Electronic Communications of the EASST
Volume 4 (2006)

Proceedings of the Second International Workshop on Graph and Model Transformation (GraMoT 2006)

Parsing of Adaptive Star Grammars

Mark Minas

15 pages

Guest Editors: Gabor Karsai, Gabriele Taentzer
Managing Editors: Tiziana Margaria, Julia Padberg, Gabriele Taentzer
ECEASST Home Page: http://www.easst.org/eceasst/ ISSN 1863-2122

Parsing of Adaptive Star Grammars

Mark Minas

Mark.Minas@unibw.de, http://www.unibw.de/Mark.Minas
Institut für Softwaretechnik
Universität der Bundeswehr München, Germany

Abstract: In a recent paper, adaptive star grammars have been proposed as an extension of node and hyperedge replacement grammars [DHJ+06]. A rule in an adaptive star grammar is actually a rule schema which, via the so-called cloning operation, yields a potentially infinite number of concrete rules. Adaptive star grammars are motivated by application areas such as modeling and refactoring object-oriented programs, and they are more powerful than node and hyperedge replacement grammars by this mechanism. It has been shown that the membership problem is decidable for a reasonably large subclass of adaptive star grammars, however no parser has been proposed. This paper describes such a parser for this subclass motivated by the well-known string parser by *Cocke*, *Younger*, and *Kasami*.

Keywords: Graph parser, adaptive star grammar

1 Introduction

Refactoring is a software engineering technique that aims at enhancing the structure of object-oriented software while preserving its behavior. Hoffmann, Janssens, and Van Eetvelde have introduced cloning and expanding graph transformations for refactoring [HJV06]. Cloning is used in the context of *multiple* nodes that may be copied ("cloned") arbitrarily often. However, well-formedness of such graph transformations could only be specified in terms of a meta-model. Meta-models, however, are not well suited for program syntax specifications because of the recursive structure of program syntax. In a recent paper, *adaptive star grammars* have been introduced as an extension of node and hyperedge replacement grammars and based on the mechanism of cloning [DHJ+06]. Such grammars are well suited for specifying the language of graphs that are used for refactoring. It has been shown that the membership problem for a certain subclass of those grammars is decidable. However, no parser usable in practice has been presented. This paper proposes such a parser.

The running example of this paper consists of arithmetic expressions with variables that are used consistently with their declaration. We allow to declare arbitrarily many integer and float variables. Expressions are either variables or the sum of two expressions. However, an integer expression cannot be added to a float expression directly. The integer expression has to be converted to a float expression by a conversion operator first. Graphs are used to represent such expressions together with variable declarations. The language of such graphs cannot be described in a context-free way. But it can be specified with an adaptive star grammar. The example, hence, shows the usefulness of adaptive star grammars from a practical point of view, too.

The next section briefly surveys other work on graph parsing. Sec. 3 then summarizes the concept of adaptive star grammars as proposed in [DHJ+06]. More restricted versions of these concepts better suited for parsing are used in this paper. Sec. 4 proposes a Cocke-Younger-Kasami-style parser for adaptive star grammars without isolated nodes in their rules' right-hand sides. This additional restriction is dropped in Sec. 5. Sec. 6 concludes the paper.

2 Related Work

Graph parsers have been used in different application areas, but mainly in the context of visual language parsing. Rekers and Schürr have proposed a parser for context-sensitive graph grammars with an additional layering condition [RS97]. Visual language syntax is specified by such grammars, i.e., the parser can be used for syntax checking. The paper also contains a nice survey of other parsing approaches. More efficient extensions or modifications of this parser, also in the context of visual language recognition, have been proposed more recently [BTS00, ZZC01]. The parser described in this paper has been motivated by the parser for hypergraph grammars used in DIAGEN [Min02]. This parser is restricted to hypergraph grammars being context-free with some extensions beyond context-freeness that allow for efficient parsing in practical cases. Other work on graph parsing, e.g., covers computer linguistics [SF04] or theoretical research on the complexity of graph parsers [Dre93].

3 Adaptive Star Grammars

Throughout the paper, let $\mathcal{V}$ be a set of labels partitioned into sets $\dot{\mathcal{V}}$ and $\bar{\mathcal{V}}$ of node and edge labels, resp. A finite subset Σ of $\mathcal{V}$ is called a labeling alphabet. Its two components are $\dot{\Sigma} = \Sigma \cap \dot{\mathcal{V}}$ and $\bar{\Sigma} = \Sigma \cap \bar{\mathcal{V}}$. Node labels $\dot{\mathcal{V}}$ are partitioned into sets $\mathcal{N}$ and $\mathcal{T}$ of nonterminal resp. terminal labels. As usual, a *graph* $G = \langle \dot{G}, \bar{G}, s_G, t_G, \dot{\ell}_G, \bar{\ell}_G \rangle$ consists of finite sets $\dot{G}$ of *nodes* and $\bar{G}$ of *edges*, of *source* and *target functions* $s_G, t_G \colon \bar{G} \to \dot{G}$, and of node and edge labeling functions $\dot{\ell}_G \colon \dot{G} \to \dot{\mathcal{V}}$ and $\bar{\ell}_G \colon \bar{G} \to \bar{\mathcal{V}}$. However, nonterminal nodes cannot be target of any edge, i.e., for all edges $e \in \bar{G}$, it is required that $\dot{\ell}(t_G(e)) \notin \mathcal{N}$. The set of all graphs labeled over a labeling alphabet Σ is denoted by $\mathcal{G}_\Sigma$.

An edge is said to be *incident* with its source and target nodes, and makes these nodes *adjacent* to each other. For $n \in \dot{G}$, $\mathrm{Inc}_G(n)$ denotes all edges of G incident with n. For $A \subseteq \dot{G}$, $G \setminus A$ denotes the subgraph of G induced by $\dot{G} \setminus A$. Morphisms and isomorphisms are defined as usual. And $Z = X \cup Y$ where X and Y are graphs with disjoint edge sets, i.e., $\bar{X} \cap \bar{Y} = \emptyset$, denotes the union of X and Y with node set $\dot{Z} = \dot{X} \cup \dot{Y}$ and edge set $\bar{Z} = \bar{X} \cup \bar{Y}$.

