurations are graphs, and transitions are described by graph transformation rules. This definition is more precise than the original definition of PTD parsing in [4], but avoids the technical complications occurring in the precise definition of PSR parsing for HR grammars [6], where graphs are represented textually as sequences of literals, and transitions are defined by the transformation of literal sequences, involving substitution and renaming operations on node identifiers. The use of graph transformation and graph morphisms avoids the explicit handling of these technical issues.

A configuration consists of a threaded graph as in Definition 6, which represents its stack and its read input, edges without profile that induce its unread input, and further edges that serve as flags, distinguishing different types of nodes.

**Definition 9 (Configuration).** Given a HR grammar  $\Gamma = \langle \Sigma, \mathcal{N}, \mathcal{P}, Z \rangle$  and its profiled alphabet  $\tilde{\Sigma}$ , let  $\blacksquare$ ,  $\otimes$ , and  $\circ$  be fresh symbols of arity 1. A graph  $G$  without isolated nodes is a *configuration* (of  $\Gamma$ ) if the following hold:

- The subgraph  $thread(G)$  induced by its  $\tilde{\Sigma}$ -edges is a threaded graph.
- Exactly one thread node  $h$  of  $thread(G)$  is attached to a  $\blacksquare$ -edge, representing the top of the stack.
- Every kernel node of every profiled edge between the start node of the thread and  $h$  is attached to a  $\circ$ -edge, marking it as read.
- Every node of every  $\Sigma$ -edge that is not attached to a profiled edge at the same time is attached to a  $\otimes$ -edge, marking it as unread.
- No node is attached to several edges with labels in  $\{\blacksquare, \otimes, \circ\}$ .

We let  $read(G)$ , the *read input*, denote the subgraph of  $thread(G)$  induced by the profiled edges between the first thread node and  $h$  (including the  $\bullet$ -edges attached to those nodes). The (threaded) subgraph of  $thread(G)$  induced by the profiled edges between  $h$  and the last node of the thread (again including the  $\bullet$ -edges attached to those nodes) represents the *stack*  $stack(G)$ , and the subgraph  $unread(G)$  induced by the  $\Sigma$ -edges represents the *unread input*. The union of  $unread(G)$  and the kernel of  $read(G)$  is the *input* represented by  $G$ , denoted by  $input(G)$ .

A configuration  $G$  is

- *initial* if  $read(G) = \langle \bullet \rangle$  and  $stack(G) = \tilde{Z}$ , and
- *accepting* if  $stack(G) = \langle \bullet \rangle$  and  $unread(G) = \langle \rangle$ .

**Definition 10 (Top-Down Parser).** Let  $\Gamma$  be a HR grammar and  $\mathcal{R}$  a set of conditional rules. A derivation  $G \Rightarrow_{\mathcal{R}}^* H$  is a *parse* if  $G$  is an initial configuration. A parse  $G \Rightarrow_{\mathcal{R}}^* H$  is *successful* if  $H$  is an accepting configuration. A configuration  $G$  is *promising* (with respect to  $\mathcal{R}$ ) if there is an accepting configuration  $H$  so

that  $G \Rightarrow_{\mathcal{R}}^* H$ .  $\mathcal{R}$  is a *top-down parser* for  $\Gamma$  if, for each initial configuration  $G$ ,  $\text{unread}(G) \in \mathcal{L}(\Gamma)$  if and only if  $G$  is promising.  $\mathcal{R}$  *terminates* if there is no infinite parse.

Consider in the following a threaded version  $\tilde{\Gamma} = \langle \tilde{\Sigma}, \tilde{\mathcal{N}}, \tilde{\mathcal{P}}, \tilde{Z} \rangle$  of a HR grammar  $\Gamma = \langle \Sigma, \mathcal{N}, \mathcal{P}, Z \rangle$ . We define two types of general top-down parsing rules, called match and expand rules.

**Definition 11 (Match and Expand Rules).** For every terminal symbol  $a^\nu \in \tilde{\Sigma} \setminus \tilde{\mathcal{N}}$ , the *match rule*  $t_{a^\nu} : P \circlearrowright R$  is given as follows:

