Parsing with Graph Grammars Based on Conditional Borrowing Hyperedge Replacement

Frank Drewes[1,†], Berthold Hoffmann[2,†] and Mark Minas[3,*,†]

[1]Institutionen för datavetenskap, Umeå universitet, SE-90187 Umeå, Sweden
[2]Fachbereich 3 – Informatik, Universität Bremen, D-28334 Bremen, Germany
[3]Institut für Softwaretechnologie, Universität der Bundeswehr München, D-85577 Neubiberg, Germany

Abstract

In graph grammars based on conditional borrowing hyperedge replacement (cBHR), productions replace a hyperedge by a hypergraph that may contain borrowed nodes and condition edges. After the context-free derivation is completed, borrowed nodes are contracted with nodes of the derived graph, while condition edges are used to enforce constraints on the result. In this paper, we introduce a nondeterministic top-down parser for cBHR grammars and formally prove its correctness. This provides a foundational step toward efficient parsing techniques for grammar classes that extend hyperedge replacement beyond bounded treewidth.

Keywords

graph grammar, hyperedge replacement, conditional borrowing, parsing

1. Introduction

The hyperedge replacement (HR) grammar is probably the most successful type of graph grammar, enjoying many of the nice properties of context-free string grammars [1]. Unfortunately, HR grammars have two disadvantages: on the one hand, their languages can be NP-complete, meaning that there is little hope that parsing could be implemented efficiently. On the other hand, HR languages are of bounded treewidth, which is a severe restriction in many practical contexts. Thus, HR grammars appear to be both too powerful and too weak.

To address the lack of efficient parsing algorithms, one can try to find subclasses which enable polynomial parsing. A lot of research in this direction can be found in the literature. For the current line of work, two examples of such subclasses are the predictive top-down (PTD) parsable and the predictive shift-reduce (PSR) parsable HR grammars proposed and studied in [2, 3, 4, 5].

The restriction to bounded treewidth can be overcome by carefully adding capabilities in such a way that the efficient parsing techniques (and the restrictions they rely on) generalize to the resulting grammar classes. An example of the latter are the recently proposed conditional borrowing hyperedge replacement (cBHR) grammars [6], which in turn extend the borrowing hyperedge replacement (BHR) grammars of [7].

The productions of BHR grammars feature so-called borrowed nodes in their right-hand sides. Once the entirely context-free derivation of a graph is finished (i.e., no nonterminals are left), all borrowed nodes are fused with non-borrowed nodes, yielding the generated graph. This *contraction* step enables BHR grammars to generate graph languages of unbounded treewidth.

In the present paper, we formalize top-down parsing for cBHR grammars and prove its correctness as a step towards PTD parsing for these grammars. The parser we propose here is nondeterministic

Joint Proceedings of the STAF 2026 Workshops: AgileMDE, GCM, ICMM, LLM4SE, TTC. Rennes, France, June 29-July 3, 2026
*Corresponding author.
†These authors contributed equally.
drewes@cs.umu.se (F. Drewes); hof@uni-bremen.de (B. Hoffmann); mark.minas@unibw.de (M. Minas)
https://www.umu.se/personal/frank-drewes/ (F. Drewes); https://user.informatik.uni-bremen.de/hof/ (B. Hoffmann); https://www.unibw.de/mark.minas (M. Minas)
0000-0001-7349-7693 (F. Drewes); 0000-0002-5608-996X (B. Hoffmann); 0000-0002-8968-9013 (M. Minas)
© 2026 Copyright © 2026 for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0)

CEUR Workshop Proceedings ceur-ws.org ISSN 1613-0073
published 2026-11-26

CEUR-WS.org/Vol-4338/GCM-paper6.pdf

and thus still inefficient as its deterministic implementation requires backtracking, but it is a necessary prerequisite for efficient PTD parsers.

The paper is structured as follows. The next section compiles the basic notions around graphs and graph expressions. Sect. 3 recalls borrowing hyperedge replacement and its extension to conditional borrowing hyperedge replacement from [6]. Sect. 4 contains an example (statecharts) for illustration of cBHR grammars and the parser, which is developed in Sect. 5. Sect. 6 concludes the paper by discussing in broad strokes what is necessary to make the parser deterministic, in order to extend predictive top-down parsing to cBHR grammars.

2. Preliminaries

We let $\mathbb{N}$ denote the set of non-negative integers and $[n]$ the set $\{1, \dots, n\}$ for all $n \in \mathbb{N}$. A^* denotes the set of all finite sequences over a set A; the empty sequence is denoted by ε. For a sequence $s \in A^*$, $|s|$ denotes the length of s, and $[s]$ denotes the set of all members of A occurring in s. If $f\colon A \to B$ is a function, $f^*\colon A^* \to B^*$ denotes the function defined by $f^*(a_1 \cdots a_n) = f(a_1) \cdots f(a_n)$, and $f^{-1}(b) = \{a \in A \mid f(a) = b\}$ for every $b \in B$. For two functions $f\colon A \to B$ and $g\colon B \to C$, $g \circ f$ defines the function $g \circ f\colon A \to C$ defined by $(g \circ f)(x) = g(f(x))$. As usual, for a binary relation $\rightsquigarrow \subseteq A \times A$, we let $\rightsquigarrow^n$ ($n \in \mathbb{N}$), $\rightsquigarrow^+$, and $\rightsquigarrow^*$ denote the n-fold composition with itself, its transitive closure, and its reflexive-transitive closure, respectively.

2.1. Graphs and Graph Concatenation

We consider edge-labeled hypergraphs (which we simply call graphs), i.e., edges are attached to sequences of nodes. To be able to compose graphs in the way originally proposed by Engelfriet and Vereijken [8], we supply each graph with two sequences of distinguished nodes, its front and rear interfaces.[1]

A ranked set (Σ, rank) consists of a finite set Σ of elements and a function $\mathit{rank}\colon \Sigma \to \mathbb{N}$, which assigns a rank to each element $a \in \Sigma$. The pair (Σ, rank) is usually identified with Σ, keeping rank implicit.

Definition 2.1 (Graph)**.** Let Σ be a ranked set of symbols. A *graph* over Σ is a tuple $G = (\dot{G}, \bar{G}, \mathit{att}_G, \mathit{lab}_G, \mathit{front}_G, \mathit{rear}_G)$, where $\dot{G}$ and $\bar{G}$ are disjoint finite sets of *nodes* and *edges*, respectively, $\mathit{att}_G\colon \bar{G} \to \dot{G}^*$ attaches sequences of nodes to edges, $\mathit{lab}_G\colon \bar{G} \to \Sigma$ labels edges with symbols so that $|\mathit{att}_G(e)| = \mathit{rank}(\mathit{lab}_G(e))$ for every edge $e \in \bar{G}$, and the node sequences $\mathit{front}_G, \mathit{rear}_G \in \dot{G}^*$ are the *front* and *rear* interface, respectively, of G. The graph Id_k with $\bar{Id}_k = \varnothing$, $\dot{Id}_k = [k]$, and $\mathit{front}_{\mathit{Id}_k} = 1 \cdots k = \mathit{rear}_{\mathit{Id}_k}$ is the *identity graph (on k nodes)*.

An a-edge is an edge labeled by the symbol a. Given a subset Δ of Σ, we denote by $G|_\Delta$ the graph obtained from G by deleting all a-edges with $a \notin \Delta$.

For a graph G and $\alpha \in \dot{G}^*$, we let $\alpha : G = (\dot{G}, \bar{G}, \mathit{att}_G, \mathit{lab}_G, \alpha, \mathit{rear}_G)$ and $G : \alpha = (\dot{G}, \bar{G}, \mathit{att}_G, \mathit{lab}_G, \mathit{front}_G, \alpha)$ denote the graphs where front_G and rear_G are set to α, respectively.

The *type* of a graph G is $(|\mathit{front}_G|, |\mathit{rear}_G|)$, $\mathbb{G}_\Sigma^{(m,n)}$ denotes the set of all graphs of type (m, n), and $\mathbb{G}_\Sigma = \bigcup_{m,n \in \mathbb{N}} \mathbb{G}_\Sigma^{(m,n)}$. A subset of $\mathbb{G}_\Sigma^{(m,n)}$ is a *graph language* (of type (m, n)).

Given graphs $G, H \in \mathbb{G}_\Sigma$, a (graph) *morphism* $m\colon G \to H$ is a pair $(\dot{m}, \bar{m})$ of functions $\dot{m}\colon \dot{G} \to \dot{H}$ and $\bar{m}\colon \bar{G} \to \bar{H}$ such that $\mathit{lab}_G = \mathit{lab}_H \circ \bar{m}$, $\mathit{att}_H \circ \bar{m} = \dot{m}^* \circ \mathit{att}_G$, $\mathit{front}_H = \dot{m}^*(\mathit{front}_G)$, and $\mathit{rear}_H = \dot{m}^*(\mathit{rear}_G)$; m is called *injective* or *surjective*, respectively, if both $\dot{m}$ and $\bar{m}$ are, and it is called an *isomorphism* if $\dot{m}$ and $\bar{m}$ are both bijective. Two graphs $G, H \in \mathbb{G}_\Sigma$ are called *isomorphic*, written $G \cong H$, if there is an isomorphism $m\colon G \to H$. The set $[G]_\cong = \{H \mid G \cong H\}$ is called the *isomorphism class* of a graph $G \in \mathbb{G}_\Sigma$.