The central notion of adaptive star grammars is the *star*, a generalized version of the hyperedges known from hyperedge replacement grammars [DHK97]. A star is a graph S that consists of a nonterminal node x, edges incident with x (called *arms* of x) and its adjacent nodes (called *border* nodes). By the definition of graphs, each arm points from the center node to a border node. A star is *straight* if the target nodes of its arms are pairwise distinct. For a graph G and a node $x \in \dot{G}$, the star $G(x)$ denotes the subgraph of G consisting of x, all its incident edges, and all its adjacent nodes.

Adaptive star grammars are based on star rules. A *star rule* $p = (L ::= R)$ is a rule which

replaces the star L (left-hand side) by a graph R (right-hand side). L and R share precisely the border nodes of L. A star rule is called straight if the left-hand side and every star of the right-hand side is straight. A graph G can be derived to graph H by p, written $G \Rightarrow_p H$, if G has some node x such that the left-hand side L is isomorphic to $G(x)$. The replacement of x by R yields the graph H which is obtained from the disjoint union of G and R by removing x and its arms, and identifying every border node b of L in R with its image $m(b) \in \dot{G}$.

So far, we have used regular nodes only. So-called *multiple* nodes, however, may be copied ("cloned") arbitrarily often. A similar mechanism can be found in the PROGRES graph transformation language [SWZ99]. We use a special subset $\mathscr{T}$ of multiple node labels. The remaining node labels are said to be *singular* ones. We assume that every singular node label l has a copy $\ddot{l}$ among the multiple node labels. A node is said to be singular or multiple depending on its label. $\ddot{G}$ denotes the set of all multiple nodes of graph G.

Cloning is performed by the cloning operation. Using this operation, a multiple node is turned into any number of singular nodes, its clones. Thus, we define $G^{\frac{x}{n}}$ to be obtained from G by replacing the multiple node x of G with $n \geq 0$ singular clones in the following way: Let $G'(x)$ be obtained from $G(x)$ by replacing the label $\ddot{l}$ of x by l. Then take the disjoint union of the graph $G \setminus \{x\}$ and n copies of $G'(x)$. Finally, identify the $n+1$ copies of each node in $\dot{G}(x) \setminus \{x\}$ with each other.

The result obtained by cloning a number of nodes is independent of the order in which those nodes are treated as shown in [DHJ+06]. We define a cloning operation for the set of multiple nodes in a graph. For each multiple node in the set, the necessary information about the number of desired clones is given by a so-called *multiplicity function* $\mu \colon \ddot{G} \to \mathbb{N}$. If $\ddot{G} = \{x_1, x_2, \ldots, x_k\}$ (where $x_1, \ldots, x_k$ are pairwise distinct), then G^μ is the graph defined by $G^\mu = (\ldots((G^{\frac{x_1}{\mu(x_1)}})^{\frac{x_2}{\mu(x_2)}})\ldots)^{\frac{x_k}{\mu(x_k)}})$.

Note that this definition of cloning is a variation of the definition in [DHJ+06] where clones of multiple nodes can be multiple nodes again. However, as shown in [DHJ+06], this more general concept is not required for parsing.

Finally an *adaptive star grammar* $\Gamma = \langle \Sigma, N, P, Z \rangle$ consists of a labeling alphabet Σ containing only terminal labels, a finite set $N \subseteq \mathscr{N}$ of nonterminals, a finite set P of straight star rules over $\Sigma \cup N$, and an *initial nonterminal* $Z \in N$. Star rules may contain multiple nodes and, hence, cannot be used directly for derivation. Instead, *star rule clones* have to be created first. A star rule clone of a rule $p = (L ::= R) \in P$ is a star rule $p' = (L^\mu ::= R^\mu)$ where μ is a multiplicity function for L and R, and R^μ does not contain multiple nodes. Clearly, every star rule clone is a star rule. The (in general infinite) set of all star rule clones of the set P is denoted by P^Δ. This allows to define the *language generated by* Γ as the set of all terminal graphs without multiple nodes that can be derived by star rule clones in a finite number of steps from a graph consisting of a single node labeled Z, i.e, $\mathscr{L}(\Gamma) = \{G \in \mathscr{G}_{\Sigma \setminus \ddot{\Sigma}} \mid Z \Rightarrow^+_{P^\Delta} G\}$.

Note that this definition of adaptive star grammars is different from [DHJ+06]. The definition in this paper actually corresponds to the notion of so-called *straight* adaptive star grammars in [DHJ+06], that is the sub-class of adaptive star grammars whose membership problem is decidable.

As an example, we consider graphs that represent arithmetic expressions with variables that are used consistently with their declaration as described in Sec. 1. Graph G_6 in Fig. 2 shows how

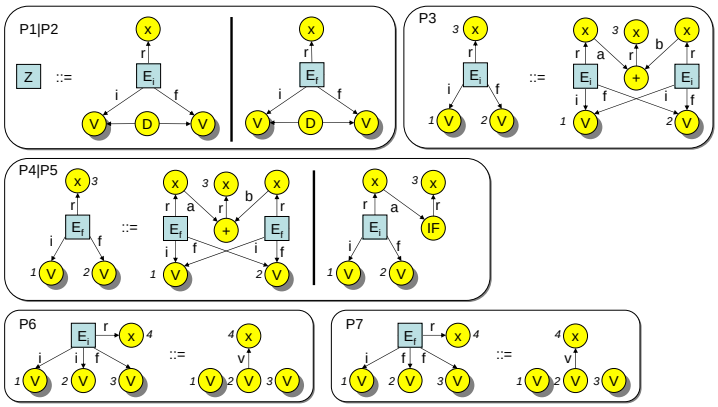

Figure 1: Star rules of an adaptive star grammar for the running example.