- The pattern  $P$  is a configuration where
  - $\text{read}(P) = \langle \bullet \rangle$ ,
  - $\text{unread}(P)$  consists of one  $a$ -edge  $\underline{e}$  with  $a \in \Sigma \setminus \mathcal{N}$  and  $\text{att}_P(\underline{e}) = \underline{v}_1 \dots \underline{v}_k$  (where  $\text{arity}(a) = k$ ), with a  $\circ$ -edge attached to every  $\underline{v}_i$  with  $i \in \nu$  and a  $\otimes$ -edge attached to every  $\underline{v}_i$  with  $i \notin \nu$ , and
  - $\text{stack}(P)$  consists of one  $a^\nu$ -edge  $e$  with  $\text{att}_P(e) = v_1 \dots v_k v_{k+1} v_{k+2}$  such that  $v_i = \underline{v}_i$  if  $i \in \nu$ . If  $i \notin \nu$ , then  $v_i$  is not attached to  $\underline{e}$ .
- The replacement  $R$  is a configuration where
  - $\text{read}(R) = \text{stack}(P)$ , with a  $\circ$ -edge attached to every  $v_i$ , for  $i \in [k]$ ,
  - $\text{stack}(R) = \langle \bullet \rangle$ ,
  - $\text{unread}(R) = \langle \rangle$ .
- The mapping  $t_{a^\nu}$  identifies node  $v_i$  with  $\underline{v}_i$  if and only if  $i \notin \nu$ .

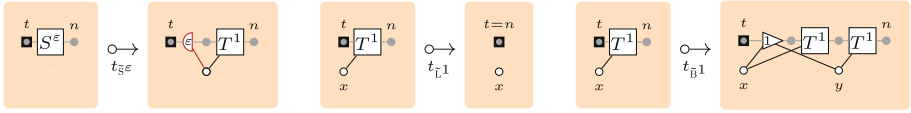
For each of the threaded productions  $p: \tilde{P} \circlearrowright \tilde{R}$  in  $\tilde{\mathcal{P}}$ , the *expand rule*  $t_p: P \circlearrowright R$  is given as follows:

- $\text{read}(P) = \text{read}(R) = \langle \bullet \rangle$ ,
- $\text{unread}(P) = \text{unread}(R) = \langle \rangle$ ,
- $\text{stack}(P) = \tilde{P}$  and  $\text{stack}(R) = \tilde{R}$ ,
- the mapping  $t_p^\circ$  is the same as in  $p$ ;

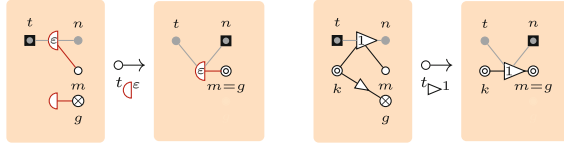
We let  $\mathcal{R}_{\tilde{\Gamma}}^{\text{M}}$  denote the set of all match rules for terminal symbols, and  $\mathcal{R}_{\tilde{\Gamma}}^{\text{E}}$  the set of all expand rules for productions of  $\tilde{\Gamma}$ . In the following, we will show that  $\mathcal{R}_{\tilde{\Gamma}} = \mathcal{R}_{\tilde{\Gamma}}^{\text{M}} \cup \mathcal{R}_{\tilde{\Gamma}}^{\text{E}}$  is in fact a top-down parser for  $\Gamma$ , hence we call  $\mathcal{R}_{\tilde{\Gamma}}$  a *general top-down parser of  $\tilde{\Gamma}$  (for  $\Gamma$ )*.

*Example 3 (General Top-Down Parser for Trees).* The expand rules of the general top-down parser for trees in Fig. 5 differ from the threaded productions only in the  $\blacksquare$ -edge marking the top of the stack. (We draw  $\blacksquare$ - and  $\circ$ -edges *around* the nodes to which they are attached, so that they look like distinguished kinds of nodes. Nodes with an attached  $\otimes$ -edge are drawn as  $\otimes$ , omitting the attached edge in the drawing.) The match rules for the two edge patterns needed are shown in Fig. 6.

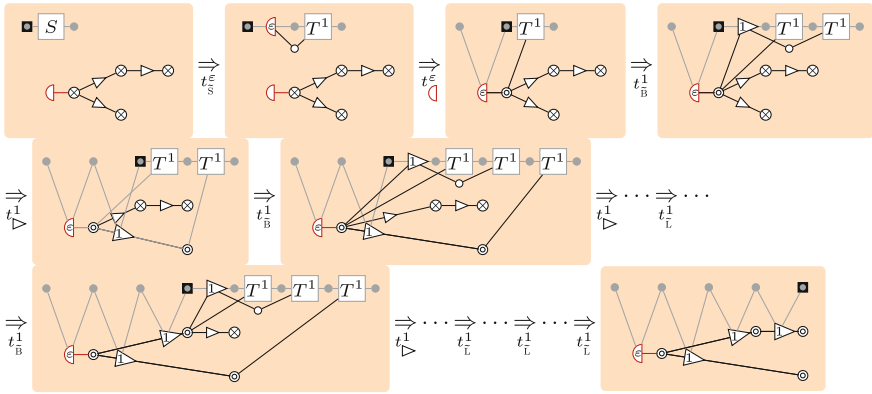
Figure 7 shows snapshots of a successful parse of the tree in Fig. 2 with these rules, where five configurations are omitted for brevity. The parse constructs the leftmost derivation in Fig. 4.



