# Adaptive Star Grammars*

Frank Drewes[1], Berthold Hoffmann[2], Dirk Janssens[3],
Mark Minas[4], and Niels Van Eetvelde[3] **

[1] Umeå universitet, Sweden
[2] Universität Bremen, Germany
[3] Universiteit Antwerpen, Belgium
[4] Universität der Bundeswehr München, Germany

**Abstract.** We propose an extension of node and hyperedge replacement grammars, called adaptive star grammars, and study their basic properties. 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. We prove that cloning can be applied lazily. Unrestricted adaptive star grammars are shown to be capable of generating every type-0 string language. However, we identify a reasonably large subclass for which the membership problem is decidable.

## 1 Introduction

Software engineering tools for model transformation or refactoring do often represent models and programs by graphs. Our earlier research in this area [1] revealed that the structure of such graphs cannot be captured by graph schemas, because models and programs have a recursive syntactical structure. Graph grammars are among the most natural candidates for specifying for specifying recursively structured graphs. For example, a graph grammar could be designed to generate the set of all program graphs as defined in [1].

The purpose of this paper is to introduce adaptive star grammars and to study their basic properties. Being context-free devices with nice computational properties, hyperedge and node replacement grammars [2–4] have proven particularly useful for defining graph languages. Unfortunately, these types of graph grammars turn out to be too weak to generate program graphs in a reasonable way. Therefore, we propose an extension, called adaptive star grammar, that is not only able to capture the context-free structure of object-oriented programs, but also aspects such as scope rules, overriding of methods, and references of variable and parameter uses to their definitions.

A star rule is a rule which replaces a nonterminal node together with its outgoing edges – a *star* – with another graph. This graph is glued to the *border*

* Supported by SeGraVis (www.segravis.org), a European research training network.
** On leave to Universität Bremen on a SeGraVis grant (October 2005–January 2006).

*nodes* of the star, i.e., to the nodes pointed to by the outgoing edges of the nonterminal node. The replacement process is similar to the well-known notion of hyperedge replacement, where the nonterminal node corresponds to the hyperedge being replaced. To increase the generative power of the device, border nodes of the left-hand side of a star rule may be designated as so-called multiple nodes. These nodes can be *cloned* prior to the application of the star rule. Cloning simply replicates a multiple node together with its incident edges any number of times (including 0). Thus, a star rule containing multiple nodes is actually a rule schema. In fact, even the host graph may contain multiple nodes, and these can be cloned as well in order to make a rule applicable.

We note here that the set nodes of PROGRES [5] and FUJABA [6] are similar to our multiple nodes. In the model transformation language GMORPH [7], a more general notion of cloning is provided whose *collection containers* correspond to the notion of a *multiple subgraph*. A similar concept is addressed in [8].

As our first main result, we show that cloning can be applied in both an eager and a lazy manner. Thus, derivations can be carried out effectively. Our second and third results concern the generative power of adaptive star grammars and the membership problem. Unrestricted adaptive star grammars can generate all recursively enumerable string languages (encoded as chain graphs in the usual way). Thus, these grammars are too powerful if given structures need to be parsed. However, in our third main result, we identify a reasonably large class of adaptive star grammars for which membership is decidable.

The structure of this paper is as follows. In the next section, we define the basic notions regarding stars and star replacement. Section 3 introduces the cloning operation. Based on this, adaptive star grammars are introduced in Section 4. In this section we also discuss a nontrivial example that applies adaptive grammars to generate program graphs. Two derivation strategies, eager and lazy cloning, are studied in Section 5 and demonstrated on the example. In Section 6, the generative power and the membership problem of adaptive star grammars are investigated. Section 7 concludes the paper.

## 2 Star Replacement

We start by defining the type of graphs considered in this paper. Throughout the paper, let $\mathbf{\Sigma}$ be a set of labels which is partitioned into two disjoint, countably infinite sets $\dot{\mathbf{\Sigma}}$ and $\bar{\mathbf{\Sigma}}$ of node and edge labels, resp. A finite subset $\Sigma$ of $\mathbf{\Sigma}$ is called a labeling alphabet. Its two components are $\dot{\Sigma} = \Sigma \cap \dot{\mathbf{\Sigma}}$ and $\bar{\Sigma} = \Sigma \cap \bar{\mathbf{\Sigma}}$.

Intuitively, in the type of grammars to be defined later on, stars are the nonterminal items to be replaced. Therefore, we reserve an infinite supply $\mathcal{N} \subseteq \dot{\mathbf{\Sigma}}$ of node labels called nonterminals. (We assume that the remaining set $\dot{\mathbf{\Sigma}} \setminus \mathcal{N}$ of terminal labels is infinite as well.) In the following definition of graphs, we prohibit edges that point to nonterminal nodes. In particular, nonterminal nodes cannot be connected by edges. In this way, stars become a generalised version of the hyperedges known from hyperedge replacement grammars [2, 3].