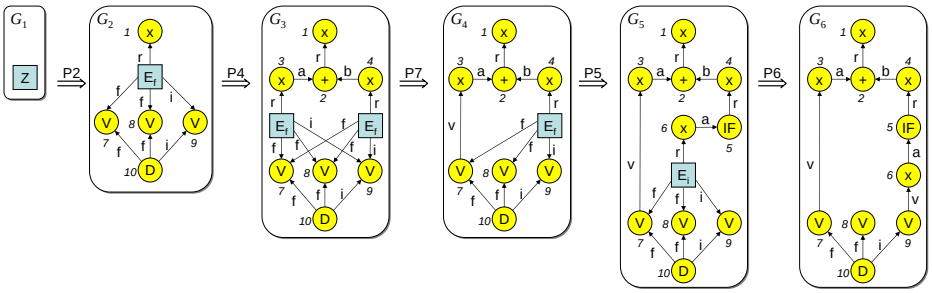

Figure 2: Sample derivation using the grammar in Fig. 1.

declarations and expressions are represented by a graph: Three variable are declared, represented by V nodes adjacent to the declaration node D. The label of the connection indicates whether it is an integer variable (i) or a float variable (f). Expressions are represented by x nodes. An x node, if it is a variable access, is connected by a v edge with a V node. Otherwise, it is connected by an r edge with the operator (+ or IF for *integer-float conversion*). Arguments of an operator are indicated by a and b edges. G_6 in Fig. 2 represents the expression $v_7 + IF(v_9)$ with integer variable v_9 and float variables v_7 and v_8.

Fig. 1 shows the rules of an adaptive star grammar for our running example. Terminal nodes are drawn as circles, nonterminal ones as squares. Multiple nodes (e.g., nodes 1 and 2 in rule P4) are drawn with a shadow. $L ::= R_1|R_2$ is a shorthand notation for two rules $L ::= R_1$ and $L ::= R_2$. Border nodes are indicated by integers, corresponding border nodes of the left-hand side and the right-hand side carry the same integers.

Nonterminals E_i and E_f represent an integer resp. float expression. The resulting x is connected by an r edge, and all accessible variables are connected by i and f edges indicating integer resp. float variables. Productions P1 and P2 create an integer resp. a float expression together with variable declarations from the initial graph. The sets of all integer and float variables are indicated by the multiple V nodes being connected by an i-labeled resp. f-labeled edge with the D node. P3 and P4 create a sum expression of type integer resp. float, whereas P5 creates a

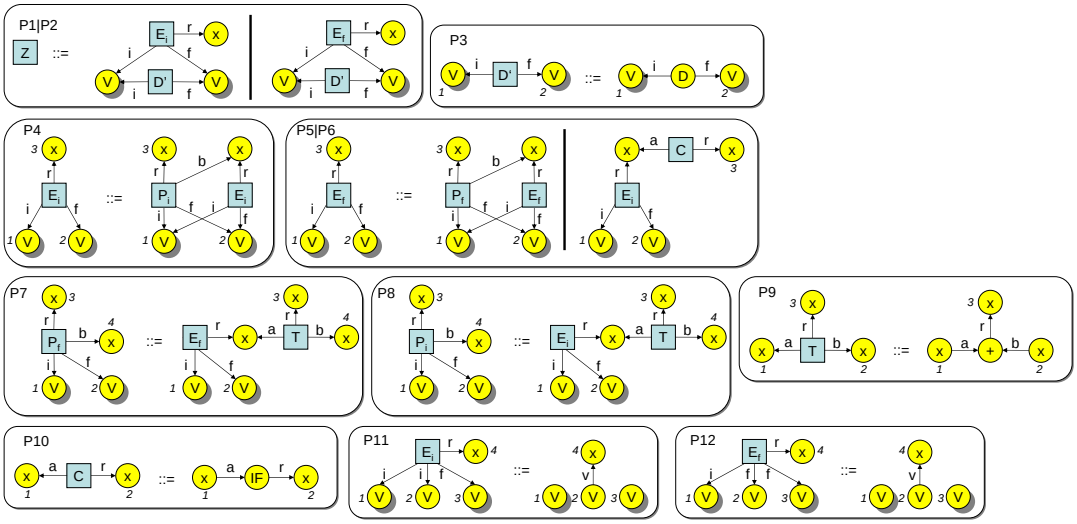

Figure 3: Star rules of the sample grammar in Fig. 1 transformed into ACNF

type conversion expression that converts an integer expression to a float expression. P6 and P7 create expressions that consists of just an access to an integer resp. float variable. Note that P6 contains two V nodes (node 1 and 2) representing integer variables (P7 is similar but with float variables). Node 1 is multiple whereas node 2 singular. That way, node 2 matches the variable being accessed whereas node 2 matches the set of all other integer variables. Therefore, it is essential that a parser must not try to match only the maximum number of all "fitting" nodes when a production contains a multiple node.

Fig. 2 shows a derivation of a graph (G_6) representing the sample expression $v_7 + IF(v_9)$ as described above. The derivation shows that the expression is well-typed and that its result is of type float. Note that the derivation does not contain any graph with multiple nodes. Multiple nodes in productions are cloned as soon as a production is applied.

4 Parsing without Isolated Nodes

The parser considered in this paper requires adaptive star grammars to be in a certain normalform, inspired by the Chomsky normalform for context-free string grammars [HU90]:

Definition 1 (Adaptive Chomsky normal form) An adaptive star grammar $\Gamma = \langle \Sigma, N, P, Z \rangle$ is in *adaptive Chomsky normal form* (*ACNF*) iff each rule $p = (L ::= R) \in P$ is of one of the following three forms: (1) *Terminal rule*: every node $n \in \dot{R}$ has a terminal label $\dot{\ell}_R(n) \in \dot{\Sigma}$. (2) *Nonterminal rule*: $R = S_1 \cup S_2$ consists of two stars S_1 and S_2 that do not share their center nodes. (3) *Chain rule*: R is a star.

An example of a grammar in ACNF is shown in Fig. 3. Rules P3, P9, P10, P11, and P12 are terminal rules. The other rules are nonterminal. Fig. 3 does not contain chain rules.

Obviously, each adaptive star grammar can be transformed into ACNF. The transformation

is similar as for context-free string grammars that are transformed into Chomsky normal form (CNF). Fig. 3 actually shows the rules after transforming the grammar in Fig. 1 into ACNF. Note that context-free string grammars in CNF contain terminal and nonterminal rules only. Chain rules are not allowed; it is always possible to transform a context-free string grammar with chain rules into an equivalent one without. It can be shown that this is not possible for every adaptive star grammar with chain rules. Therefore, ACNF has to allow chain rules.