**Fig. 5.** Expand rules of the general top-down parser for trees



**Fig. 6.** Two match rules of the general top-down parser for trees



**Fig. 7.** A top-down parse of the tree in Fig. 2

Note that match rules do not change the thread, but just “move” the matched terminal from the unread to the read subgraph of the configuration. In contrast, expand rules do not modify the unread or read subgraphs of the configuration, but just replace the first nonterminal on the thread by the replacement graph of a threaded production for this nonterminal. We can summarize these observations in the following fact:

**Fact 3.** For a parse  $G \Rightarrow_{\mathcal{R}_{\tilde{F}}}^* G' \Rightarrow_r H$  (where  $r \in \mathcal{R}_{\tilde{F}}$ ), the following hold:

1.  $input(G) \cong input(G') \cong input(H)$ ;
2. if  $r = t_{a^\nu}$  is a match for some  $a \in \Sigma \setminus \mathcal{N}$ , then  $thread(G') \cong thread(H)$ ;
3. if  $r = t_p$  for some threaded production  $p \in \tilde{\mathcal{P}}$ , then  $thread(G') \xrightarrow{\text{lm}}_p thread(H)$ .

Thus  $\mathcal{R}_{\tilde{F}}$  constitutes a top-down parser: there is a successful parse if and only if its input graph is in the language of the grammar.

**Theorem 2.** For every HR grammar  $\Gamma$  and each threaded version  $\tilde{\Gamma}$  of  $\Gamma$ ,  $\mathcal{R}_{\tilde{F}}$  is a top-down parser for  $\Gamma$ .

*Proof Sketch.* Let  $G \Rightarrow_{\mathcal{R}_{\tilde{F}}}^* H$  be a successful parse.  $\tilde{Z} = thread(G) \Rightarrow_{\tilde{\mathcal{P}}}^* thread(H)$  and  $unread(G) = input(G) \cong input(H)$  hold by Fact 3;  $input(H)$  is the kernel of  $thread(H)$  because  $H$  is accepting, and hence  $unread(G) \in \mathcal{L}(\Gamma)$  by Lemma 1.

In order to show the opposite direction, let us consider any configuration  $G$  with terminal read input  $read(G)$  and  $H'$  a terminal threaded graph with kernel  $\underline{H}' = unread(G)$ . It is easy to prove, by induction on the length of the derivation, that  $thread(G) \xrightarrow{\text{lm}}_{\tilde{\mathcal{P}}}^* read(G) \circ H'$  implies  $G \Rightarrow_{\mathcal{R}_{\tilde{F}}}^* H$  where  $H$  is an accepting configuration obtained from  $read(G) \circ H'$  by adding a  $\blacksquare$ -edge to the last thread node and  $\circ$ -edges to all kernel nodes, i.e.,  $G$  is promising. Now let  $G$  be an initial configuration with  $unread(G) \in \mathcal{L}(\Gamma)$ . By Lemma 1, there is a threaded graph  $H'$  with kernel  $unread(G)$  and  $thread(G) = \tilde{Z} \xrightarrow{\text{lm}}_{\tilde{\mathcal{P}}}^* H' = read(G) \circ H'$ . Hence,  $G$  must be promising. □

If  $\tilde{\Gamma}$  is not left-recursive, the general top-down parser terminates. Here, we say that  $\tilde{\Gamma} = \langle \tilde{\Sigma}, \tilde{\mathcal{N}}, \tilde{\mathcal{P}}, \tilde{Z} \rangle$  is left-recursive if there is a threaded graph  $G$  consisting of a single nonterminal edge labeled  $A$  (for some nonterminal  $A$ ) and there is a derivation  $G \Rightarrow_{\tilde{\mathcal{P}}}^+ H$  for some graph  $H$  such that the first profiled edge of  $H$  is also labeled with  $A$ .

**Theorem 3 (Termination).** Let  $\tilde{\Gamma}$  be a threaded version of a HR grammar. The general top-down parser  $\mathcal{R}_{\tilde{F}}$  terminates unless  $\tilde{\Gamma}$  is left-recursive.

*Proof* Assume that there is an infinite parse  $G \Rightarrow_{t_1} G_1 \Rightarrow_{t_2} G_2 \Rightarrow_{t_3} \dots$  with  $t_i \in \mathcal{R}_{\tilde{F}}$  for  $i \in \mathbb{N}$ . Since  $unread(G)$  is finite and each match operation “removes” an unread edge, there must be a  $k \in \mathbb{N}$  such that  $t_i$  is an expand rule for all  $i > k$ . As their number is finite, there must be numbers  $i$  and  $j$ ,  $k < i < j$ , such that  $stack(G_i)$  and  $stack(G_j)$  start with edges labeled with the same nonterminal. By Fact 3,  $thread(G_i) \xrightarrow{\text{lm}}_{\tilde{\mathcal{P}}}^+ thread(G_j)$ , which proves that  $\tilde{\Gamma}$  is left-recursive. □

Inconveniently, the steps of the general top-down parser are nondeterministic:

1. The *expansion* of a nonterminal  $A^\nu$  may choose any of its productions.
2. The *match* of an edge  $a^\nu$  may choose any unread edge fitting the profile  $\nu$ .