**Definition 1 (Graph).** 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 $\dot{\ell}_G \colon \dot{G} \to \dot{\mathbf{\Sigma}}$ and $\bar{\ell}_G \colon \bar{G} \to \bar{\mathbf{\Sigma}}$. 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 $\Sigma$ is denoted by $\mathcal{G}_\Sigma$.

We use common terminology regarding graphs. For instance, an edge is said to be *incident* with its source and target nodes, and makes these nodes *adjacent* to each other. 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. The notation $G \cong_m H$ denotes the fact that graphs $G$ and $H$ are isomorphic via the isomorphism $m$.

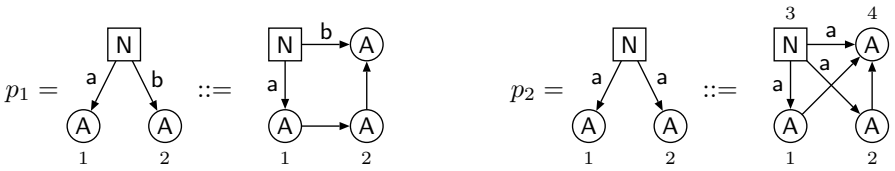
Next, we define a central notion of this paper, the *star*.

**Definition 2 (Star).** For a graph $G$ and a node $x \in \dot{G}$, $G(x)$ denotes the subgraph of $G$ consisting of $x$, all its incident edges, and all its adjacent nodes. A graph of the form $G(x)$ is a *star* if $\dot{\ell}_G(x) \in \mathcal{N}$. In this case, $G(x)$ is also called a *star occurrence in* $G$.

Thus, a star is a graph $S$ that consists of a nonterminal node $x$ and its adjacent nodes. In the following, these will be called the *center node* of $S$ and the *border nodes* of $S$, resp. The edges are called the *arms* of $x$. By Definition 1, 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.

**Definition 3 (Star Rule).** A *star rule* $S ::= R$ consists of a star $S$, called its *left-hand side*, and a graph $R$, called its *right-hand side*, that share precisely the border nodes of $S$. When we modify such a rule, it is considered to be a single graph, namely the union of $S$ and $R$.

*Example 1 (Star Rules).* Two examples of star rules are shown below:




Nonterminal nodes are drawn as boxes; they have two border nodes labeled A, and two arms. For both rules, the border nodes 1 and 2 are drawn twice: they belong both to the right-hand side and to the left-hand side.

**Definition 4 (Star Replacement).** Let $G$ be a graph, and $p = (S ::= R)$ a star rule so that $S \cong_m G(x)$ for some node $x \in \dot{G}$. 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 $S$ in $R$ with its image $m(b) \in \dot{G}$. In this situation, we also write $G \Rightarrow_{x,p,m} H$.

Obviously, star replacement is a restricted form of DPO graph transformation [9] (with injective occurrence morphisms). In fact, star replacement is more or less equivalent to hyperedge replacement [2, 3], because the center node of a star together with its arms can be seen as a hyperedge.

Star replacement does not cover node replacement [4], as the left-hand side of a star rule has a fixed number of arms, whereas nonterminals in node-replacement grammars can be replaced independently of the number of edges incident with them. The notion of cloning introduced in the next section is a formal mechanism that makes it possible to overcome this limitation of star replacement. Rules are specified in a generic way so that they adapt to several contexts of a nonterminal, but not necessarily to all. Next, we formalize the adaptation process, which we call cloning, and then we use it to define adaptive star grammars.

## 3 Cloning

In this section, we formalize the notion of cloning. We use a special set of labels designating so-called multiple nodes. A similar mechanism can be found in the PROGRES graph transformation language [5].

Formally, we assume from now on that $\dot{\mathbf{\Sigma}} \setminus \mathcal{N}$ contains a subset $\ddot{\mathbf{\Sigma}}$ of *multiple node labels*. The remaining node labels are said to be *singular* ones. Further, we assume that there is a bijection $\ddot{}: \dot{\mathbf{\Sigma}} \setminus (\mathcal{N} \cup \ddot{\mathbf{\Sigma}}) \to \ddot{\mathbf{\Sigma}}$. Thus, 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. The set of multiple nodes in a graph $G$ is denoted by $\ddot{G}$, i.e., $\ddot{G} = \{v \in \dot{G} \mid \dot{\ell}_G(v) \in \ddot{\mathbf{\Sigma}}\}$. In figures, we draw multiple nodes as nodes with a "shadow", as is seen in Definition 6.