Before parsing adaptive star grammars in ACNF that may contain rules with isolated nodes in their right-hand sides, we consider a sub-class of adaptive star grammars in ACNF in this section:

Definition 2 (Restricted adaptive Chomsky normal form) An adaptive star grammar $\Gamma = \langle \Sigma, N, P, Z \rangle$ is in *restricted adaptive Chomsky normal form* (*RACNF*) iff Γ is in ACNF and P does not contain any terminal rule with an isolated node in its right-hand side.

The grammar with rules shown in Fig. 3 is not in RACNF since rules P11 and P12 contain isolated nodes in their right-hand sides (nodes 1 and 3).

The parsing algorithm in Fig. 4 (called $*$-parser in the following) requires an adaptive star grammar $\Gamma = \langle \Sigma, N, P, Z \rangle$ in RACNF and parses a terminal graph $G \in \mathscr{G}_{\Sigma \setminus \tilde{\Sigma}}$ without isolated nodes and with a number $n = |\bar{G}|$ of edges. The algorithm uses as global data structures n "levels" $L_1, L_2, \ldots, L_n$ and a working set $Q \subseteq \bigcup_{i=1}^{n} L_i$ at every point of time. Each element of a level L_i is a pair $\langle S, E \rangle$ consisting of a star S and a set $E \subseteq \bar{G}$ with i edges of G. Each border node of S is a node of G.

The parsing algorithm is inspired by the parsing algorithm by *Cocke*, *Younger*, and *Kasami* (CYK) for context-free string grammars in Chomsky normalform [You67, Kas65]. Given a string $w = a_1 a_2 \cdots a_n$, the CYK-parser uses an $n \times n$-matrix $M = (m_{ij})_{i,j}$ as a global data structure; however, only those matrix elements m_{ij} with $i + j \leq n + 1$ are used. Each matrix element m_{ij}, after the parser has finished, contains those nonterminal symbols that can be derived to the substring $a_j a_{j+1} \cdots a_{j+i-1}$, i.e., the substring starting at character j and having length i. The levels $(L_i)_i$ of the $*$-parser correspond to the matrix elements $(m_{ij})_{i,j}$: The CYK-parser classifies substrings by their length and the index of their first character and assigns them to corresponding matrix elements. Graphs, other than strings, do not have a linear structure that allows to classify substrings by the index of their first character. Hence, the $*$-parser classifies subgraphs by the number of their edges only and, therefore, maintains an array $L_1, \ldots, L_n$ of sets instead of a matrix of sets. Each level L_i, after the parser has finished, contains those stars (together with a set of edges) that can be derived to a subgraph of G containing i edges. The specific subgraph of G that can be derived from a star in one of the levels is encoded by the set of edges that, together with the star, is actually stored in the level. Note that, if G is a member of the language of Γ, each subset $E \subseteq G$ uniquely identifies a subgraph of G since Γ is in RACNF, i.e., G must not contain isolated nodes.

The structure of the $*$-parser in Fig. 4 is similar to the structure of the CYK-parser. Step 1 creates those stars (together with their sets of edges) that can be derived to the corresponding subgraph in a single derivation step. This derivation step must use a terminal rule because nonterminal rules and chain rules produce nonterminal nodes that are not contained in G. Step 2 then fills the array of levels by combining pairs of stars and reducing them by nonterminal and

```
    Step 1:
1   for each terminal rule (L ::= R) ∈ P do
2     for each multiplicity function μ and injective morphism m : R^μ → G do
3       H := m(R^μ) and k = |H̄|;
4       if Inc_G(m(n)) ⊆ H̄ for each non-border node n of R^μ then
5         let m' be an extension of m that maps the center node of L^μ to a fresh node;
6         call procedure add(m'(L^μ), H̄)
7       end if
8     end do
9   end do

    Step 2:
10  while Q ≠ ∅ do
11    select a pair ⟨X, E_X⟩ ∈ Q and remove ⟨X, E_X⟩ from Q;
12    for i := 1 to n − |E_X| do
13      for each pair ⟨Y, E_Y⟩ ∈ L_i such that E_X ∩ E_Y = ∅ do
14        for each nonterminal rule (L ::= R) ∈ P, multiplicity function μ, and
15                  isomorphism m : R^μ → X ∪ Y do
16          if Inc_G(m(x)) ⊆ E_X ∪ E_Y for each non-border node x of R^μ then
17            let m' be an extension of m that maps the center node of L^μ to a fresh node;
18            call procedure add(m'(L^μ), E_X ∪ E_Y)
19          end if
20        end do
21      end do
22    end do
23  end do

24  procedure add(X, E)
25    { X is a star; each border node of X is a node of G; E ⊆ Ḡ is a set of edges }
26  begin
27    k := |E|;
28    if L_k does not contain a pair ⟨S, E⟩ such that S ≅ X then
29      add ⟨X, E⟩ to L_k and Q;
30      for each chain rule (L ::= R) ∈ P, multiplicity function μ,
31                and isomorphism m : R^μ → X do
32        let m' be an extension of m that maps the center node of L^μ to a fresh node;
33        call procedure add(m'(L^μ), E)
34      end do
35    end if
36  end
```

Figure 4: $*$-parser

chain rules.

In detail, the algorithm in Fig. 4 works as follows: Lines 1–9 try every terminal rule that may have created a subgraph of G. Such a subgraph must be isomorphic to a clone of the terminal rule's right-hand side. Lines 2–8 iterate over each of those clones (specified by its multiplicity function μ) and each subgraph H (specified by the image of morphism m). Line 4 checks whether H is correctly connected to the rest of G, i.e., that there is no edge from a node that does not belong to H to a node created by this rule. Parsing, i.e., applying this rule in reverse direction, would produce dangling edges otherwise. Morphism m maps R^μ and, hence, border nodes of L^μ to G. Line 5 extends morphism m to m' such that m' maps border nodes of L^μ in the same way as m, but additionally maps the center node of L^μ to a fresh, new node. The m'-image of L^μ is the star that can be derived to H by applying the current rule clone.