[1] In contrast to [1], however, we do not also divide the attached nodes of edges into sources and targets.

Assumption 2.2. In the following, unless stated otherwise, all graphs are understood as *abstract* graphs, i.e., considered up to isomorphism. More precisely, any reference to a graph G should be interpreted as referring to its isomorphism class $[G]_{\cong}$.

Definition 2.3 (Sequential Graph Composition [8])**.** Let $G \in \mathbb{G}_{\Sigma}^{(i,k)}$ and $H \in \mathbb{G}_{\Sigma}^{(k,j)}$. We assume for simplicity that $rear_G = front_H$, $\dot{G} \cap \dot{H} = [rear_G] = [front_H]$, and $\bar{G} \cap \bar{H} = \varnothing$ (otherwise an appropriate isomorphic copy of G or H is used). The (*sequential*) *composition* $G \cdot H$ of G and H is the graph C such that $\dot{C} = \dot{G} \cup \dot{H}$, $\bar{C} = \bar{G} \cup \bar{H}$, $att_C = att_G \cup att_H$, $lab_C = lab_G \cup lab_H$, $front_C = front_G$, and $rear_C = rear_H$. Thus, $C \in \mathbb{G}_{\Sigma}^{(i,j)}$.

Given a sequence $G_1, \ldots, G_n$ of $n \geq 1$ graphs such that $|rear_{G_i}| = |front_{G_{i+1}}|$ for every $i \in [n-1]$, the sequential composition $G_1 \cdot \cdots \cdot G_n$ is written as $\mathrm{comp}(G_1 \cdots G_n)$.

Note that the concatenation $G \cdot H$ is again an abstract graph, i.e., defined up to isomorphism.

We recall frontal subgraphs from [9].

Definition 2.4 (Frontal Subgraph)**.** Let $H \in \mathbb{G}_{\Sigma}^{(i,j)}$. A graph $G \in \mathbb{G}_{\Sigma}^{(i,k)}$ with $\dot{G} \subseteq \dot{H}$ and $\bar{G} \subseteq \bar{H}$ is a *frontal subgraph* of H if

- $lab_G(e) = lab_H(e)$ and $att_G(e) = att_H(e)$ for all $e \in \bar{G}$, and
- $front_G = front_H$.

2.2. Graph Expressions

We now recall the definition of graph expressions from [9]. Similarly to regular expressions for string languages, graph expressions construct graph languages from singleton languages with the operations union, sequential composition, and Kleene star on graphs. In graph expressions, we will use the notation known from regular expressions for these operations.

Definition 2.5 (Graph Expressions)**.** Let $i, j, k \in \mathbb{N}$.

- The *sequential composition*, or *concatenation* of graph languages $\mathcal{L} \subseteq \mathbb{G}_{\Sigma}^{(i,k)}$ and $\mathcal{M} \subseteq \mathbb{G}_{\Sigma}^{(k,j)}$ is $\mathcal{L} \cdot \mathcal{M} = \{G \cdot H \mid G \in \mathcal{L}, H \in \mathcal{M}\}$.
- The (*Kleene*) *star* $\mathcal{L}^*$ of a graph language $\mathcal{L} \subseteq \mathbb{G}_{\Sigma}^{(i,i)}$ is the smallest graph language containing Id_i and, for all graphs $G \in \mathcal{L}$ and $H \in \mathcal{L}^*$, the graph $G \cdot H \in \mathcal{L}^*$.

Note that concatenation is associative. For non-empty $\mathcal{L} \subseteq \mathbb{G}_{\Sigma}^{(i,i)}$, we abbreviate the n-fold iterated concatenation of $\mathcal{L}$ with itself by $\mathcal{L}^n$, i.e., $\mathcal{L}^0 = \{Id_i\}$ and $\mathcal{L}^{n+1} = \mathcal{L} \cdot \mathcal{L}^n$ for $n \in \mathbb{N}$.

The set $\mathbb{E}_{\Sigma}^{(i,j)}$ of *graph expressions* ex of type (i, j) over Σ and the languages $\mathcal{L}(ex)$ they denote are defined inductively, as follows:

1. $\varnothing \in \mathbb{E}_{\Sigma}^{(i,j)}$ with $\mathcal{L}(\varnothing) = \varnothing$.
2. If $G \in \mathbb{G}_{\Sigma}^{(i,j)}$, then $G \in \mathbb{E}_{\Sigma}^{(i,j)}$ with $\mathcal{L}(G) = G$.
3. If $ex_1, ex_2 \in \mathbb{E}_{\Sigma}^{(i,j)}$, then $ex_1 | ex_2 \in \mathbb{E}_{\Sigma}^{(i,j)}$ with $\mathcal{L}(ex_1 | ex_2) = \mathcal{L}(ex_1) \cup \mathcal{L}(ex_2)$.
4. If $ex_1 \in \mathbb{E}^{(i,j)}$ and $ex_2 \in \mathbb{E}_{\Sigma}^{(j,k)}$, then $ex_1 ex_2 \in \mathbb{E}_{\Sigma}^{(i,k)}$ with $\mathcal{L}(ex_1 ex_2) = \mathcal{L}(ex_1) \cdot \mathcal{L}(ex_2)$.
5. If $ex \in \mathbb{E}_{\Sigma}^{(i,i)}$, then $ex^* \in \mathbb{E}_{\Sigma}^{(i,i)}$ with $\mathcal{L}(ex^*) = \mathcal{L}(ex)^*$.

We let $\hat{\mathbb{E}}_{\Sigma}^{(i,j)}$ be the subset of $\mathbb{E}_{\Sigma}^{(i,j)}$ obtained by considering, in case 2 above, only graphs $G \in \mathbb{G}_{\Sigma}^{(i,j)}$ such that $front_G$ is free of repetitions. Hence, the graph languages $\mathcal{L}(ex)$ for $ex \in \hat{\mathbb{E}}_{\Sigma}^{(i,j)}$ contain only graphs whose front interfaces are repetition-free.

To save parentheses, we use the convention that the Kleene star binds stronger than concatenation, which binds stronger than union.

3. Conditional Borrowing Hyperedge Replacement Grammars

The definition of conditional borrowing hyperedge replacement uses labeling alphabets which do not only contain the ranked alphabet Σ, but also a ranked alphabet $X = N \cup \{\square\} \cup C$ such that Σ, N, $\{\square\}$, and C are pairwise disjoint. The additional labels serve the following purposes:

- *nonterminal names* in N indicate places for hyperedge replacement;
- the *borrow symbol* $\square$ (of rank 1) designates the attached node of a $\square$-edge as a *borrowed node*, to be contracted with another node later on;
- *condition names* in C indicate places where conditions must be satisfied.

Edges labeled with these symbols are accordingly called nonterminal, borrow, and condition edges, respectively.

Given $Y \subseteq X$, we write $\mathbb{G}_\Sigma(Y)$ and $\mathbb{G}_\Sigma(Y)^{(m,n)}$ for the subset of those graphs in $\mathbb{G}_{\Sigma\cup Y}$ and $\mathbb{G}^{(m,n)}_{\Sigma\cup Y}$, respectively, which do not have any borrowed nodes in their front interface. Only such graphs will be considered in the following. Similarly, we write $\mathbb{E}_\Sigma(Y)$ and $\hat{\mathbb{E}}_\Sigma(Y)$ for the sets of expressions in $\mathbb{E}_{\Sigma\cup Y}$ and $\hat{\mathbb{E}}_{\Sigma\cup Y}$, respectively, in which only such graphs occur.

Conceptually, the way conditions are used in conditional borrowing hyperedge replacement is orthogonal to the generation process itself: any formalism specifying graph properties can be "plugged in". However, to be of practical use, such a formalism should be should expressive enough to specify non-trivial conditions yet allow for a reasonably efficient implementation. We thus follow [6], using a version of the graph formulas of [9] without nested quantification to avoid PSPACE completeness of condition testing.

Definition 3.1 (Shallow Graph Formulas)**.** For $m \in \mathbb{N}$ and a ranked set Σ of edge labels, the set $\mathbb{SF}^m_\Sigma$ of *shallow graph formulas* of type m over Σ (simply called *formulas* in the following) is defined inductively as follows:

(1) $\mathit{true}, \mathit{false} \in \mathbb{SF}^m_\Sigma$;

(2) if $ex \in \mathbb{E}^{(m,i)}_\Sigma$ for any $i \in \mathbb{N}$, then $\exists(ex) \in \mathbb{SF}^m_\Sigma$;

(3) if $\mathit{fo}, \mathit{fo}' \in \mathbb{SF}^m_\Sigma$, then $\neg \mathit{fo}$, $\mathit{fo} \vee \mathit{fo}'$, and $\mathit{fo} \wedge \mathit{fo}'$ are in $\mathbb{SF}^m_\Sigma$.

We let $\mathbb{SF}_\Sigma = \bigcup_{m\in\mathbb{N}} \mathbb{SF}^m_\Sigma$ denote the set of all formulas over Σ.

A graph $G \in \mathbb{G}^{(m,n)}_\Sigma$ *satisfies* $\mathit{fo} \in \mathbb{SF}^m_\Sigma$, written $G \vDash \mathit{fo}$, if one of the following holds:

- $\mathit{fo} = \exists ex$ such that some graph $H \in \mathcal{L}(ex)$ is a frontal subgraph of G;
- $\mathit{fo} = \neg \mathit{fo}'$ such that $G \vDash \mathit{fo}'$ does not hold;
- $\mathit{fo} = \mathit{fo}' \vee \mathit{fo}''$ such that $G \vDash \mathit{fo}'$ or $G \vDash \mathit{fo}''$ (or both);
- $\mathit{fo} = \mathit{fo}' \wedge \mathit{fo}''$ such that $G \vDash \mathit{fo}'$ and $G \vDash \mathit{fo}''$.