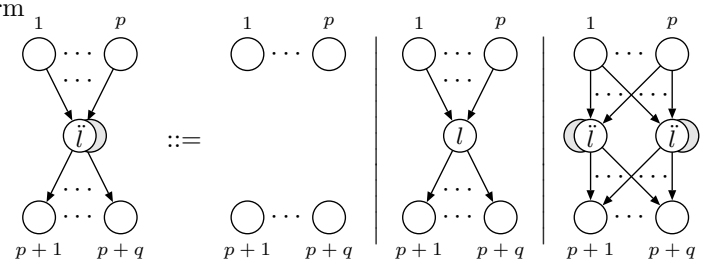
We can now define the cloning operation. Using this operation, a multiple node can be turned into any number of singular nodes, its clones. However, we also want to be able to create clones that are multiple nodes. Thus, we define $G\frac{x}{(m,n)}$ to be obtained from $G$ by replacing the multiple node $x$ with $m$ clones which are still multiple, and $n$ singular clones.

**Definition 5 (Cloning Operation).** Let $G$ be a graph, $x \in \ddot{G}$ a multiple node, and $m, n \geq 0$. The *clone* $G\frac{x}{(m,n)}$ is the graph constructed as follows. 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 $G \setminus \{x\}$, $m$ copies of $G(x)$, and $n$ copies of $G'(x)$. Finally, identify the $m + n + 1$ copies of each node in $\dot{G}(x) \setminus \{x\}$ with each other.

The $m+n$ copies of $x$ in $G\frac{x}{(m,n)}$ are called the clones of $x$. Obviously, $G\frac{x}{(m,n)}$ is defined only up to isomorphism. However, $G \setminus \{x\}$ is of course isomorphic to the subgraph of $G\frac{x}{(m,n)}$ induced by the nodes that are not clones of $x$.

The process of cloning can be described by graph transformation. The rules in the following definition should be considered as rules in the DPO approach, where the interface graph is the discrete graph consisting of the nodes $1, \ldots, p+q$.

**Definition 6 (Cloning Rules).** The set $\Delta$ of *cloning rules* consists of all rules of the form




for all $\ddot{l} \in \ddot{\Sigma}$ and all $p, q \geqslant 0$.[5] The three rule schemas will be denoted by $rem(\ddot{l})$, $sing(\ddot{l})$, and $copy(\ddot{l})$, respectively.

The application of a cloning rule performs a cloning operation, in which a multiple node is either removed, turned into a singular node, or copied. It should be clear that the cloning rules suffice to describe all clonings. More precisely, let $G$ be a graph containing a multiple node $x$. For all $m, n \geqslant 0$, $G\frac{x}{(m,n)}$ is derived from $G$ using the cloning rules, as follows: $G\frac{x}{(0,0)}$ is obtained by an application of $rem(\ddot{l})$. Moreover, for $m+n > 0$, $G\frac{x}{(m,n)}$ is obtained by $m+n-1$ applications of $copy(\ddot{l})$ and $n$ applications of $sing(\ddot{l})$.

The result obtained by cloning a number of nodes is independent of the order in which those nodes are treated.

**Lemma 1 (Cloning is Commutative).** *For a graph $G$ with distinct multiple nodes $x$ and $y$, and for $m, n, m', n' \geq 0$,*

$$\left(G\frac{x}{(m,n)}\right)\frac{y}{(m',n')} \cong \left(G\frac{y}{(m',n')}\right)\frac{x}{(m,n)}.$$

*Proof.* Obviously, if two rules in $\Delta$ are applied to distinct multiple nodes of $G$, the result does not depend on the order of these rule applications. As argued above, $\Delta$ describes cloning correctly. This yields the statement. □

We define a cloning operation for a 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.

**Definition 7 (Iterated Cloning).** Let $G$ be a graph. A *multiplicity function* for $G$ is a function $\mu\colon \ddot{G} \to \mathbb{N}^2$. If $\ddot{G} = \{x_1, x_2, \ldots, x_k\}$ (where $x_1, \ldots, x_k$ are pairwise distinct), then $G^\mu$ is the graph defined by

$$G^\mu = \left(\ldots\left(\left(G\frac{x_1}{\mu(x_1)}\right)\frac{x_2}{\mu(x_2)}\right)\ldots\frac{x_k}{\mu(x_k)}\right).$$

By Lemma 1, $G^\mu$ is defined uniquely up to isomorphism. In the following, when defining a multiplicity function $\mu$, we will specify only those multiplicities $\mu(x)$ which are not equal to $(1, 0)$.