Procedure add(X,E) (line 24–36) is responsible of adding this star to its corresponding level if the level does not yet contain an isomorphic star that derives to the same subgraph H (line 28). Moreover, add(X,E) recursively follows all chain rules that may be applied deriving $m'(L^\mu)$ and, hence, H (line 30–34). Note that we need not check for dangling edges as in line 4 since the right-hand side of a chain rule is a star and, therefore, cannot have non-border nodes. And since chain rules do not create new terminal edges, all results from this process must be added to the same level L_k.

If procedure add(X,E) adds a new pair to a level L_i, this pair is added to working set Q, too. Q, therefore, contains exactly those pairs that have not yet been considered. Hence, step 2 iterates over Q as long as Q is not yet empty (line 10-23). $\langle X, E_X\rangle$ is the current pair. If the star of the current pair is contained in a derivation of G and derived by application of a nonterminal rule[1], it must be combined with another star Y that derives to a subgraph of G that does not have any edges in common with the subgraph that is derived from the current star X. Dangling edges would be created otherwise. This condition is checked by $E_X \cap E_Y = \emptyset$ in line 13. In a CYK-like manner, the combination of X and Y must be isomorphic to some clone of a nonterminal rule's right-hand side. Since the border nodes of X and Y are nodes of G, too, $X \cup Y$ denotes this combination. Like in step 1, the clone of the rule is represented by a multiplicity function. Again, dangling edges must be avoided. This is checked in line 16. The star that derives to $X \cup Y$ is the m'-image of L^μ where morphism m' is the extension of the isomorphism m such that the border nodes of L^μ are mapped in the same way as by m, but the center node of L^μ is mapped to a fresh new node. Procedure add(X,E), again, takes care of adding the star with its set of eventually derived edges, i.e., the disjoint union of those sets associated with the stars X and Y, to the corresponding level and considering chain rules.

Correctness of the $*$-parser can be shown (see appendix). Because of these results, a graph G without isolated nodes belongs to the language of an adaptive star grammar in RACNF iff the $*$-parser, applied to G, terminates with a pair $\langle Z, \bar{G}\rangle$ in level $L_{|\bar{G}|}$.

The $*$-parser would actually work with adaptive star grammars in plain ACNF, too, i.e., with grammars that may have isolated nodes in some of its rules' right-hand sides. However, parsing would be really expensive because of line 2 in Fig. 4. If the right-hand side R of a rule $L ::= R$ contains an isolated singular node, any node of G with a suitable label has to be considered. The number of possible morphisms, therefore, grows exponentially with the number of isolated

[1] The case that this star is derived by a chain rule is handled by procedure add(X,E).

singular nodes. The situation gets even worse if R contains an isolated multiple node. This node can be cloned arbitrarily, i.e., any subset of G nodes with a suitable label must be considered. As a consequence, the number of pairs $\langle X, E \rangle$ grows exponentially in the number of isolated nodes and exponentially in the number of G nodes.

A slight modification of the parser in the following section allows to avoid this combinatorial explosion by avoiding to explicitly consider each possibility and avoiding to create each possible pair $\langle X, E \rangle$.

5 Parsing with Isolated Nodes

We now drop the requirement of an adaptive star grammar to be in RACNF. However, we still require the grammar to be in ACNF. It can be shown that the class of adaptive star grammars in RACNF is less powerful than the class of all adaptive star grammars which is the same as the ones in ACNF. Hence, transformation of a grammar in ACNF into an equivalent one in RACNF is not possible in general. Instead, we follow the idea of avoiding the combinatorial explosion by encoding all possibilities in a single graph and dealing with such graphs instead of plain stars in the $*$-parser. This will require another transformation of the grammar. However, the encoding may be imprecise; graphs may represent proper supersets of all possibilities. The bottom-up $*$-parser, hence, will find derivations that are not correct. A subsequent top-down phase has to filter out those wrong derivations.

If a terminal rule $L ::= R$ contains isolated nodes in its right-hand side R, we consider the maximal sub-graph R' of R without isolated nodes and search for multiplicity functions μ and morphisms m as shown in line 2 of Fig. 4, however based on R' instead of R. These μ and m are called *core* multiplicity functions resp. *core* morphisms. In order to take into account the isolated nodes in R, μ and m must then be extended. We obtain different stars $m'(L^{\mu})$ with the same set $\bar{H}$. However, the set of all those stars based on the same core multiplicity function and core morphism have a common subgraph S with common center node x (up to isomorphism). The stars differ in nodes of G that may be or may be not present in the star. If a node is present, it is adjacent to the center node. We encode the set of all those stars by adding to S all nodes that are present in one of those stars and connect them with x in the same way. If the same node may be connected to x by edges with different labels in different stars, we add different copies of the node to S. Each copy is connected with x by an edge with one of these different edge labels. Each node that does not belong to the common subgraph and its incident edge are marked as *optional*. The other nodes and labels are called *mandatory* in the following.

As an example, let the graph G_6 in Fig. 2 be graph G, and P11 in Fig. 3 be the terminal rule. R' consists of nodes 2 and 4 with their incident edge labeled v. A possible core morphism maps node 2 and 4 of P11 to node 7 resp. 3 of G. The set of all possible stars has those nodes 7 and 3 as well as a new nonterminal center node with label E_i as common subgraphs. Nodes 8 and 9 of G are optional, and they can be connected with the center node either by an i or an f edge. The encoding of all of those stars is shown as star S_4 in Fig. 6. Nodes marked optional are drawn as diamonds, and edges marked optional by dashed arrows. Note that nodes 8 and 9 are contained twice each as 8_1 and 8_2 resp. 9_1 and 9_2 since both nodes can be connected with the center nodes by edges with different labels.

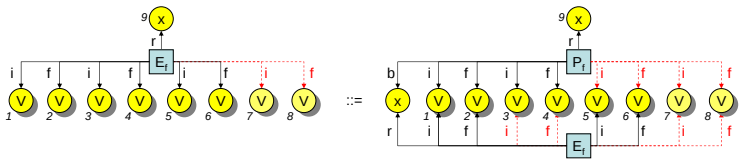

Figure 5: Transformed rule P5 of Fig. 3.

It is clear that this encoding may represent a proper superset of all possible stars because the encoding is the same for isolated singular and multiple nodes, i.e., we cannot tell from the encoding how many of the optional nodes have to be selected to obtain a member of the set of all possible stars. This problem is solved by a subsequent top-down phase after the bottom-up $*$-parser (see below).