We now recall the definition of *conditional borrowing hyperedge replacement grammars* (conditional BHR grammars) from [6].

Definition 3.2 (Conditional Borrowing Hyperedge Replacement)**.** A *conditional borrowing hyperedge replacement grammar* (*conditional BHR grammar*, or *cBHR grammar* for short) is a quintuple $\Gamma = (\Sigma, X, \pi, S, \varphi)$, where Σ and X are as above, such that

1. π is a *production mapping* $\pi\colon N \to \hat{\mathbb{E}}_\Sigma(N \cup \{\square\})$ such that, for every $A \in N$, the *right-hand side* $\pi(A)$ is of type $(\mathit{rank}(A), n)$ for some $n \in \mathbb{N}$,[2]
2. $S \in N$ is the *start symbol*, and
3. the *condition assignment* φ is a function $\varphi\colon C \to \mathbb{SF}_\Sigma$ such that $\varphi(c) \in \mathbb{SF}^{\mathit{rank}(c)}_\Sigma$ for every condition name $c \in C$.

[2]The rear of $\pi(A)$ is irrelevant; this liberty makes productions easier to write.

Consider a graph G with a nonterminal A-edge $e \in \bar{G}$. *Replacing* e by $R \in \mathcal{L}(\pi(A))$ yields $H = ((G - e) : att_G(e)) \cdot (R : \varepsilon)$. We then write $G \Rightarrow_A H$ or $G \Rightarrow_\pi H$.

A *contraction* of a graph G is a surjective morphism $\xi \colon G' \to G''$ such that G' is obtained from G by removing all borrow edges, $\dot{G}'' = \{v \in \dot{G} \mid v \text{ is not borrowed}\}$, $\bar{G}'' = \bar{G}'$, $\bar{\xi}$ is the identity, and $\dot{\xi}(v) = v$ for all $v \in \dot{G}''$. Then we write $G \underset{\xi}{\rightsquigarrow} G''$ or just $G \rightsquigarrow G''$.

A graph $H \in \mathbb{G}_\Sigma(C)$ *satisfies* φ, and we write $H \vDash \varphi$, if

$$att_H(e) : H \ \vDash \ \varphi(lab_H(e)) \tag{1}$$

for every condition edge $e \in \bar{H}$. Note that, since $\varphi(lab_H(e))$ is a formula over Σ, condition edges in H are invisible to it, i.e., whether or not (1) holds does not depend on the condition edges present in H.

The *language* generated by Γ is

$$\mathcal{L}(\Gamma) = \left\{ \tilde{G}|_\Sigma \;\middle|\; \tilde{G} \in \mathbb{G}_\Sigma(C) \text{ with } \langle S \rangle \overset{*}{\underset{\pi}{\Rightarrow}} G \rightsquigarrow \tilde{G} \text{ and } \tilde{G} \vDash \varphi \right\}.$$

Here, $\langle S \rangle$ denotes the graph of type $(m, 0)$, where m is the rank of S, consisting of a single edge, e, with label S, and m nodes such that $att_{\langle S \rangle}(e) = front_{\langle S \rangle}$. We observe that the application of a production always yields a graph with an empty rear. In particular, $\mathcal{L}(\Gamma) \subseteq \mathbb{G}_\Sigma^{(m,0)}$.

Borrowing hyperedge replacement (i.e., cBHR without conditions) corresponds to hyperedge replacement grammars with a single non-HR rule, which performs a contraction. Contractions in the BHR grammars of [10] are more restrictive in that borrowed nodes must not be contracted with nodes generated by the same production. In the settings of the present paper, this restriction can be enforced by conditions. Moreover, following [8, Def. 10], our productions have graph expressions as right-hand sides rather than single graphs. It should be clear that the use of these EBNF-style productions is only a convenience measure allowing a compact notation without affecting expressive power. In a way entirely similar to the case of context-free string grammars, these productions can be replaced by standard HR productions at the expense of introducing additional nonterminals.

4. A Conditional BHR Grammar for Statecharts

We use statecharts as a running example in this paper. In fact, we reuse it and its description from [6]. UML state machines [11] specify system behavior in terms of state transitions that occur in response to events. They are derived from Statecharts [12], which extend classical finite automata with hierarchical structuring mechanisms such as composite states. In the following, we do not elaborate on the precise semantic differences between UML state machines and Statecharts; instead, we just use the term *statechart*.

A statechart as in Fig. 1a consists of states and transitions between them. A filled black circle • denotes an *initial* state, whereas a state represented by ⦿ denotes a *final* state. Rectangles with rounded corners represent either *simple* or *composite* states. Composite states differ from simple states in that they contain one or more *regions*. Each region serves as a container for its own states and transitions. A composite state with exactly one region is called a *simple composite* state; a composite state with multiple regions is called an *orthogonal* state. In orthogonal states, regions are separated by dashed lines. When a composite state becomes active, all of its regions become active simultaneously, and they execute independently of one another.

The statechart in Fig. 1a models an auction process and comprises five top-level states: the initial one on the left, the orthogonal state Auction, and the three simple states Canceled, Finished, and Rejected. The state Auction contains two regions, Bidding and Payment, each with its own initial and final states. Its intended semantics is beyond this paper and not considered here.

In the following, we require transitions of a statechart to satisfy the following conditions:

(C1) A chart must be *connected*, meaning its inner states and optional final state must be reachable from the initial state via transitions. However, reachability can be complex, as shown in Fig. 1a.

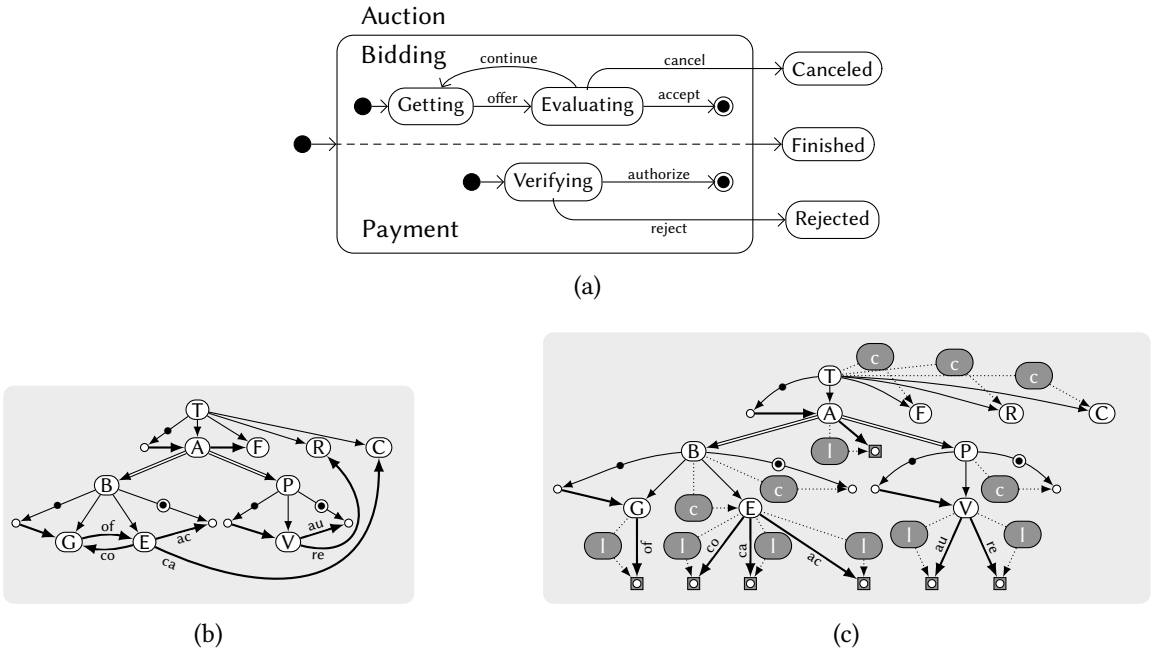


Figure 1: (a) shows a statechart modeling an auction, (b) the graph representing the statechart in (a). Node names are abbreviated by their initial letter; T names the top region; transition names are abbreviated by their first two letters. (c) shows the graph that can be derived by the productions in Fig. 2 and contracted to the graph shown in (b). The condition labels conn and legal have been abbreviated to c and l, respectively.

There is no sequence of transitions from the top-level initial state to the top-level Canceled state. Rather, the latter is reached from the contained state Evaluating, which is indirectly reachable through the initial state of region Bidding.

(C2) Transitions must not target initial states.

(C3) Transitions may only cross the border of regions and states if they do not enter any of the regions of an orthogonal state "from outside".

These conditions enforce well-structured statecharts with predictable control flow and unambiguous execution semantics. (C1) and (C2) are consistent with, and in the case of (C2) directly reflect, the interpretation of Harel's statecharts [12] as reactive models.

Figure 1b shows how we represent the auction statechart as a graph. In such illustrations, we draw nodes as circles and non-binary edges as boxes, the boxes containing the edge label and being connected to their attached nodes by lines; binary edges are drawn as arrows and have their label ascribed, or are distinguished by different arrow styles.