We consider a parse  $G \Rightarrow_{\mathcal{R}_{\tilde{F}}}^* H$  as a *blind alley* if the configuration  $H$  is not accepting, but does not allow further steps (using  $\mathcal{R}_{\tilde{F}}$ ). This is the case if

- $stack(H)$  starts with an edge  $a'$ , but  $t_{a'}$  does not apply (*edge mismatch*), or
- $stack(H) = \langle \bullet \rangle$  but  $unread(H) \neq \langle \rangle$  (*input too big*).

Due to nondeterminism, a successful parse may nevertheless exist in such a situation. Exploring the entire search space of parses to determine whether a successful one exists is very inefficient.

## 6 Predictive Top-Down Parsing for HR Grammars

The aim of predictive top-down parsing for threaded HR grammars is to avoid backtracking, the major source of inefficiency of a straightforward implementation of the general top-down parser. So we have to cope with the nondeterminism identified in the previous section. In every configuration of a parse, it must efficiently be possible to predict which choices of moves are *wrong* in the sense that they lead into a blind alley, whereas other moves could still lead to a successful parse if there is any. However, this is most likely not achievable for every threaded HR grammar  $\tilde{\Gamma}$  because Theorem 2 in combination with the known NP-completeness of some HR languages would otherwise imply that P=NP. For such a grammar, certain configurations will allow more than one expansion, and it may be the case that any of them is promising, or just some of them (or none).

Thus backtrack-free parsing only seems to be possible for HR grammars that make correct moves of their top-down parsers predictable.

Let us first define *predictive expand rules* that will prevent a parser from running into blind alleys by additionally checking so-called *lookahead conditions*. Henceforth, given a rule  $r: P \multimap R$  and a condition  $c$  over  $P$ , we denote the conditional rule  $r': c \parallel P \multimap R$  by  $r[c]$ .

**Definition 12 (Predictive expand rules).** Let  $\Gamma$  be a HR grammar,  $\tilde{\Gamma}$  a threaded version of  $\Gamma$ , and  $\mathcal{R}_{\tilde{\Gamma}} = \mathcal{R}_{\tilde{\Gamma}}^M \cup \mathcal{R}_{\tilde{\Gamma}}^E$  its general top-down parser. For an expand rule  $t_{p\nu}: P \multimap R \in \mathcal{R}_{\tilde{\Gamma}}^E$ , a condition  $c$  over  $P$  is a *lookahead condition* for  $t_{p\nu}$  if the following holds:

For every derivation  $G \Rightarrow_{\mathcal{R}_{\tilde{\Gamma}}}^* H \Rightarrow_{t_{p\nu}}^m H'$  where  $G$  is an initial configuration and  $H$  is promising,<sup>5</sup> if  $m \models c$  then  $H'$  is promising.

A set  $\mathcal{R} = \{t_{p\nu}[c_{p\nu}] \mid t_{p\nu} \in \mathcal{R}_{\tilde{\Gamma}}^E\}$  of conditional rules is a set of *predictive expand rules* for  $\tilde{\Gamma}$  if  $c_{p\nu}$  is a lookahead condition for every  $t_{p\nu} \in \mathcal{R}_{\tilde{\Gamma}}^E$ .

In the following, we briefly describe a simple way to check whether a set of predictive expand rules can be obtained from  $\mathcal{R}_{\tilde{\Gamma}}^E$ . For this purpose, let  $G$  be any initial configuration and  $t_{p\nu}: P \multimap R$  any expand rule so that  $G \Rightarrow_{\mathcal{R}_{\tilde{\Gamma}}}^* H \Rightarrow_{t_{p\nu}}^m H'$  where  $H'$  is promising, i.e., there is an accepting configuration  $F$  such that

---

<sup>5</sup> From now on, we call a configuration promising if it is in fact promising with respect to  $\mathcal{R}_{\tilde{\Gamma}}$ .

$$\text{either } H \Rightarrow_{t_{p^\nu}}^m H' \Rightarrow_{\mathcal{R}_{\tilde{F}}^E}^* K \Rightarrow_{\mathcal{R}_{\tilde{F}}^M} K' \Rightarrow_{\mathcal{R}_{\tilde{F}}}^* F \tag{1}$$

$$\text{or } H \Rightarrow_{t_{p^\nu}}^m H' \Rightarrow_{\mathcal{R}_{\tilde{F}}}^* F \tag{2}$$