[5] The labels of the nodes $1, \ldots, p+q$ as well as the edge labels have been omitted to avoid cluttering the figure. They carry over from the left-hand side to the right-hand sides in the obvious way. Note also that the nodes $1, \ldots, p+q$ may be multiple.

## 4 Adaptive Star Grammars

In this section, we define adaptive star grammars. The rules of these grammars are star rules which may contain multiple nodes that can be cloned before a rule is applied. The graphs being derived may contain multiple nodes as well, and so they may also be cloned in order to make a rule applicable. Let us first define the cloning of (nodes in) star rules.

**Definition 8 (Star Rule Clone).** Let $p = (S ::= R)$ be a star rule. A *star rule clone* of $p$ is a star rule $p''$ such that $p \Rightarrow^*_\Delta p'$ for some $p'$ from which $p''$ can be obtained by taking a *quotient*, i.e., identifying pairs of border nodes (that have the same label). The set of all star rule clones of a set $P$ of star rules is denoted by $P^\Delta$.

Note that neither edges nor non-border nodes are identified by taking quotients. Clearly, every star rule clone is a star rule. We can now define adaptive star grammars and the graph languages they generate.

**Definition 9 (Adaptive Star Grammar).** An *adaptive star grammar* $\Gamma = \langle \Sigma, N, P, Z \rangle$ consists of

- a labeling alphabet $\Sigma$ containing only terminal labels,
- a finite set $N \subseteq \mathcal{N}$ of nonterminals,
- a finite set $P$ of star rules over $\Sigma \cup N$ with straight left-hand sides, and
- an *initial nonterminal* $Z \in N$.

The *language generated by* $\Gamma$ is $\mathcal{L}(\Gamma) = \{G \in \mathcal{G}_{\Sigma \setminus \ddot{\Sigma}} \mid Z \Rightarrow^+_{\Delta P} G\}$. Here, $Z$ denotes the graph consisting of a single node labeled $Z$, and $\Delta P = \Delta \cup P^\Delta$.

Thus, derivation steps in adaptive star grammars can be of two different types: On the one hand, multiple nodes in the host graph can be cloned, and, on the other hand, star rule clones can be applied.

We now discuss a particular application of star grammars.

*Example 2 (A grammar for program graphs).* As a nontrivial example we now discuss a star grammar modeling the structure of object-oriented programs by generating graphs called *program graphs*. This type of graphs has been developed for studying refactoring in [1]. Due to space restrictions, only a simplified method body specification is considered here, where method bodies contain only assignments and method calls. The grammar is shown in Fig. 1. A more complete specification based on star grammars can be found in [10].

We use terminal node labels $B, E, V, M$ that correspond to method body root, entity occurrence, variable, and method, respectively. Furthermore we have nonterminal node labels $BODY, STS, ST, EXP, ACC, ASS, CALL, APS$. The label $BODY$ is the initial nonterminal. It generates an $STS$ star (statement sequence), connected to a $B$ node, and to a multiple $N$ node. The latter is a shorthand covering both $V$ and $M$. From the modeling point of view, it can be seen as a