The graph representation of the statechart is essentially a tree with nodes representing states or regions and edges representing containment. However, it also has crosscutting edges representing transitions. The root node describes the region of the entire statechart, containment is indicated by thin arrows. Arrows pointing to initial and final states carry • and ⦿. Transitions are drawn as thick arrows. Double-lined arrows point from orthogonal states to their regions.

The language of statechart graphs is defined by the productions and conditions in Fig. 2. The drawing convention for front and rear interfaces of graphs is that their nodes are connected with double lines to the left and right border, respectively, of the box underlying the graph. Both sequences are ordered from top to bottom.

Prior to contraction, the productions in Fig. 2 derive tree-like graphs where transition edges are attached to borrowed nodes instead of their intended target nodes. Then, a contraction turns these transition edges into crosscutting transitions. The condition edges conn and legal (depicted as rectangles with rounded corners) make sure that the resulting graph satisfies the conditions (C1), (C2), and (C3).

Figure 1c shows the uncontracted graph that can be derived by the productions starting at the start symbol S. This graph can then be contracted to the auction graph in Fig. 1b. The idea behind the

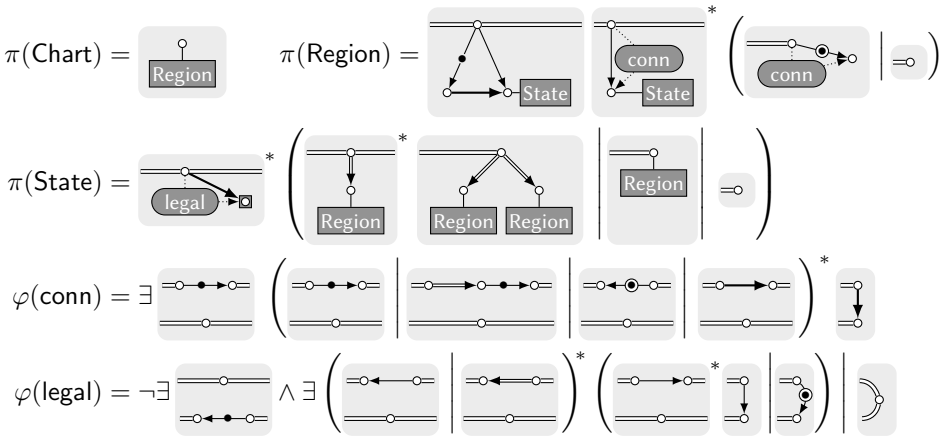


Figure 2: A conditional BHR grammar for statecharts. Chart is the startsymbol.

productions is as follows: A nonterminal Region-edge is attached to a node if it represents a region. Each region contains a unique initial state and at least one additional state. The latter is represented by a node and an attached nonterminal State-edge. A region may also contain an arbitrary number of additional states and an optional final state. These states must be reachable from their initial state, as required by (C1); this condition is expressed by binary conn-edges. Finally, every state may have an arbitrary number of outgoing transitions to other states, represented by borrowed nodes. A legal-edge describes the condition that the borrowed node will eventually be contracted with a legal node that fulfills conditions (C2) and (C3). Each state node is either a region by itself (in which case it is a simple composite state) or it is an orthogonal state containing at least two regions, or it is just a simple state.

To see how connectedness is specified by conn-edges, consider such an edge with its attached nodes, say x and y. Its condition $\varphi(\mathsf{conn})$ is satisfied if state y is reachable from the initial state of region x. In the simplest case, y is reachable from the initial state via a sequence of transitions, i.e., thick arrows. However, $\varphi(\mathsf{conn})$ also covers the more complicated cases where y is reachable across region and state boundaries. Note in particular the final-state edge occurring in $\varphi(\mathsf{conn})$ that allows to continue with the enclosing simple composite state as soon as its final state has been reached.

Finally, legal-edges prevent contractions that produce transitions between illegal states. Note that $\varphi(\mathsf{legal})$ has the form $\neg\exists\varrho \wedge \exists(\psi|\vartheta)$. The part containing ϱ prevents transitions from entering initial states, as required by (C2), and ψ ensures condition (C3). The final part, ϑ, permits both nodes of a legal-edge to be identified by contractions, yielding looping transitions.

5. Top-Down Parsing

We now formalize a top-down parser for cBHR grammars and prove its correctness. For technical simplicity, we assume from here on that start symbols of cBHR grammars have rank 0. This can be achieved in a standard manner by adding a new start symbol and one additional rule.

We first define a notion of atoms, graphs that consist of a single nonterminal edge.

Definition 5.1. For $A \in N$, a graph G is an *A-atom* (or simply *atom*) if G has exactly one edge, e, labeled A, and front_G, rear_G, and $\mathit{att}_G(e)$ are free of repetitions. The atom is *pure* if all nodes in its front and rear interfaces are attached to e, i.e., $\mathit{front}_G, \mathit{rear}_G \subseteq [\mathit{att}_G(e)]^*$.

Given graphs G, H and $k \in \mathbb{N}$, H is a *k-extension* (or simply an *extension*) of G if there is a sequence w of k nodes in $\dot{H}$ such that $\dot{G} = \dot{H} \setminus [w]$, $\mathit{front}_H = w \cdot \mathit{front}_G$, $\mathit{rear}_H = w \cdot \mathit{rear}_G$, $\bar{G} = \bar{H}$, $\mathit{att}_G = \mathit{att}_H$, and $\mathit{lab}_G = \mathit{lab}_H$. Let $\mathrm{ext}_k(G)$ denote the k-extensions of G.

We can now define a variant of cBHR grammars useful for parsing.

Definition 5.2 (cBHR grammar for parsing)**.** As before, let $X = N \cup \{\square\} \cup C$ and Σ be disjoint. A *cBHR grammar for parsing* is a tuple of the form $\Gamma = (\Sigma, X, \Pi, S, \varphi)$, where

1. $\Pi \subseteq \mathbb{G}_N \times \mathbb{G}_\Sigma(X)^*$ is a set of productions such that, for every production $(L, R) \in \Pi$, L is a pure atom and R a sequence of $k \in \mathbb{N}$ graphs $G_1, \ldots, G_k$ where for every $i \in [k]$, either $G_i \in \mathbb{G}_\Sigma(X \setminus N)$ or it is an extension of a pure atom. Moreover, the sequential composition $\mathrm{comp}(R)$ is defined and has the same type as L.
2. $S \in N$ is the *start symbol* with $\mathit{rank}(S) = 0$.
3. The function $\varphi\colon C \to \mathbb{SF}_\Sigma$ is the *condition assignment* such that $\varphi(c) \in \mathbb{SF}_\Sigma^{\mathit{rank}(c)}$ for every condition name $c \in C$.

Consider any two sequences α and α' of graphs such that their sequential composition is defined, and a production $(L, R) \in \Pi$ with $R = G_1 \cdots G_m$ for some $m \in \mathbb{N}$. We say that α can be *rewritten* to α' using (L, R), denoted $\alpha \triangleright_\Pi \alpha'$, if there are sequences β, β' of graphs and $k \in \mathbb{N}$ such that β does not contain a graph with nonterminal edges, $\alpha = \beta\, \mathrm{ext}_k(L)\, \beta'$, and $\alpha' = \beta\, \gamma\, \beta'$ with $\gamma = \mathrm{ext}_k(G_1) \cdots \mathrm{ext}_k(G_m)$. Note that β must not contain an atom, which means that a rewrite step $\alpha \triangleright_\Pi \alpha'$ always rewrites the leftmost nonterminal edge in α. By the context-freeness of hyperedge replacement, this restriction does not affect the generated language.

The *language* $\mathcal{L}(\Gamma)$ generated by Γ contains all graphs $H|_\Sigma$ such that there is a sequence α of graphs and $H \in \mathbb{G}_\Sigma(C)$ with

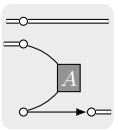
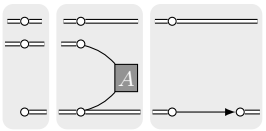
$$\langle S \rangle \triangleright_\Pi^* \alpha,\ \mathrm{comp}(\alpha) \rightsquigarrow H, \text{ and } H \vDash \varphi.$$

Note that, in contrast to Definition 3.2, cBHR grammars for parsing require the start symbol to have rank 0. This assumption simplifies the presentation of the top-down parser in the following and does not impose any real restriction. The nodes attached at the start symbol indicate where the derivation begins and, consequently, where the parsing process must start.

In practice, these nodes can always be identified by alternative means, such as attaching dedicated edges. In our statechart example (see Example 5.1 below), we use a specific condition edge that "selects" the unique node at which parsing must start.