Consider case (1) first. There is an isomorphism  $iso : unread(K) \rightarrow unread(H)$  because  $K$  is obtained from  $H$  by expand rules only. Let  $e$  be the edge of  $unread(K)$  that is read by the match operation  $K \Rightarrow_{\mathcal{R}_{\tilde{F}}^M} K'$  and  $E$  the subgraph of  $K$  induced by  $e$ . Clearly,  $m(P)$  as well as  $iso(E)$  are both subgraphs of  $H$ . Now select a graph  $C$  and an injective morphism  $m'$  so that  $P \subseteq C$ ,  $m = m'|_P$ , and  $m'(C) = m(P) \cup iso(E)$ . By definition,  $m \models \exists C$ . In case (2),  $unread(H)$  is empty and  $m \models \exists P$ .

We can make use of this as follows. For an expand rule  $t_{p^\nu}$ , performing the above analysis for all derivations of types (1) and (2) yields only finitely many distinct graphs  $C$  (up to isomorphism). These graphs  $C_1, \dots, C_n$  can be computed by procedures similar to the construction of FIRST and FOLLOW sets for LL( $k$ ) parsing [15, Sect. 5.5]. Defining  $\hat{c}_{p^\nu} = \exists C_1 \vee \exists C_2 \vee \dots \vee \exists C_n$  we thus obtain for all promising graphs  $H, H'$  that  $H \Rightarrow_{t_{p^\nu}}^m H'$  implies  $m \models \hat{c}_{p^\nu}$ . Thus, by contraposition, if  $H$  is promising and  $H \Rightarrow_{t_{p^\nu}}^m H'$  but  $m \not\models \hat{c}_{p^\nu}$ , then  $H'$  cannot be promising.

Note, however, that  $m \models \hat{c}_{p^\nu}$  does not necessarily imply that  $H'$  is promising if  $H \Rightarrow_{t_{p^\nu}}^m H'$  and  $H$  is promising. Therefore,  $\hat{c}_{p^\nu}$  can in general not directly serve as a lookahead condition. To solve this problem, we define a relation  $\sqsubset$  on expand rules. For this purpose, let us consider two different expand rules  $t_{p_a^\nu}, t_{p_b^\nu} \in \mathcal{R}_{\tilde{F}}^E$  with isomorphic left-hand sides. Without loss of generality, we assume that the left-hand sides are identical. We define  $t_{p_a^\nu} \sqsubset t_{p_b^\nu}$  if there is an initial configuration  $G$  and a derivation  $G \Rightarrow_{\mathcal{R}_{\tilde{F}}}^* H \Rightarrow_{t_{p_a^\nu}}^m H'$  where  $H'$  is promising and  $m \models \hat{c}_{p_b^\nu}$ . In fact, relation  $\sqsubset$  can be defined while conditions  $\hat{c}_{p_i^\nu}$  are constructed.<sup>6</sup>

Note that  $\sqsubset$  is in general not an ordering and that it may even contain cycles  $t_{p_a^\nu} \sqsubset t_{p_b^\nu} \sqsubset \dots \sqsubset t_{p_a^\nu}$ . But if there are no such cycles, one can create (by topological sorting) a linear ordering  $<$  on all expand rules with isomorphic left-hand sides (where we again assume that they have in fact identical left-hand sides) so that  $t_{p_a^\nu} \sqsubset t_{p_b^\nu}$  always implies  $t_{p_a^\nu} < t_{p_b^\nu}$ . We then define, for each expand rule  $t_{p^\nu}$ , the condition  $c_{p^\nu} \equiv \hat{c}_{p^\nu} \wedge \neg c_1 \wedge \neg c_2 \wedge \dots \wedge \neg c_n$  where  $\{c_1, c_2, \dots, c_n\} = \{\hat{c}_{\tilde{p}^\nu} \mid t_{\tilde{p}^\nu} < t_{p^\nu}\}$ . The following lemma states that these conditions can serve as lookahead conditions for predictive expand rules:

**Lemma 2.** *Let  $\Gamma$  be a HR grammar,  $\tilde{\Gamma}$  a threaded version of  $\Gamma$ , and  $\mathcal{R}_{\tilde{F}} = \mathcal{R}_{\tilde{F}}^M \cup \mathcal{R}_{\tilde{F}}^E$  its general top-down parser. If  $\sqsubset$  is acyclic and the condition  $c_{p^\nu}$  is defined as above for each expand rule  $t_{p^\nu} \in \mathcal{R}_{\tilde{F}}^E$ , then  $\{t_{p^\nu}[c_{p^\nu}] \mid t_{p^\nu} \in \mathcal{R}_{\tilde{F}}^E\}$  is a set of predictive expand rules for  $\tilde{\Gamma}$ .*

<sup>6</sup>  $\hat{c}_{p_i^\nu}$  identifies edges that must occur in  $H$  if  $H \Rightarrow_{t_{p_b^\nu}}^m H''$  where  $H''$  is promising. And if these edges may also occur in  $H$  if  $H'$  is promising, we define  $t_{p_a^\nu} \sqsubset t_{p_b^\nu}$ .