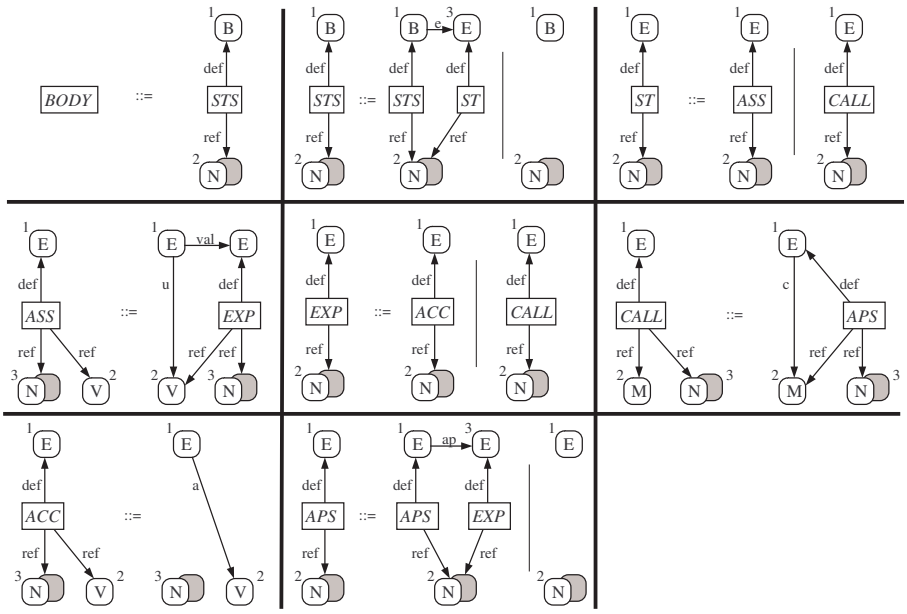



**Fig. 1.** A star grammar method body syntax tree specification

supertype of nodes of type $V$ and $M$. Any singular node cloned out of a multiple $N$ node becomes a node of type $V$ or $M$. Alternatively, to stay within the formalism of adaptive star grammars, the rules of the grammar can be modified by applying $copy(N)$ to each of the $N$ nodes and relabeling the two clones into a $V$ node and an $M$ node.

The $STS$ star generates recursively a number of statements ($ST$), each of which can be rewritten into an assignment ($ASS$) or a method call ($CALL$). The right-hand side of an assignment is an expression ($EXP$). Expressions are either calls or variable accesses ($ACC$). Calls can have actual parameters (generated by $APS$) which are expressions.

The edge labels are e, a, u, c, val, ap, def, ref. The first six of these stand for syntax tree expression, variable access, variable update, method call, assignment value and actual parameter respectively. The edge labels def and ref are used for the arms of nonterminal nodes. The body root node $B$ groups a set of $E$ nodes, connected by e edges (cf. Fig. 2). Each of these nodes represents an occurrence of a variable, or a method call in the syntax tree. In the first case it is connected by an outgoing a or u edge to a variable and in the second case by a c edge to a method. Assignment and call occurrences may have additional val or ap edges to other $E$ nodes.

Every nonterminal has a ref arm. It is always connected to a multiple node of type $N$ representing all the *referable* symbols (visible methods, variables, formal parameters, types) that can be used in the program part derived from the non-

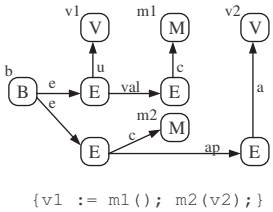



**Fig. 2.** Method body graph example

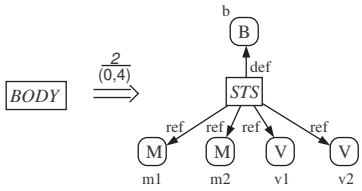



**Fig. 3.** Eager cloning derivation

terminal. The *ACC* star has three arms: the def arm shows the start node of the statement. The two others are ref arms. This has the effect of selecting out of the complete set of referred elements one particular variable that is of particular relevance, all the other referable elements are represented by the multiple node. The nonterminal node disappears and an a edge is created between the *E* node and the *V* node (cf. Fig 1). The *CALL* and *ASS* rules are similar but create c and u edges and an additional *EXP* nonterminal.

An example of a method body graph that can be generated by the grammar is given in Figure 2, together with its textual equivalent. The node identifiers relate the program entities with their graphical representations.

By using adaptive star grammars instead of context-free string grammars for generating models of object-oriented programs, typical properties of object-oriented languages, such as the fact that every use of an identifier has a matching declaration, can be modeled. This is realized by the ref arms, which record the existing entities that can be used in a function. It can also be shown that the use of adaptive star grammars makes it possible to enforce the visibility constraints for class attributes, i.e. to ensure that method bodies in derived program graphs never contain accesses to private attributes of other classes. However, adaptive star grammars seem to be unable to cope with more complex constraints like the parameter correspondence. It does not seem to be possible to generate exactly as many actual parameters as formal parameters and make their types match.

## 5 Eager and Lazy Cloning

In this section, we will study an important aspect regarding derivations, namely the interplay between cloning and rule application. Cloning can be performed eagerly, where cloning on the host graph is done as early as possible. The following lemma shows that when a star replacement is followed by a cloning step in which one of the nodes of the host graph is cloned, then this results in the same graph as the one obtained by performing the cloning operation before the star replacement. If the cloned node is a border node of the star occurrence that is replaced, then an appropriate cloning of the rule used is needed. The straightforward proof is omitted.