Lemma 5.3. *For every cBHR grammar Γ whose start symbol has rank 0, there is a cBHR grammar Γ' for parsing such that $\mathcal{L}(\Gamma) = \mathcal{L}(\Gamma')$.*

Proof sketch. Let $\Gamma = (\Sigma, X, \pi, S, \varphi)$. Then $\Gamma' = (\Sigma, X, \Pi, S, \varphi)$ is obtained as follows. As for context-free grammars in EBNF, we can assume without loss of generality that, for every $A \in N$, $\pi(A)$ has the form $G_1 \mid \cdots \mid G_n$ for graphs $G_1, \ldots, G_n \in \mathbb{G}_\Sigma(X)$. Each of the graphs G_i can straightforwardly be written as a composition $G_i = \mathrm{comp}(G_i^1 \cdots G_i^{m_i})$ such that each of the G_i^j is either in $G_i \in \mathbb{G}_\Sigma(X \setminus N)$ or is an extension of a pure atom. For example,

$G_i =$  can be decomposed into 

thus yielding a production $(L, G_i^1\, G_i^2\, G_i^3)$ where L is a pure atom of the nonterminal A. We omit the correctness proof of this construction. □

Example 5.1 (cBHR grammar for parsing statecharts)**.** Figure 3 shows a cBHR grammar for parsing constructed from the cBHR grammar for statecharts in Fig. 2. As usual, we use the shorthand notation $L ::= R_1 \mid \cdots \mid R_k$ for productions $(L, R_1), \ldots, (L, R_k)$ that share a common left-hand side L. The names below "::=" refer to the names of the corresponding productions.

The nonterminals Region* and State* derive the Kleene stars in the productions for Region and State of the original grammar.

Note that we have added a condition top to production s; it identifies the node at which parsing shall start. It just requires that the state represented by the start node is not contained in another state. Note

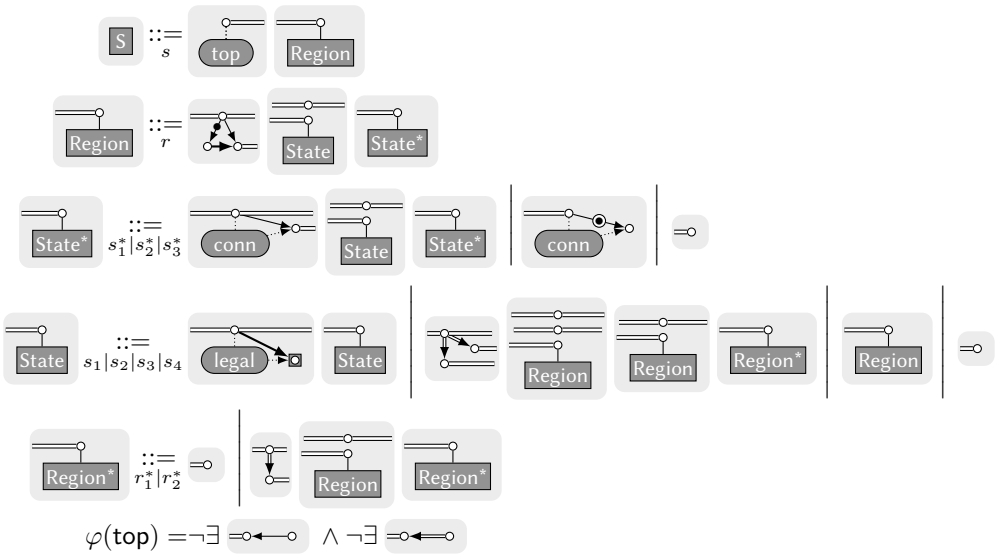


Figure 3: Productions of a conditional BHR grammar for parsing statecharts

that this condition does not change the language generated by the grammar, it just allows for starting the parse deterministically.

In the following, we consider a fixed cBHR grammar $\Gamma = (\Sigma, X, \Pi, S, \varphi)$ for parsing. We then denote rewrite steps $\alpha \triangleright_\Pi \alpha'$ simply as $\alpha \triangleright \alpha'$.

5.1. Parser

The parser is a top-down parser. It works similarly to a pushdown automaton recognizing a context-free language by trying to construct a leftmost derivation. For an input graph H of type $(0, 0)$, a *configuration* is a pair (M, σ) where $M \in \mathbb{G}_\Sigma(\{\square\})$, $M|_\Sigma$ is the subgraph of H that has already been processed (or "matched") by the parser, and σ is a sequence of graphs used as a stack. M may contain borrow edges, indicating "unresolved" nodes that have been processed as borrowed nodes, but have not yet been matched to non-borrowed nodes. Specifically, the front interface of M is always empty. Its rear interface indicates where the next move shall take place.

The stack σ contains a sequence of *stack elements*. The first element is the top of the stack. Each stack element is either a graph in $\mathbb{G}_\Sigma(X \setminus N)$ or an extension of a pure atom. If the top of the stack exists, its front interface has the same length as the rear interface of M.

A *parse* is a sequence of moves $(M_1, \sigma_1) \vdash \cdots \vdash (M_n, \sigma_n)$. In the *start configuration* (M_1, σ_1), the graph M_1 is the empty graph, and the stack σ_1 contains only the atom $\langle S \rangle$ of the start symbol, with which the construction of the derivation begins.

A parse *terminates* at (M_n, σ_n) if no further moves are possible. In particular, this is the case if the stack is empty, i.e., $\sigma_n = \varepsilon$. Such a parse is *accepting* if the entire input graph has been processed, i.e., $M_n = H$; in particular, M_n must not contain any borrow edge. If $\sigma_n \neq \varepsilon$, a successor configuration may or may not exist, depending on the following two cases:

(M1) The top of σ_n is a graph $G \in \mathbb{G}_\Sigma(X \setminus N)$. Let $\sigma_n = G\sigma'$ for some sequence σ' of stack elements, $G' = \varepsilon : G|_\Sigma : \varepsilon$, B the set of borrowed nodes of G, $U = \dot{G} \setminus (B \cup [\mathit{front}_G])$ the set of newly created, non-borrowed (or "unborrowed") nodes of G, and R the *resolved nodes* of M_n, i.e., its nodes without attached borrow edges.

As we will see, rear_{M_n} and front_G have the same length. We then look for a morphism $\mu \colon G' \to H$ satisfying the following properties:

$$\dot{\mu}^*(\mathit{front}_G) = \mathit{rear}_{M_n} \tag{2}$$

$$\bar{\mu}(\bar{G}') \cap \bar{M}_n = \varnothing \tag{3}$$

$$\dot{\mu} \text{ is injective on } U \subseteq \dot{G}' \tag{4}$$

$$\dot{\mu}(U) \cap R = \varnothing \tag{5}$$

$$\dot{\mu}^*(att_G(e)) : H \vDash \varphi(lab_G(e)) \quad \text{for each condition edge } e \in \bar{G}. \tag{6}$$

Finally, let α be any sequence of nodes in $\dot{M}_n$ (free of repetition) such that $[\alpha] = \dot{\mu}(\dot{G}) \cap \dot{M}_n$, i.e., $[\alpha]$ is the set of nodes $\mu(G')$ and M_n share. Note that $[\mathit{front}_{\mu(G')}] = [\mathit{rear}_{M_n}] \subseteq [\alpha]$.

Property (2) ensures that $\mu(G')$ can be concatenated with M_n to yield M_{n+1}. Property (3) ensures that edges of H that have already been matched in earlier moves are not matched again. Properties (4) and (5) deal with non-borrowed nodes of G that are newly created. (4) ensures that none of them is contracted with any other of them, and (5) ensures that none of them is contracted with any other non-borrowed node that has been matched in an earlier move. The latter are exactly the resolved nodes of M_n, i.e., R. Finally, property (6) ensures that each constraint imposed by a condition edge is satisfied. Note that this condition is evaluated in H.

If such a match exists, a successor configuration exists and is obtained as follows:

i. Pop G from stack σ_n to yield stack σ',

ii. Mark the match of G as processed by concatenating M_n and $\mu(G')$ to yield a graph M_{n+1}, and update the information of unresolved nodes in M_{n+1} by adding and removing borrow edges.

More precisely, construct $M' = (M_n : \alpha) \cdot (\alpha : \mu(G'))$. M_{n+1} is then obtained from M' by attaching new borrow edges to all nodes in $\dot{\mu}(B) \setminus (\dot{M}_n \cup \dot{\mu}(U))$, and by removing all borrow edges attached to any node in $\dot{\mu}(U)$. Note that the rear interface of $\mu(G')$ thus becomes the rear interface of M_{n+1}.

The move then yields the successor configuration (M_{n+1}, σ_{n+1}) with $\sigma_{n+1} = \sigma'$.

There is no successor configuration if such a match μ does not exist; that is, the parse terminates (and fails).

(M2) The top G of σ_n is a *k-extension of a pure atom L* for some $k \in \mathbb{N}$, i.e., $\sigma_n = G\sigma'$ for some sequence σ' of stack elements. Note again that that $|\mathit{rear}_{M_n}| = |\mathit{front}_G|$. A successor configuration exists if there is a production $(L, R) \in \Pi$ with $R = G_1 \cdots G_m$ for some $m \in \mathbb{N}$; the successor configuration is then obtained as follows:

i. Pop G from stack σ_n to yield stack σ' and

ii. push k-extensions of each of the graphs $G_1, \ldots, G_m$ onto the stack.

Then $M_{n+1} = M_n$ and $\sigma_{n+1} = \mathrm{ext}_k(G_1) \cdots \mathrm{ext}_k(G_m)\, \sigma'$.

There is no successor configuration if no such production exists; that is, the parse terminates (and fails).

Example 5.2. Figure 4 illustrates the first 14 parser moves performed by the top-down parser when processing the auction graph shown in Fig. 1b, using the cBHR grammar for parsing defined in Fig. 3. Each $\vdash$ symbol is annotated with the corresponding move number and, in the case of an (M2) move, additionally labeled with the production applied.

Explanatory notes accompanying the individual parser moves describe the effect of each move. Note in particular that condition edges (depicted as rectangles with rounded corners) only trigger the verification of the associated constraint; they are not incorporated into the graph M_i, which forms the first component of the subsequent parser configuration.