Step 2 of the $*$-parser has to deal with those star encodings that may contain nodes and edges marked optional. And step 2 has to create new stars that encode possibly large sets of stars, too, without explicitly creating all represented stars. This is done by transforming each nonterminal rule into a rule that can be immediately applied to such encoding graphs. Step 2 then uses those transformed rules instead of the original nonterminal ones.

The transformation is explained for rule P5 in Fig. 3. The result is shown in Fig. 5. When using P5 (or its transformed version) in step 2, two stars X and Y are combined to $X \cup Y$. Without loss of generality, let X have a center node labeled P_f and Y a center node labeled E_f. Each V node adjacent to the center nodes may be marked optional. And each node, optional or mandatory, can be connected with the center node by an i edge or an f edge. Each V node in the right-hand side of P5 is connected to both nonterminal nodes by edges with the same labels. The transformed rule must be applicable to combinations of stars $X \cup Y$ where a node is optional in X and Y, or mandatory in X and Y, or optional in either X or Y, and mandatory in the other star. This is reflected in the right-hand side in Fig. 5: Instead of two multiple nodes labeled V in the original rule P5, there are eight of them. For i and f edges, each of those four combinations of optional/mandatory are represented. The left-hand side contains those eight multiple V nodes as border nodes, too. However, only those nodes resp. edges that are optional in X and Y remain optional after creating the new star from the left-hand side (node 7 and 8). This is so because a node that is mandatory in X or Y has to mandatory in the combination $X \cup Y$, too.

Note that no nodes in Fig. 5 are drawn as optional since this may cause conflicts for a combination of a node being optional and mandatory at the same time.

Step 2 now uses such transformed nonterminal rules. However, combinations of X and Y need not be isomorphic to a clone R^μ of a rule's right-hand side because $X \cup Y$ may contain optional nodes and edges that cannot be matched with R^μ. When searching for an isomorphism, the parser may drop optional components, but no mandatory ones.

When the $*$-parser terminates and a pair $\langle Z, \bar{G} \rangle$ is contained in $L_{|\bar{G}|}$, we must check whether this really indicates that G belongs to the grammar's language. This can be checked top-down by following the rules from Z to G. The $*$-parser must keep track of all derivations for that purpose. This is easily done by not only storing pairs $\langle X, E \rangle$, but also the stars that have been used when creating $\langle X, E \rangle$.

Fig. 6 shows the $*$-parser working on G_6 in Fig. 2. Actually, it shows what stars the $*$-parser

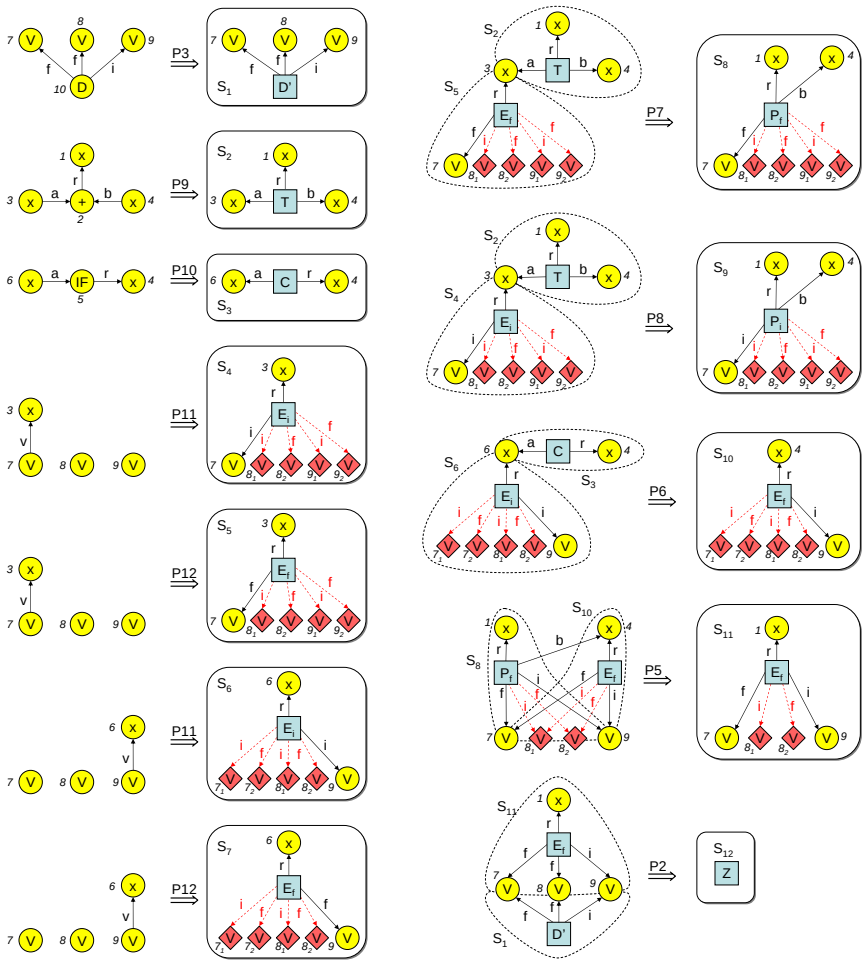

Figure 6: Reductions performed by the $*$-parser when parsing G_6 in Fig. 2.

creates, organized as pairs $U \stackrel{P_i}{\Longrightarrow} S$. The left column depicts the (maximal) subgraphs U of G matched by right-hand sides of terminal rules. The rule is indicated by P_i. S shows the created star. The right column depicts the results of step 2. Dashed areas in U indicate which stars are combined. Levels L_i that contain those stars have been omitted to avoid cluttering of the figure. The bottom right pair that contains Z as a star proves (after the top-down check) that G_6 belongs into the grammar's language.

6 Conclusions

This paper has outlined a parsing algorithm for adaptive star grammars, an extension of node and hyperedge replacement grammars. A previous paper had shown that the membership problem for a certain subclass of adaptive star grammars is decidable, however, no efficient parser had been proposed [DHJ+06]. This subclass of adaptive star grammars can be analyzed by the proposed

parser that is based on the ideas of the well-known parser by *Cocke*, *Younger*, and *Kasami* for context-free string grammars.