*Proof.* Consider any derivation  $G \Rightarrow_{\mathcal{R}_{\tilde{F}}}^* H \Rightarrow_{\mathcal{R}_{\tilde{F}}^E} H'$  where  $G$  is an initial configuration, and  $H$  is promising. Then there is an expand rule  $t_{p^\nu}$  so that  $H \Rightarrow_{t_{p^\nu}}^m K$  and  $K$  is promising. By construction,  $m \models \hat{c}_{p^\nu}$ . If there were a smaller expand rule  $t_{\bar{p}^\nu} \prec t_{p^\nu}$  with  $m \models \hat{c}_{\bar{p}^\nu}$ , then this would imply  $t_{p^\nu} \sqsubset t_{\bar{p}^\nu}$  because  $K$  is promising, and therefore,  $t_{p^\nu} \prec t_{\bar{p}^\nu}$ , contradicting the linearity of  $\prec$ . Therefore,  $m \models \neg \hat{c}_{\bar{p}^\nu}$  for  $t_{\bar{p}^\nu} \prec t_{p^\nu}$  and  $m \models \hat{c}_{p^\nu}$ , i.e.,  $t_{p^\nu}$  is the only expand rule that satisfies its lookahead condition for  $H$ , i.e.,  $m \models c_{p^\nu}$ .  $\square$

The proof shows that these lookahead conditions always select a unique expand rule. Clearly, this cannot succeed for situations where expand rules can turn a promising configuration into two or more promising successor configurations.

However, the existence of a set of predictive expand rules is not sufficient for obtaining a predictive top-down parser. The threaded HR grammar must satisfy the following property as well:

**Definition 13 (Free edge choice property).** Let  $\Gamma$  be a HR grammar,  $\tilde{\Gamma}$  a threaded version of  $\Gamma$ , and  $\mathcal{R}_{\tilde{\Gamma}} = \mathcal{R}_{\tilde{\Gamma}}^M \cup \mathcal{R}_{\tilde{\Gamma}}^E$  its general top-down parser.  $\tilde{\Gamma}$  is said to possess the *free edge choice property* if, for every derivation  $G \Rightarrow_{\mathcal{R}_{\tilde{\Gamma}}}^* H \Rightarrow_{\mathcal{R}_{\tilde{\Gamma}}^M} H'$  where  $G$  is an initial configuration and  $H$  is promising,  $H'$  is promising as well.

**Theorem 4.** Let  $\Gamma$  be a HR grammar,  $\tilde{\Gamma}$  a threaded version of  $\Gamma$  without left-recursion, and  $\mathcal{R}_{\tilde{\Gamma}} = \mathcal{R}_{\tilde{\Gamma}}^M \cup \mathcal{R}_{\tilde{\Gamma}}^E$  its general top-down parser.  $\mathcal{R}^{\text{ptd}} = \mathcal{R}_{\tilde{\Gamma}}^M \cup \mathcal{R}$  is a terminating top-down parser for  $\Gamma$  that cannot run into blind alleys if  $\mathcal{R}$  is a set of predictive expand rules for  $\tilde{\Gamma}$  and  $\tilde{\Gamma}$  has the free edge choice property.

*Proof.* Let  $\Gamma$ ,  $\tilde{\Gamma}$ , and  $\mathcal{R}^{\text{ptd}}$  be as in the theorem. Moreover, let  $\tilde{\Gamma}$  satisfy the free edge choice property, and let  $\mathcal{R}$  be a set of predictive expand rules for  $\tilde{\Gamma}$ . Each derivation  $G \Rightarrow_{\mathcal{R}^{\text{ptd}}}^* H$  where  $G$  and  $H$  are initial and accepting configurations, resp., is also a successful parse in  $\mathcal{R}_{\tilde{\Gamma}}$ , i.e.,  $\text{unread}(G) \in \mathcal{L}(\Gamma)$  by Theorem 2.

Now let  $G$  be any initial configuration with  $\text{unread}(G) \in \mathcal{L}(\Gamma)$ , i.e.,  $G$  is promising. Any infinite derivation  $G \Rightarrow_{\mathcal{R}^{\text{ptd}}} H_1 \Rightarrow_{\mathcal{R}^{\text{ptd}}} H_2 \Rightarrow_{\mathcal{R}^{\text{ptd}}} \dots$  would also be an infinite parse  $G \Rightarrow_{\mathcal{R}_{\tilde{\Gamma}}} H_1 \Rightarrow_{\mathcal{R}_{\tilde{\Gamma}}} H_2 \Rightarrow_{\mathcal{R}_{\tilde{\Gamma}}} \dots$ , contradicting Theorem 3.