**Lemma 2 (Eager Cloning).** *Let $G$, $H$ be graphs, let $p$ be a star rule, and let $G \Rightarrow_{x,p,m} H$ be a star replacement. Let $y \in \ddot{G}$ and let $k, l \geqslant 0$.*

1. *If $y$ is not a border node of $G(x)$, then $G\frac{y}{(k,l)} \Rightarrow_{x,p,m} H\frac{y}{(k,l)}$.*
2. *If $y$ is a border node of $G(x)$, then let $\tilde{y} = m^{-1}(y)$ and $\tilde{p} = p\frac{\tilde{y}}{(k,l)}$. Let $\tilde{m}$ be an extension of $m$, mapping the $k+l$ clones of $\tilde{y}$ bijectively to the $k+l$ clones of $y$. Then $G\frac{y}{(k,l)} \Rightarrow_{x,\tilde{p},\tilde{m}} H\frac{y}{(k,l)}$.*

Consequently, we obtain a normalform $nf(P)$ of a set $P$ of star rules, such that $nf(P)$ is the subset of all star rule clones in $P^{\Delta}$ that do not contain multiple nodes. In particular, rule application creates only singular nodes.

*Example 3 (Eager cloning in program graphs).* The example program of Figure 2 can be derived using eager cloning. We consider the *STS* star in Figure 3. It is obtained by executing the initial rule being cloned by $\frac{2}{(0,4)}$ to generate all the needed variables and methods in advance. From that point, the star rules have to be cloned by $\frac{3}{(0,3)}$ for the *CALL*, *ACC* and *ASS* rules and $\frac{2}{(0,4)}$ for the other rules.

**Corollary 1.** *For every adaptive star grammar $\Gamma = \langle \Sigma, N, P, Z \rangle$, it holds that $\mathcal{L}(\Gamma) = \{G \in \mathcal{G}_{\Sigma \setminus \ddot{\Sigma}} \mid Z \Rightarrow^{+}_{nf(P)} G\}$.*

When constructing a derivation of an adaptive star grammar, it would obviously be desirable to postpone cloning as much as possible: we use incremental cloning rules to construct derivations so that cloning is kept at a minimum. In order to characterize which clonings can be postponed until after a given star replacement, the following auxiliary notion is useful.

**Definition 10 (Indistinguishability).** Let $\tilde{p} = (\tilde{S} ::= \tilde{R}) \in P^{\Delta}$ be a quotient of a rule $p'$ that is obtained by cloning a rule $p = (S ::= R) \in P$. For a border node $y$ of $\tilde{S}$, its set of *precursors* is the set of nodes $x$ of $S$ such that $y$ is the image of either $x$ or a clone of $x$ under the quotient. Two border nodes $y_1, y_2$ of $\tilde{S}$ are *indistinguishable* (in $\tilde{p}$) if they have the same set of precursors in $S$ and there exists an automorphism of $\tilde{p}$ that interchanges them while leaving the other nodes invariant.

**Definition 11 (Lazy Cloning).** Let $\tilde{p} = (\tilde{S} ::= \tilde{R}) \in P^{\Delta}$. A derivation $G \Longrightarrow^{*}_{\Delta} \tilde{G} \Longrightarrow_{x,\tilde{p},m} H$ constitutes a *lazy step* if only border nodes of $G(x)$ are cloned in $G \Longrightarrow^{*}_{\Delta} \tilde{G}$, and, moreover, there do not exist distinct border nodes $y_1, y_2$ of $\tilde{S}$ such that $y_1$ and $y_2$ are indistinguishable in $\tilde{p}$, and $m(y_1)$ and $m(y_2)$ are clones of the same node of $G(x)$.

Note that $\tilde{p}$ can in general be obtained in different ways from a rule $p \in P$, and hence the notion of a lazy step is defined only with respect to a fixed choice of $p$, $p'$ and a quotient map. However for our purposes it is sufficient to consider a step as lazy if there exists such a choice.

The next result shows that lazy cloning is correct: every derivation can be rearranged into a sequence of lazy steps followed by a number of cloning steps.

**Theorem 1 (Correctness of Lazy Cloning).** *Let $P$ be a set of star rules. For every derivation $G \Longrightarrow^*_{\Delta P} H$ there is a graph $\hat{H}$ that can be derived from $G$ by a sequence of lazy steps, such that $\hat{H} \Longrightarrow^*_{\Delta} H$.*

*Proof.* It suffices to consider a derivation of the form $G \Longrightarrow^n_{\Delta} \tilde{G} \Longrightarrow_{x,\tilde{p},m} H$ which is not a lazy step, and to prove that one of its cloning steps can be postponed until after the star replacement. For this purpose, assume that $\tilde{p} = (\tilde{S} ::= \tilde{R})$ is a quotient of some rule obtained from $p = (S ::= R) \in P$ by a sequence of cloning steps.