The top-condition verified during the second move ensures that node T is the only node that can be matched by the node attached to the top-edge.

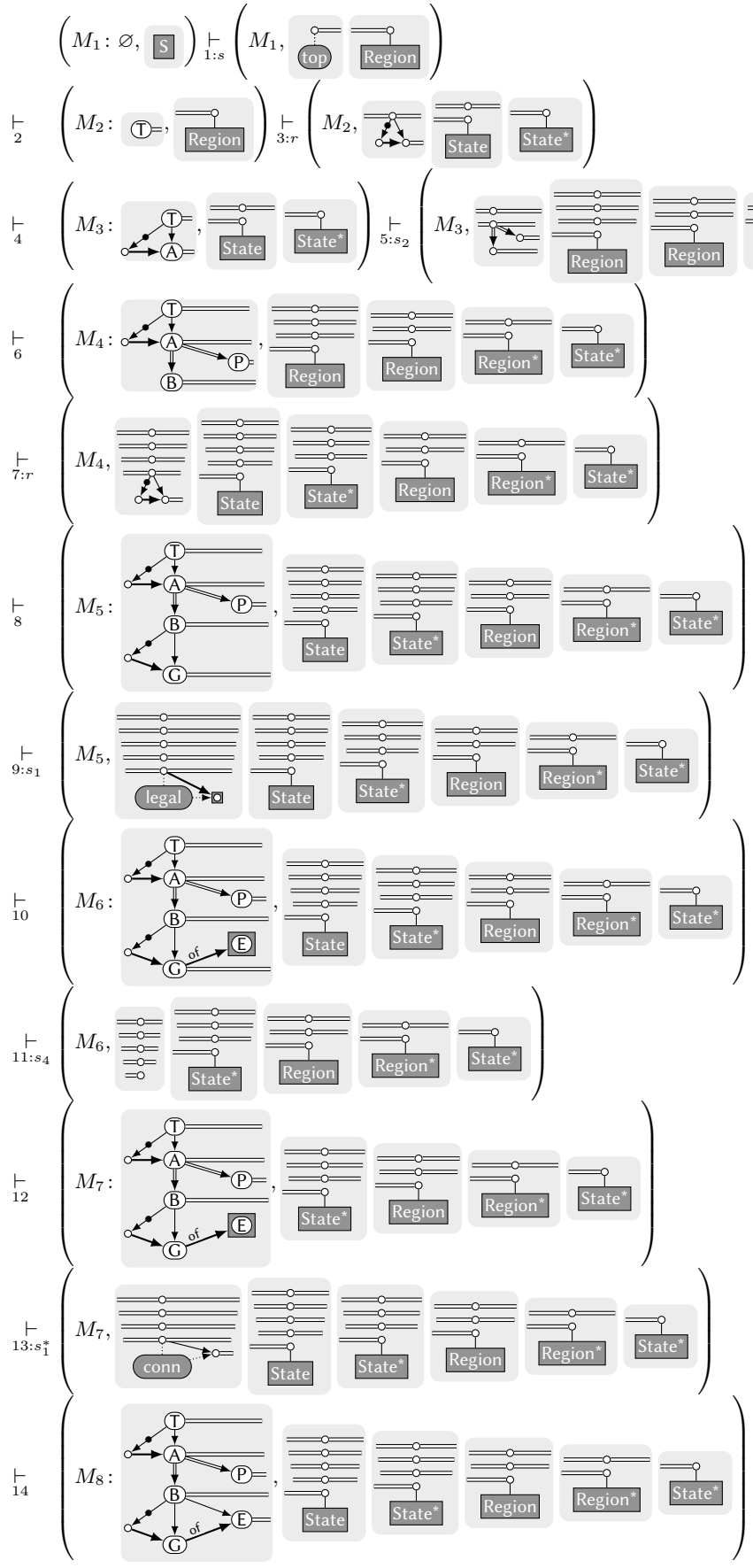


Figure 4: Fourteen moves of the top-down parser for recognizing the Auction statechart

The following theorem is the main result of this paper. It states that the parser is correct in the sense that it accepts an input graph if and only if the input graph is valid:

Theorem 5.4. *A graph H is accepted by the parser if and only if $H \in \mathcal{L}(\Gamma)$.*

We prove the theorem in two steps: In Sect. 5.2, we show that every graph in the language $\mathcal{L}(\Gamma)$ is accepted by the parser. In Sect. 5.3, we then show the reverse direction, i.e., every graph accepted by the parser is in $\mathcal{L}(\Gamma)$.

5.2. Correctness, first step: for every graph $H \in \mathcal{L}(\Gamma)$ there is an accepting parse

For every graph $H \in \mathcal{L}(\Gamma)$, by definition there is a sequence $\alpha_1 \triangleright \alpha_2 \triangleright \cdots \triangleright \alpha_n$ of rewrite steps such that

$$\alpha_1 = \langle S \rangle, \mathrm{comp}(\alpha_n) \underset{\xi}{\rightsquigarrow} H',\ H' \vDash \varphi, \text{ and } H = H'|_\Sigma$$

for some graph $H' \in \mathbb{G}_\Sigma(C)$ and contraction ξ.

By the definition of rewrite steps, for each $i \in [n-1]$ there exist graph sequences β_i, β_i' and an extension A_i of a pure atom such that $\alpha_i = \beta_i A_i \beta_i'$, and A_i is replaced in the i-th rewrite step. Consequently, $\beta_1 = \beta_1' = \varepsilon$ and $A_1 = \langle S \rangle$. The chain of rewrite steps has thus the form

$$\langle S \rangle = \beta_1 A_1 \beta_1' \ \triangleright\ \beta_2 A_2 \beta_2' \ \triangleright\ \cdots \ \triangleright\ \beta_{n-1} A_{n-1} \beta_{n-1}' \ \triangleright\ \alpha_n$$

where α_n does not contain any graph with a nonterminal edge. Each β_i must be a prefix of α_n because rewrite steps always replace the leftmost nonterminal edge. For every $i \in [n-1]$, let ℓ_i be the length of β_i. Obviously, $\ell_1 = 0$, and for convenience, we let $\ell_n = |\alpha_n|$. Clearly, $\ell_1 \leq \cdots \leq \ell_n$. Note that $\ell_{i+1} - \ell_i = 0$ or $\ell_{i+1} - \ell_i > 1$ is possible.

We now consider the sequence $(1, \ell_1), \ldots, (n, \ell_n)$ of pairs. For reasons that will become clear below, we "fill the gaps" in this sequence in the following way: Let I be the set of all pairs $(i, j) \in [n] \times [\ell_n]$ such that $\ell_{i-1} \leq j \leq \ell_i$ for $i > 1$, and order these pairs lexicographically. That way, we end up with a sequence

$$(i_1, j_1), \ldots, (i_N, j_N)$$

of pairs such that $(i_1, j_1) = (1, \ell_1) = (1, 0)$ and, for all $k \in [N-1]$,

$$(i_{k+1}, j_{k+1}) = \begin{cases} (i_k, j_k + 1) & \text{if } j_k < \ell_{i_k} \\ (i_k + 1, j_k) & \text{if } j_k = \ell_{i_k} \end{cases} \tag{7}$$

Finally, we define the graphs $G_0, \ldots, G_{\ell_n}$ by $G_j = \mathrm{comp}(\alpha_j')$ for every $j \in [\ell_n]$, where α_j' is the prefix of α_n of length j, and G_0 is the empty graph.

We are now ready to define configurations $T_{ij} = (M_j, \sigma_{ij})$ for all pairs $(i, j) \in I$ as follows:

- Let M_j' be the graph obtained from $G_j|_\Sigma$ by applying the contraction morphism ξ restricted to $G_j|_\Sigma$. That way, M_j' is a subgraph of H. The graph M_j is then obtained from M_j' by attaching borrow edges to all nodes in $\dot{\xi}(\dot{G}_j) \setminus \dot{G}_j$.
- σ_{ij} is the suffix of α_i such that $\alpha_i = \alpha_j' \sigma_{ij}$ (which is α_i itself if $j = 0$). Note that every σ_{ij} is well-defined because $j \leq \ell_i$ by the definition of I.

Lemma 5.5. *$T_{i_1 j_1} \vdash T_{i_2 j_2} \vdash \cdots \vdash T_{i_N j_N}$ is an accepting parse for H.*

Proof. We prove the lemma by showing that $T_{i_1 j_1}$ and $T_{i_N j_N}$ are the first and last parser configuration, respectively, of an accepting parse for H, and that $T_{i_p j_p} \vdash T_{i_{p+1} j_{p+1}}$ is a valid parser move for every $p \in [N-1]$.

$T_{i_1 j_1}$ is clearly the start configuration since $T_{i_1 j_1} = (M_0, \langle S \rangle)$ and M_0 is the empty graph. To show that $T_{i_N j_N}$ is the final parser configuration of an accepting parse, first observe that $\xi \colon G_{\ell_n} \to H$ is a surjective morphism, $\dot{H} = \{v \in \dot{G}_{\ell_n} \mid v \text{ is not borrowed}\}$, and $\dot{\xi}(v) = v$ for every $v \in \dot{H}$ by

Definition 3.2. Hence, $M'_{\ell_n} = H$. Because of $\dot{\xi}(\dot{G}_{\ell_n}) = \dot{H} \subseteq \dot{G}_{\ell_n}$, we can conclude $\dot{\xi}(\dot{G}_{\ell_n}) \setminus \dot{G}_{\ell_n} = \varnothing$ and thus $M_{\ell_n} = M'_{\ell_n} = H$. Consequently, $T_{i_N j_N} = (M_{\ell_n}, \varepsilon) = (H, \varepsilon)$, which is the last parser configuration of an accepting parse for H.