Finally assume that  $\mathcal{R}^{\text{ptd}}$  runs into a blind alley starting at  $G$ , i.e., there is a derivation  $G \Rightarrow_{\mathcal{R}^{\text{ptd}}}^* H$ ,  $H$  is not accepting, and there is no configuration  $H'$  so that  $H \Rightarrow_{\mathcal{R}^{\text{ptd}}} H'$ . By the free edge choice property and  $\mathcal{R}$  being a set of predictive expand rules,  $H$  must be promising, i.e., there is a configuration  $H''$  so that  $H \Rightarrow_{\mathcal{R}_{\tilde{\Gamma}}^M} H''$  or  $H \Rightarrow_{\mathcal{R}_{\tilde{\Gamma}}^E} H''$ . In either case, there is a configuration  $H'$  so that  $H \Rightarrow_{\mathcal{R}^{\text{ptd}}} H'$ , contradicting the assumption.  $\square$

This theorem justifies to call a threaded HR grammar  $\tilde{\Gamma}$  *predictively top-down parsable* (PTD for short) if  $\tilde{\Gamma}$  satisfies the free edge choice property and there is a set of predictive expand rules for  $\tilde{\Gamma}$ .

*Example 4 (A Predictive Top-Down Tree Parser).* The threaded tree grammar in Example 2 is PTD. To see this, let us construct lookahead conditions for expand rule  $t_{\bar{b}^1}$  and  $t_{\bar{t}^1}$  as described above.

Inspection of expand rule  $t_{\bar{B}^1}$  shows that choosing this rule cannot produce a promising configuration if the unread part of the input does not contain a  $\triangleright$ -edge starting at node  $x$ . The existence of this edge is hence requested by the graph condition  $\hat{c}_{\bar{B}^1} \equiv \exists C_{\bar{B}^1}$ , defined by the supergraph  $C_{\bar{B}^1}$  of the pattern of  $t_{\bar{B}^1}$  (see Fig. 8). No such edge can be requested for expand rule  $t_{\bar{L}^1}$ ; each match of  $t_{\bar{L}^1}$  satisfies  $\hat{c}_{\bar{L}^1} \equiv \exists C_{\bar{L}^1}$  since  $C_{\bar{L}^1}$  is just the pattern of  $t_{\bar{L}^1}$ . Condition  $\hat{c}_{\bar{L}^1}$  is in particular satisfied if choosing  $t_{\bar{B}^1}$  produces a promising configuration, and therefore  $t_{\bar{B}^1} \sqsubset t_{\bar{L}^1}$ . By Lemma 2, we can choose lookahead conditions  $c_{\bar{B}^1} \equiv \hat{c}_{\bar{B}^1} \equiv \exists C_{\bar{B}^1}$  and  $c_{\bar{L}^1} \equiv \hat{c}_{\bar{L}^1} \wedge \neg c_{\bar{B}^1} \equiv \neg \exists C_{\bar{B}^1}$ .

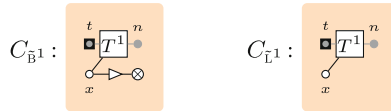


Fig. 8. Graphs defining  $\hat{c}_{\bar{B}^1} \equiv \exists C_{\bar{B}^1}$  and  $\hat{c}_{\bar{L}^1} \equiv \exists C_{\bar{L}^1}$  for expand rule  $t_{\bar{B}^1}$  and  $t_{\bar{L}^1}$ , resp.

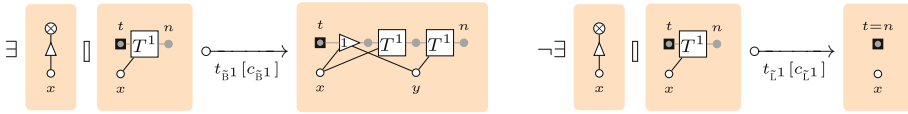


Fig. 9. Predictive expand operations of the tree parser

Figure 9 shows the resulting predictive expand rules for the nonterminal  $T$  of the tree parser. For brevity, lookahead conditions show only those subgraphs that must or must not exist in order to apply  $t_{\bar{B}^1}$  or  $t_{\bar{L}^1}$ . The match rules and the expand rule  $t_{\bar{S}^\varepsilon}$  for the start production remain the same as in Example 3. Moreover, it is easy to see that match rule  $t_{\bar{>}^1}$  produces a promising configuration for each of its matches, i.e., the threaded tree grammar has the free edge choice property. With these modified expand rules, the predictive parser can select the same parse as in Fig. 7. As mentioned earlier, other well-known examples that allow for predictive parsing include palindromes,  $a^n b^n c^n$ , arithmetic expressions, and Nassi-Shneiderman diagrams.