If $G \Longrightarrow^n_{\Delta} \tilde{G}$ clones a node that is not a border node of $G(x)$, then we may assume that the corresponding step is the last step of $G \Longrightarrow^n_{\Delta} \tilde{G}$ (see Lemma 1). This step is parallel independent of the star replacement, and thus the desired result follows from well-known results about DPO graph rewriting. So assume that all steps in $G \Longrightarrow^n_{\Delta} \tilde{G}$ clone border nodes of $G(x)$. By assumption, $G \Longrightarrow^n_{\Delta} \tilde{G} \Longrightarrow_{x,\tilde{p},m} H$ is not a lazy step. Hence, there exist distinct nodes $y_1, y_2$ of $\tilde{S}$, a multiple border node $z$ of $G(x)$ and two clones $z_1, z_2$ of $z$ in $\tilde{G}$ such that $y_1$ and $y_2$ are indistinguishable, $m(y_1) = z_1$, and $m(y_2) = z_2$. If at least one of $z_1$, $z_2$ is singular, then it is obtained by an application of $sing(\ddot{l})$, and obviously this step can be postponed until after the star replacement. So assume that both $z_1$ and $z_2$ are multiple. Again by Lemma 1 one may assume that the cloning step that produces $z_2$ is the last step of $G \Longrightarrow^n_{\Delta} \tilde{G}$. Moreover, since both $z_1$ and $z_2$ are clones of $z$, $z_2$ can be obtained as a clone of $z_1$. Thus, $G \Longrightarrow^{n-1}_{\Delta} G' \Longrightarrow_{\Delta} \tilde{G}$, where $G'$ is the graph obtained from $\tilde{G}$ by deleting $z_2$ and its incident edges, and $\tilde{G} = G'\frac{z_1}{(2,0)}$. Now let $p' = (S' ::= R')$ and $H'$ be obtained from $\tilde{p}$ and $H$ by deleting $y_2$, $z_2$ and their incident edges, respectively. Then $G' \Longrightarrow_{x,p',m'} H'$, where $m'$ is the restriction of $m$ to $S'$. Moreover, $\tilde{p} = p'\frac{y_1}{(2,0)}$, because $y_1$ and $y_2$ are indistinguishable, and it follows from the definition of a star replacement that $H = H'\frac{z_1}{(2,0)}$. The result follows. □

*Example 4 (Lazy cloning in program graphs).* A lazy derivation of the first statement `v1 := m1()` of the method body example would, after applying the initial rule, immediately apply the *STS* and *ST* rule without cloning to arrive at the *ASS* nonterminal. The rest of the derivation is shown in Fig. 4. To clarify which of the possible rules is used for star replacement, rule names carry an index. Note that all star replacements together with the preceding cloning steps are lazy steps.

## 6 The Membership Problem

This section consists of two parts. In the first part, we show that adaptive star grammars can generate every recursively enumerable string language. We will do this by sketching how to simulate a slightly modified version of the well-known counter machines. Hence, in particular, the membership problem is unsolvable. In the second part of the section, a restriction is studied under which this problem becomes decidable.

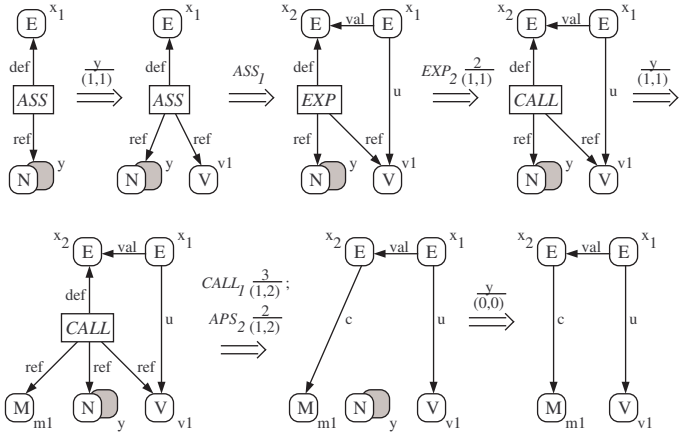



**Fig. 4.** A lazy derivation for a call statement

Let us first define a variant of (nondeterministic) counter machines that turns out to be particularly suitable for our situation. An *offline counter machine* (OCM, for short) with $k \geqslant 1$ counters is a system $M = (Q, \mathcal{A}, I, q_0, F)$ consisting of a finite set $Q$ of states, a finite input alphabet $\mathcal{A}$, a finite set $I$ of instructions, an initial state $q_0 \in Q$, and a set $F \subseteq Q$ of final states. Each instruction has the form $(q, i, z) \mapsto (q', j)$, where $q, q' \in Q$, $1 \leqslant i \leqslant k$, and $(z, j) \in \{(\text{ZERO}, +1), (\text{NONZERO}, -1), (\text{NONZERO}, +1)\}$.