It remains to be shown that $T_{i_p j p} \vdash T_{i_{p+1} j_{p+1}}$ for every $p \in [N-1]$. For the following, choose any $p \in [N-1]$. By (7), we must distinguish two cases:

Case 1. $j_p < \ell_{i_p}$ and hence $i_{p+1} = i_p$ and $j_{p+1} = j_p + 1$. For convenience, let $i = i_p = i_{p+1}$ and $j = j_p$ and thus $j_{p+1} = j + 1$.

Hence, $\alpha'_{j+1} = \alpha'_j G$ is a prefix of β_i for some $G \in \mathbb{G}_\Sigma(X \setminus N)$, and thus $\sigma_{ij} = G \cdot \sigma_{i(j+1)}$. Let us define the sets B and U in the same way as for move (M1) in the description of the parser, i.e., B is the set of borrowed nodes of G and $U = \dot{G} \setminus (B \cup [\mathit{front}_G])$ is the set of newly created, "unborrowed" nodes of G.

Recall that M'_j is obtained from $\mathrm{comp}(\alpha'_j)|_\Sigma$ by applying morphism ξ, and M_j is constructed from M'_j by attaching borrow edges to all nodes in $\dot{\xi}(\dot{G}_j) \setminus \dot{G}_j$. M'_{j+1} and M_{j+1} are constructed analogously. Now let $G' = \varepsilon : G|_\Sigma : \varepsilon$. Since M'_j and M'_{j+1} are subgraphs of H, there must be a morphism $\mu \colon G' \to H$ and a sequence α of H-nodes that occur in M'_j as well as $\mu(G')$ such that properties (2)–(6) are satisfied, and $M'_{j+1} = (M'_j : \alpha) \cdot (\alpha : \mu(G'))$. Hence, M_{j+1} can also be obtained by first constructing $M' = (M_j : \alpha) \cdot (\alpha : \mu(G'))$, and then removing and adding certain borrow edges as follows:

- Consider a borrow edge attached to node v in M_j that must be removed when constructing M_{j+1}. Consequently, $v \in \dot{\xi}(\dot{G}_j) \setminus \dot{G}_j$, but $v \notin \dot{\xi}(\dot{G}_{j+1}) \setminus \dot{G}_{j+1}$. Since $\dot{G}_j \subseteq \dot{G}_{j+1}$, it follows $v \in \dot{\xi}(\dot{G}_j) \subseteq \dot{\xi}(\dot{G}_{j+1})$ and $v \in \dot{G}_{j+1} \setminus \dot{G}_j$, i.e., v is a node added to M_{j+1} as a $\dot{\mu}$-image of a non-borrowed node of G and thus $v \in \dot{\mu}(U)$.
- Now consider a borrow edge that must be attached to a node v in M_{j+1}, but there was no such borrow edge in M_j. Consequently, $v \in \dot{\xi}(\dot{G}_{j+1}) \setminus \dot{G}_{j+1}$, but $v \notin \dot{\xi}(\dot{G}_j) \setminus \dot{G}_j$. Hence, $v \in \dot{\xi}(\dot{G}_{j+1})$, $v \notin \dot{G}_{j+1} \supseteq \dot{G}_j$ and thus $v \notin \dot{\xi}(G_j) = \dot{M}_j$. Consequently, $v \in \dot{\mu}(B) \setminus (\dot{M}_j \cup \dot{\mu}(U))$.

Obviously, all conditions for move (M1) are satisfied, and thus $T_{i_p j p} \vdash T_{i_{p+1} j_{p+1}}$ follows.

Case 2. $j_p = \ell_{i_p}$ and hence $i_{p+1} = i_p + 1$ and $j_{p+1} = j_p = \ell_{i_p}$.

β_{i_p} is a prefix of α_n of length $\ell_{i_p} = j_p$, and hence $\alpha'_{j_p} = \beta_{i_p}$. Then $\sigma_{i_p j_p} = A_{i_p} \beta'_{i_p}$ follows from $\alpha_{i_p} = \beta_{i_p} A_{i_p} \beta'_{i_p}$. Since $\alpha_{i_p} \triangleright \alpha_{i_p+1} = \alpha_{i_{p+1}}$, there are a production $(L, R) \in \Pi$ with $R = H_1 \cdots H_m$ and some $k, m \in \mathbb{N}$ such that $A_{i_p} = \mathrm{ext}_k(L)$ and $\alpha_{i_{p+1}} = \beta_{i_p} \mathrm{ext}_k(H_1) \cdots \mathrm{ext}_k(H_m) \beta'_{i_p}$. Hence $\sigma_{i_{p+1} j_{p+1}} = \mathrm{ext}_k(H_1) \cdots \mathrm{ext}_k(H_m) \beta'_{i_p}$, and

$$T_{i_p j_p} = (M_{j_p}, \sigma_{i_p j_p}) = (M_{j_p}, \mathrm{ext}_k(L) \beta'_{i_p}) \vdash (M_{j_p}, \mathrm{ext}_k(H_1) \cdots \mathrm{ext}_k(H_m) \beta'_{i_p}) = T_{i_{p+1} j_{p+1}}$$

follows. □

5.3. Correctness, second step: if $H \in \mathbb{G}_\Sigma$ has an accepting parse, then $H \in \mathcal{L}(\Gamma)$

Consider any graph $H \in \mathbb{G}_\Sigma$ and an accepting parse $(M_1, \sigma_1) \vdash (M_2, \sigma_2) \vdash \cdots \vdash (M_n, \sigma_n)$ for H. To prove $H \in \mathcal{L}(\Gamma)$, we construct, for every $i \in [n]$, a sequence α_i of graphs in $\mathbb{G}_\Sigma(X \setminus N)$ such that their sequential composition is defined, and a morphism $\xi_i \colon \mathrm{comp}(\alpha_i)|_\Sigma \to M_i$ as follows:

Let $\alpha_1 = \varepsilon$, and let ξ_1 be the empty morphism.[3]

For $i > 1$, consider the parser move $(M_{i-1}, \sigma_{i-1}) \vdash (M_i, \sigma_i)$ with $\sigma_{i-1} = G \cdot \sigma$. The move is either an (M1) or an (M2) move:

- We let $\alpha_i = \alpha_{i-1}$ and $\xi_i = \xi_{i-1}$ if the move is an (M2) move, i.e., G is an atom.
- Otherwise, i.e., if it is an (M1) move and $G \in \mathbb{G}_\Sigma(X \setminus N)$, we let $\alpha_i = \alpha_{i-1} \cdot G$. Note that every node and edge of $\mathrm{comp}(\alpha_i)|_\Sigma = \mathrm{comp}(\alpha_{i-1} \cdot G)|_\Sigma$ is also a node or edge of $\mathrm{comp}(\alpha_{i-1})|_\Sigma$ or G (or in both of them if they are corresponding interfaces nodes).[4] We refer to the description

[3] By convention, we let $\mathrm{comp}(\varepsilon)$ be the empty graph.

[4] Strictly speaking, all graphs under consideration are abstract ones, i.e., defined only up to isomorphism. To avoid unnecessary technical overhead, we treat them as concrete graphs and argue on the level of nodes and edges. In particular, we implicitly

of move (M1) and in particular to the definition of the morphism $\mu\colon G' \to H$ underlying the move $(M_{i-1}, \sigma_{i-1}) \vdash (M_i, \sigma_i)$. The morphism $\xi_i\colon \operatorname{comp}(\alpha_i)|_\Sigma \to M_i$ is constructed from ξ_{i-1} and this morphism μ as follows: We let $\dot{\xi}_i(v) = \dot{\xi}_{i-1}(v)$ if v is a node of $\operatorname{comp}(\alpha_{i-1})$, otherwise $\dot{\xi}_i(v) = \dot{\mu}(v)$. The edge mapping $\bar{\xi}_i$ is chosen such that $\bar{\xi}_i(e) = \bar{\xi}_{i-1}(e)$ if e is an edge of $\operatorname{comp}(\alpha_{i-1})$. Every other edge e in $\operatorname{comp}(\alpha_i)|_\Sigma$ is an edge of G'; we then let $\bar{\xi}_i(e) = \bar{\mu}(e)$.

Lemma 5.6. $H \in \mathcal{L}(\Gamma)$.

Proof. First note that $\alpha_1\sigma_1 = \langle S\rangle$, and $\alpha_i\sigma_i \trianglerighteq \alpha_{i+1}\sigma_{i+1}$ for each $i \in [n-1]$ by the constriction of α_j, $j \in [n]$. $\langle S\rangle \triangleright^* \alpha_n$ then follows from $\sigma_n = \varepsilon$ as the parse is accepting.

By construction, every α_i is a prefix of α_n for every $i \in [n]$, and $\xi_i\colon \operatorname{comp}(\alpha_i)|_\Sigma \to M_i$ is constructed, for every graph in the sequence α_i, from the corresponding morphism μ when that graph has been processed by its (M1) move. Consequently, ξ "inherits" from properties (3)–(5) of these μ-morphisms:

- $\dot{\xi}$ is bijective for all non-borrowed nodes in $\operatorname{comp}(\alpha_i)$, i.e., for all nodes that are not attached to a borrow edge.
- $\bar{\xi}$ is bijective.