The parser has been implemented in Java like the parser for hyperedge replacement grammars in DIAGEN [Min02]. The DIAGEN parser generally runs in super-polynomial time, and so does the $*$-parser. But for applications in practice, the DIAGEN parser runs in polynomial time. Experiments with the $*$-parser implementation suggest a similar behavior, too. E.g., it has been applied to the running example of this paper. On a Pentium M computer with 1.6GHz, the sample graph in Fig. 2 (10 nodes and edges) required 1.5ms, and a larger expression with 400 nodes and more than 500 edges 1.4s. The measured running time was quadratic on the number of edges of the graph.

The parser will then become a component in the planned implementation of the programming language DIAPLAN [DHKM05] for shape analysis, and likewise for refactoring of object-oriented programs [HJV06].

Bibliography

[BTS00] P. Bottoni, G. Taentzer, A. Schürr. Efficient Parsing of Visual Languages based on Critical Pair Analysis and Contextual Layered Graph Transformation. In *VL '00: Proceedings of the 2000 IEEE International Symposium on Visual Languages (VL'00)*. Pp. 59–60. IEEE Computer Society, Washington, DC, USA, 2000.

[DHJ+06] F. Drewes, B. Hoffmann, D. Janssens, M. Minas, N. Van Eetvelde. Adaptive Star Grammars. In Corradini et al. (eds.), *Graph Transformations, Third International Conference, ICGT 2006, Natal, Rio Grande do Norte, Brazil, Sept. 17–23, 2006*. LNCS 4178, pp. 77–91. Springer-Verlag, Sept. 2006.

[DHK97] F. Drewes, A. Habel, H.-J. Kreowski. Hyperedge Replacement Graph Grammars. In Rozenberg (ed.), *Handbook of Graph Grammars and Computing by Graph Transformation. Vol. I: Foundations*. Chapter 2, pp. 95–162. World Scientific, Singapore, 1997.

[DHKM05] F. Drewes, B. Hoffmann, R. Klein, M. Minas. Rule-Based Programming with DiaPlan. *ENTCS* 117(1), 2005. Proc. International Workshop on Graph-Based Tools (GRABATS'04).

[Dre93] F. Drewes. Recognising k–connected hypergraphs in cubic time. *Theoretical Computer Science* 109:83–122, 1993.

[HJV06] B. Hoffmann, D. Janssens, N. Van Eetvelde. Cloning and Expanding Graph Transformation Rules for Refactoring. *ENTCS* 152(4), 2006. Proc. Graph and Model Transformation Workshop (GRAMOT'05).

[HU90] J. E. Hopcroft, J. D. Ullman. *Introduction To Automata Theory, Languages, And Computation*. Addison-Wesley, Boston, MA, USA, 1990.

[Kas65] T. Kasami. An efficient recognition and syntax analysis algorithm for context-free languages. Technical report AFCRL-65-758, Air Force Cambridge Research Laboratory, Bedford Mass., 1965.

[Min02] M. Minas. Concepts and Realization of a Diagram Editor Generator Based on Hypergraph Transformation. *Science of Computer Programming* 44(2):157–180, 2002.

[RS97] J. Rekers, A. Schürr. Defining and Parsing visual Languages with Layered Graph Grammars. *Journal of Visual Languages and Computing* 7(3), 1996/97.

[SF04] S. Seifert, I. Fischer. Parsing String Generating Hypergraph Grammars. In Ehrig et al. (eds.), *Graph Transformations*. LNCS 3256, pp. 352 – 267. Springer-Verlag, Berlin, 2004.

[SWZ99] A. Schürr, A. Winter, A. Zündorf. The PROGRES Approach: Language and Environment. In Engels et al. (eds.), *Handbook of Graph Grammars and Computing by Graph Transformation. Vol. II: Applications, Languages, and Tools*. Chapter 13, pp. 487–550. World Scientific, Singapore, 1999.

[You67] D. Younger. Recognition and parsing of context-free Languages in Time n^3. *Information and Control* 10(2):189–208, 1967.

[ZZC01] D.-Q. Zhang, K. Zhang, J. Cao. A Context-sensitive Graph Grammar Formalism for the Specification of Visual Languages. *The Computer Journal* 44(3):186–200, 2001.

A Appendix

Lemma 1 (Termination) *Given an adaptive star grammar $\Gamma = \langle \Sigma, N, P, Z \rangle$ in RACNF and a terminal graph $G \in \mathscr{G}_{\Sigma \setminus \ddot{\Sigma}}$ without isolated nodes, the $*$-parser always terminates.*

Proof. The algorithm computes stars $m'(L^{\mu})$ whose border nodes are nodes of the original graph G. Hence, the numbers of members of each set L_i is finite, and each recursive invocation of procedure *add* must terminate. The union of all levels L_i is a superset of Q, and no member is added to Q twice. Hence, the number of cycles of the while-loop in step 2 must be finite, too. The proposition now follows from the fact that each for-loop iterates over finite domains. □

Lemma 2 (Partial correctness) *Let $\Gamma = \langle \Sigma, N, P, Z \rangle$ be an adaptive star grammar in RACNF and $G \in \mathscr{G}_{\Sigma \setminus \ddot{\Sigma}}$ a terminal graph with a number $n = |\bar{G}|$ of edges, but without isolated nodes. The following proposition holds after termination of the $*$-parser for every star S without multiple nodes:*

$$S \overset{*}{\underset{P^{\Delta}}{\Longrightarrow}} G \quad \Leftrightarrow \quad \langle S, \bar{G} \rangle \in L_n$$

In order to proof this lemma, we make use of the following two elementary lemmata:

Lemma 3 (Independence of derivations 1) *Let $\Gamma = \langle \Sigma, N, P, Z \rangle$ be an adaptive star grammar, X_1, X_2 stars with disjoint edge sets and different center nodes, and G a graph such that $X_1 \cup X_2 \Rightarrow^{k}_{P^{\Delta}} G$. Then G has two subgraphs G_1 and G_2 that do not share any nonterminal node, $G = G_1 \cup G_2$, such that $X_1 \Rightarrow^{k_1}_{P^{\Delta}} G_1$, $X_2 \Rightarrow^{k_2}_{P^{\Delta}} G_2$, and $k = k_1 + k_2$.*