## 7 Conclusions

In this paper, we have defined PTD parsers for HR grammars by graph transformation rules, and shown their correctness. The definition is consistent with the implementation of PTD parsers in the *graph parser distiller* GRAPPA<sup>7</sup> described

<sup>7</sup> Available at [www.unibw.de/inf2/grappa](http://www.unibw.de/inf2/grappa). GRAPPA also distills PSR and generalized PSR parsers for CHR grammars [5, 13].



in [4], but some features are still missing: First, productions that *merge* nodes of the left-hand side have been omitted. Such productions may occur when a HR grammar is “left-factorized” in order to allow for predictive expansion. (This corresponds to left-factorization of CF string grammars for LL-parsing.) Second, PTD parsing for *contextual* HR grammars [2, 3] has not been considered. Finally, a more sophisticated way of calculating lookahead conditions, by approximating Parikh images, has been ignored.

So our next step will be to extend our definitions and proofs to cover these concepts as well. Our ultimate goal is to use this definition to relate the power of PTD parsing to that of PSR parsing, probably by using a definition of PSR parsing that is based on graph transformation as well.

**Acknowledgements.** The authors thank Annegret Habel for her valuable suggestions in several stages of this work.

## References

1. Aalbersberg, I., Ehrenfeucht, A., Rozenberg, G.: On the membership problem for regular DNLC grammars. *Discrete Appl. Math.* **13**, 79–85 (1986)
2. Drewes, F., Hoffmann, B.: Contextual hyperedge replacement. *Acta Informatica* **52**(6), 497–524 (2015). <https://doi.org/10.1007/s00236-015-0223-4>
3. 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](https://doi.org/10.1007/978-3-642-34176-2_16)
4. 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](https://doi.org/10.1007/978-3-319-21145-9_2)
5. Drewes, F., Hoffmann, B., Minas, M.: Extending predictive shift-reduce parsing to contextual hyperedge replacement grammars. In: Guerra, E., Orejas, F. (eds.) *ICGT 2019*. LNCS, vol. 11629, pp. 55–72. Springer, Cham (2019). [https://doi.org/10.1007/978-3-030-23611-3\\_4](https://doi.org/10.1007/978-3-030-23611-3_4)
6. Drewes, F., Hoffmann, B., Minas, M.: Formalization and correctness of predictive shift-reduce parsers for graph grammars based on hyperedge replacement. *J. Local Algebraic Methods Programm. (JLAMP)* **104**, 303–341 (2019). <https://doi.org/10.1016/j.jlamp.2018.12.006>
7. Ehrig, H., Ehrig, K., Prange, U., Taentzer, G.: Implementation of typed attributed graph transformation by AGG. In: *Fundamentals of Algebraic Graph Transformation*. An EATCS Series, pp. 305–323. Springer, Heidelberg (2006). [https://doi.org/10.1007/3-540-31188-2\\_15](https://doi.org/10.1007/3-540-31188-2_15)
8. Habel, A.: *Hyperedge Replacement: Grammars and Languages*. LNCS, vol. 643. Springer, Heidelberg (1992). <https://doi.org/10.1007/BFb0013875>
9. Habel, A., Müller, J., Plump, D.: Double-pushout graph transformation revisited. *Math. Struct. Comput. Sci.* **11**(5), 633–688 (2001)
10. Habel, A., Pennemann, K.H.: Correctness of high-level transformation systems relative to nested conditions. *Math. Struct. Comput. Sci.* **19**(2), 245–296 (2009)
11. Heilbrunner, S.: A parsing automata approach to LR theory. *Theor. Comput. Sci.* **15**, 117–157 (1981). [https://doi.org/10.1016/0304-3975\(81\)90067-0](https://doi.org/10.1016/0304-3975(81)90067-0)

12. Hoffmann, B.: Modelling compiler generation by graph grammars. In: Ehrig, H., Nagl, M., Rozenberg, G. (eds.) Graph Grammars 1982. LNCS, vol. 153, pp. 159–171. Springer, Heidelberg (1983). <https://doi.org/10.1007/BFb0000105>
13. 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](https://doi.org/10.1007/978-3-030-13435-8_17)
14. Lange, K.J., Welzl, E.: String grammars with disconnecting or a basic root of the difficulty in graph grammar parsing. *Discrete Appl. Math.* **16**, 17–30 (1987)
15. Sippu, S., Soisalon-Soininen, E.: Parsing Theory I: Languages and Parsing. EATCS Monographs in Theoretical Computer Science, vol. 15. Springer, Heidelberg (1988). <https://doi.org/10.1007/978-3-642-61345-6>