By similar arguments as in the proof of Lemma 5.5, one can see that, if M_i has been constructed by an (M1) move, it has borrow edges attached to exactly those nodes that are unresolved. Consequently, a node $v \in \dot{M}_i$ is attached to a borrow node if and only if every node in $\xi_i^{-1}(v)$ is a borrowed node in $\operatorname{comp}(\alpha_i)$. Now recall that M_n does not contain any borrow edge as the parse is accepting. Consequently, $\xi_n\colon \operatorname{comp}(\alpha_n)|_\Sigma \to H$ is a contraction morphism.[5]

Finally note that $\operatorname{comp}(\alpha_n)$ also contains the condition edges produced in the rewrite process $\langle S\rangle \triangleright^* \alpha_n$. Now consider the graph H' obtained from H by adding condition edges as follows: for every condition edge e in $\operatorname{comp}(\alpha_i)$ with condition label $c \in C$, add a new condition edge e' with the same label c to H' and attach it to the images of the attachment of e under $\dot{\xi}_n$. Obviously,

$$att_{H'}(e')) : H' \vDash \varphi(c)$$

by property (6) of the (M1) move that added edge e to α_n. Hence, $H' \vDash \varphi$, $H = H'|_\Sigma$, and thus $H \in \mathcal{L}(\Gamma)$. □

Together, Lemmas 5.5 and 5.6 complete the proof of Theorem 5.4.

5.4. Nondeterminism of the Statecharts parser

Note that moves (M1) and (M2) of the parser described in Sect. 5.1 are highly nondeterministic in the sense that a configuration may have more than one successor. To illustrate this, consider again the parser moves shown in Fig. 4. Its first four moves are deterministic, whereas the fifth is not. Instead of choosing production s_2, the (M2) move could also have chosen s_1, s_3, or s_4, leading to completely different parses. However, all of these alternatives eventually fail because node A (for Accepting; see Fig. 1a) is an orthogonal state and therefore requires production s_2. Consequently, if the parser selects the wrong production in its fifth step, the parse will ultimately fail. The parser must then backtrack and eventually choose s_2 in the fifth move. Fortunately, in this example, this nondeterminism can easily be avoided. Similar to a top-down string parser, the parser need only inspect the yet unconsumed portion of the input graph and always select production s_2 whenever it expands a State edge and with at least two outgoing $\Rightarrow$-edges.

The parser's sixth move is also nondeterministic. Instead of producing M_4 with rear interface T A P B, the (M1) move could have matched nodes B and P differently. In that case, it would have produced M_4'

assume that sequential compositions are formed under the simplifying conventions of Definition 2.3, possibly after replacing graphs by suitable isomorphic copies.

[5] A contraction morphism ξ requires $\dot{\xi}(v) = v$ for every non-borrowed node v. As we consider only abstract graphs, we assume, for every $i \in [n]$, that all nodes in $\operatorname{comp}(\alpha_i)$ are "chosen" such that they are in fact nodes in H and thus $\dot{\xi}_i(v) = v$ holds for every non-borrowed node v of $\operatorname{comp}(\alpha_i)$.

with rear interface T A B P and proceeded with a different parse. Unlike the nondeterminism in the fifth move, however, both alternatives in the sixth move are equally valid, in the sense that the parser can still find an accepting parse from either configuration. We call this form of nondeterminism *benign*.

In fact, a closer analysis of the cBHR grammar for parsing Statechart graphs (Fig. 3) reveals that (M1) moves exhibit only benign nondeterminism. Furthermore, any nondeterminism caused by (M2) moves can always be eliminated by inspecting the yet unconsumed portion of the input graph, as in the fifth move discussed above. Hence, although the parser remains nondeterministic, it can be implemented efficiently because backtracking is no longer required.

6. Conclusions and Future Work

Parsers as defined in this paper are inherently nondeterministic and need backtracking to parse a graph, which is inefficient. In general, this is probably unavoidable even in the case of HR languages, due to the fact that HR languages can be NP-complete. To avoid backtracking, the grammar must meet certain conditions as outlined for the Statecharts grammar. For the general case, let us say that a parser configuration κ is *promising* (with respect to a given input graph) if there is a sequence of moves starting at κ, that leads to an accepting configuration. Then there needs to be an efficient algorithm determining for every promising configuration κ a move $\kappa \vdash \kappa'$ such that κ' is promising. Thus, if $\kappa = (M, \sigma)$, there are two cases depending on the nature of the top graph G of σ: (a) if G is terminal, a match must be determined such that the corresponding move results in a promising configuration; (b) if G is nonterminal, one must similarly determine a production whose application yields a move that results in a promising configuration (in both cases assuming that κ is promising). How this can be done in the case of HR grammars (and under which assumptions) was studied in [2, 3] under the notion of predictive top-down (PTD) parsers. For BHR grammars the situation is similar and working out the necessary techniques appears to be rather straightforward. However, unless cBHR is used, contraction is very unregulated, meaning that the conditions for PTD parsing need to be rather restrictive in grammars. This is because, if the number of generated borrowed nodes grows unboundedly, then contraction can increase the original degree of any non-borrowed node arbitrarily, which is difficult to handle in PTD parsing. Conditional BHR changes this, not only because conditions impose structural requirements, but also because they can be used to guide the parsing process. In future work, we therefore plan to study PTD parsing of cBHR grammars, and in particular how the conditional aspect can enable (a) and (b) above.

Acknowledgments

We thank the reviewers and Annegret Habel for their insightful suggestions. We have tried to incorporate as many as possible of them in this paper. Unfortunately, the page limit prevented us from implementing the more extensive ones.

Declaration on Generative AI

The authors have not employed any Generative AI tools.

References

[1] A. Habel, Hyperedge Replacement: Grammars and Languages, volume 643 of *LNCS*, Springer, 1992. doi:10.1007/BFb0013875.

[2] F. Drewes, B. Hoffmann, M. Minas, Predictive top-down parsing for hyperedge replacement grammars, in: F. Parisi-Presicce, B. Westfechtel (Eds.), Graph Transformation - 8th International

Conf., ICGT 2015. Proceedings, volume 9151 of *LNCS*, Springer, 2015, pp. 19–34. doi:10.1007/978-3-319-21145-9_2.

[3] F. Drewes, B. Hoffmann, M. Minas, Graph parsing as graph transformation - correctness of predictive top-down parsers, in: F. Gadducci, T. Kehrer (Eds.), Graph Transformation - 13th International Conference, ICGT 2020, volume 12150 of *LNCS*, Springer, 2020, pp. 221–238. doi:10.1007/978-3-030-51372-6_13S.

[4] F. Drewes, B. Hoffmann, M. Minas, Predictive shift-reduce parsing for hyperedge replacement grammars, in: F. Gadducci, T. Kehrer (Eds.), Graph Transformation - 10th International Conf., ICGT 2017, Proceedings, volume 12741 of *LNCS*, Springer, 2017, pp. 106–122. doi:10.1007/978-3-319-61470-0_7.

[5] F. Drewes, B. Hoffmann, M. Minas, Formalization and correctness of predictive shift-reduce parsers for graph grammars based on hyperedge replacement, Journal on Locical and Algebraic Methods for Programming (JLAMP) 104 (2019) 303–341. doi:10.1016/j.jlamp.2018.12.006.

[6] F. Drewes, B. Hoffmann, M. Minas, Conditional borrowing hyperedge replacement, in: A. Blair, O. Semeráth (Eds.), Graph Transformation – 19th International Conference, ICGT 2026, LNCS, Springer Nature Switzerland, 2026. To appear.

[7] F. Drewes, B. Hoffmann, M. Minas, Acyclic contextual hyperedge replacement: Decidability of acyclicity and generative power, in: N. Behr, D. Strüber (Eds.), Graph Transformation - 15th International Conference, ICGT 2022, volume 13349 of *LNCS*, Springer, 2022, pp. 3–19. doi:10.1007/978-3-031-09843-7_1.

[8] J. Engelfriet, J. J. Vereijken, Context-free graph grammars and concatenation of graphs, Acta Informatica 34 (1997) 773–803. doi:10.1007/s002360050106.

[9] F. Drewes, B. Hoffmann, M. Minas, Graph formulas and their translation to alternating graph automata, in: J. Endrullis, M. Tichy (Eds.), Graph Transformation – 18th International Conference, ICGT 2025, volume 15720 of *LNCS*, Springer Nature Switzerland, Cham, 2025, pp. 112–132. doi:10.1007/978-3-031-94706-3_6.

[10] F. Drewes, B. Hoffmann, M. Minas, Rule-based top-down parsing for acyclic contextual hyperedge replacement grammars, in: F. Gadducci, T. Kehrer (Eds.), Graph Transformation - 14th International Conference, ICGT 2021, volume 12741 of *LNCS*, Springer, 2021, pp. 164–184. doi:10.1007/978-3-030-78946-6_9.

[11] Object Management Group, OMG Unified Modeling Language (OMG UML), Version 2.5.1, https://www.omg.org/spec/UML/2.5.1, 2017. Formal/2017-12-05.

[12] D. Harel, On visual formalisms, Comm. of the ACM 31 (1988) 514–530. doi:10.1145/42411.42414.