Proof. We proof the lemma by induction over k. The proposition is true for $k = 0$. For $k > 0$, we assume $X_1 \cup X_2 \Rightarrow^{k-1}_{P^{\Delta}} G' \Rightarrow_{P^{\Delta}} G$. By the induction hypothesis, there are two graphs G_1 and G_2 that do not share any nonterminal node, $G' = G_1 \cup G_2$, such that $X_1 \Rightarrow^{k_1}_{P^{\Delta}} G_1$, $X_2 \Rightarrow^{k_2}_{P^{\Delta}} G_2$, and $k - 1 = k_1 + k_2$. $G' \Rightarrow_{P^{\Delta}} G$ is defined by replacing a star in G' by a graph. As G_1 and G_2 do not share any nonterminal node, either $G_1 \Rightarrow_{P^{\Delta}} G'_1$ and $G = G'_1 \cup G_2$, or $G_2 \Rightarrow_{P^{\Delta}} G'_2$ and $G = G_1 \cup G'_2$. □

Lemma 4 (Independence of derivations 2) *Let $\Gamma = \langle \Sigma, N, P, Z \rangle$ be an adaptive star grammar, X_1, X_2 stars with disjoint edge sets and different center nodes, and G_1, G_2 arbitrary graphs such that $X_1 \Rightarrow^{*}_{P^{\Delta}} G_1$ and $X_2 \Rightarrow^{*}_{P^{\Delta}} G_2$. Then $X_1 \cup X_2 \Rightarrow^{*}_{P^{\Delta}} G_1 \cup G_2$.*

Proof. We proof the lemma by induction over $n = n_1 + n_2$ in $X_1 \Rightarrow^{n_1}_{P^{\Delta}} G_1$ and $X_2 \Rightarrow^{n_2}_{P^{\Delta}} G_2$. The proposition is true for $n = 0$. Without loss of generality, assume $n_1 > 0$ if $n > 0$, and $X_1 \Rightarrow^{n_1 - 1}_{P^{\Delta}} G'_1 \Rightarrow_{P^{\Delta}} G_1$, $X_2 \Rightarrow^{n_2}_{P^{\Delta}} G_2$. $X_1 \cup X_2 \Rightarrow^{*}_{P^{\Delta}} G'_1 \cup G_2$ follows from the induction hypothesis, and $G'_1 \cup G_2 \Rightarrow_{P^{\Delta}} G_1 \cup G_2$ from the definition of star replacements. □

Proof of Lemma 2. We assume $S \Rightarrow^{k}_{P^{\Delta}} G$ for some $k \geq 1$ first and prove $\langle S, \bar{G} \rangle \in L_n$ by induction over k. If $k = 1$, $\langle S, \bar{G} \rangle$ is added to L_n in step 1. For $k > 1$, consider the derivation sequence

$$S \overset{}{\underset{P^{\Delta}}{\Longrightarrow}} G' \overset{k-1}{\underset{P^{\Delta}}{\Longrightarrow}} G. \tag{1}$$

The first step in (1) cannot use a terminal rule as $k-1>0$ and, hence, G' must contain a non-terminal node. If the first step in (1) uses a nonterminal rule, there are two stars X_1 and X_2 that do not share their center nodes such that $G' = X_1 \cup X_2$. By lemma 3, there are two graphs G_1 and G_2, such that $G = G_1 \cup G_2$, $X_1 \Rightarrow_{P^\Delta}^{k_1} G_1$, $X_2 \Rightarrow_{P^\Delta}^{k_2} G_2$, and $k_1 + k_2 = k-1$. By the induction hypothesis, $\langle X_1, \bar{G}_1 \rangle \in L_{|\bar{G}_1|}$ and $\langle X_2, \bar{G}_2 \rangle \in L_{|\bar{G}_2|}$. Without loss of generality, $\langle X_1, \bar{G}_1 \rangle$ has been added to Q when $\langle X_2, \bar{G}_2 \rangle$ was already member of $L_{|\bar{G}_2|}$. Therefore, $\langle S, \bar{G} \rangle$ is added to L_n in step 2. If, however, the first step in (1) uses a chain rule, G' is a star, and by the induction hypothesis, $\langle G', \bar{G} \rangle \in L_n$. Hence, $\langle S, \bar{G} \rangle$ is added to L_n in the procedure invocation add$(G', \bar{G})$.

We now assume $\langle S, \bar{G} \rangle \in L_n$ and prove $S \Rightarrow_{P^\Delta}^* G$ by induction over $n = |\bar{G}|$. Since G does not have isolated nodes, $n > 0$. If $n = 1$, $\langle S, \bar{G} \rangle$ must have been added to L_n by invocation of add$(S, \bar{G})$ that has been recursively invoked by the first non-recursive invocation of add$(S', \bar{G})$ in step 1. Hence, $S \Rightarrow_{P^\Delta}^* S' \Rightarrow_{P^\Delta} G$. If $n > 1$, $\langle S, \bar{G} \rangle$ has been added to L_n either in the same way as described for $n = 1$, and hence $S \Rightarrow_{P^\Delta}^* G$, or by invocation of add$(S, \bar{G})$ that has been recursively invoked by the first non-recursive invocation of add$(S', \bar{G})$ in step 2. In the latter case, there are $\langle X_1, E_1 \rangle \in L_{|E_1|}$ and $\langle X_2, E_2 \rangle \in L_{|E_2|}$ such that $E_1 \cap E_2 = \emptyset$, $|E_1| + |E_2| = n$, and $S' \Rightarrow_{P^\Delta} X_1 \cup X_2$. By the induction hypothesis, there are graphs G_1 and G_2 with $\bar{G}_1 = E_1$, $\bar{G}_2 = E_2$, $X_1 \Rightarrow_{P^\Delta}^* G_1$, and $X_2 \Rightarrow_{P^\Delta}^* G_2$. And by lemma 4, $X_1 \cup X_2 \Rightarrow_{P^\Delta}^* G_1 \cup G_2$ where $\bar{G}' = E_1 \cup E_2 = \bar{G}$. $G = G'$ follows from the fact that G does not have isolated nodes. □