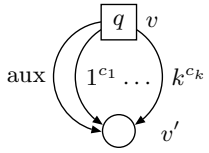
A configuration $(q, c_1 \cdots c_k) \in Q \times \mathbb{N}^k$ consists of a state $q$ and $k$ counter values $c_1, \ldots, c_k$. There is a computation step $(q, c_1 \cdots c_k) \mapsto_M (q', c_1' \cdots c_k')$ if $I$ contains an instruction $(q, i, z) \mapsto (q', j)$ with $z = \text{ZERO} \iff c_i = 0$ and

$$c_l' = \begin{cases} c_l + j \text{ if } l = i \\ c_l \qquad \text{otherwise.} \end{cases}$$

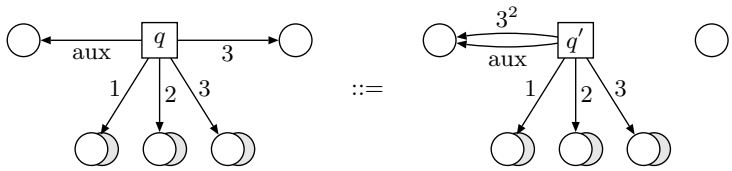
Suppose $\mathcal{A} = \{a_1, \ldots, a_{m-1}\}$ (using an arbitrary but fixed order on the symbols in $\mathcal{A}$). The initial configuration for an input string $w = a_{i_1} \cdots a_{i_n}$ is given by $\mathit{initial}_M(w) = (q_0, c\,0 \cdots 0)$. Here, $c$ is obtained by interpreting $i_1 \cdots i_n$ as a number written in base-$m$ notation. The OCM $M$ accepts $w$ if a configuration $(q, c_1 \cdots c_k)$ with $q \in F$ (called a final configuration) can be reached from $\mathit{initial}_M(w)$. As usual, the recognized language is the set of all strings accepted by $M$. It is well known that counter machines recognize all recursively enumerable languages (see, e.g., [11]). As the reader may easily check, this holds also for the variant defined above.

Let us now see how star rules can simulate an OCM. For this, consider an OCM $M$ as above. We use the states in $Q$ as nonterminals. There is only one further node label in $\dot{\Sigma} \setminus \ddot{\Sigma}$. The corresponding nodes and their multiple counterparts are considered to be unlabelled in the following.

A configuration $C = (q, c_1 \cdots c_k)$ is represented by the non-straight star $gr(C)$ shown on the right. It consists of a nonterminal center node $v$ labelled $q$, a terminal border node $v'$, and $1 + \sum_{i=1}^{k} c_i$ parallel edges from $v$ to $v'$. One of these edges is labelled with aux, whereas $c_i$ edges are labelled with $i$, for $1 \leqslant i \leqslant k$. Here, an edge label carrying an exponent abbreviates the respective number of parallel edges. The edge labelled aux ensures that $gr(C)$ is a star even if $c_1 = \cdots = c_k = 0$.




It is now rather easy to define a set $P_M$ of star rules which simulate the instructions of $M$ by removing or adding the appropriate number of arms in each step. For example, if $k = 3$ and the instruction in question is $(q, 3, \text{NONZERO}) \mapsto (q', +1)$, the resulting star rule looks like this:




To see that this rule has the desired effect, note that its application to a graph of the form $gr(C)$ requires taking a quotient which identifies all border nodes. Intuitively, this means that the arms in the rule are parallel edges in disguise. Hence, the rule applies to $gr(C)$ if counter 3 has a nonzero value and will in this case increase the number of edges labelled 3 by one.

By adding terminating rules (which remove the nonterminal node if the nonterminal is a final state of $M$), we get the following lemma.

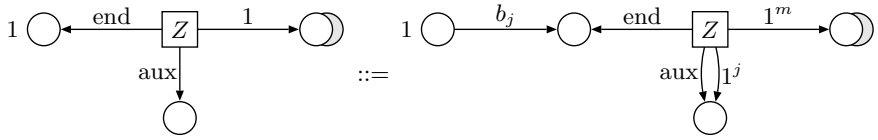
**Lemma 3.** *For every configuration $C$ of $M$, there is a derivation $gr(C) \Rightarrow^{+}_{\Delta P_M} G$ for some terminal graph $G$ if and only if there exists a computation of $M$ that turns $C$ into a final configuration. Furthermore, in this case, $G$ is the graph consisting of a single node and no edges.*

Using Lemma 3, we can now prove the promised result. For this, we identify a string $b_1 \cdots b_n \in \mathcal{A}^*$ with the graph consisting of unlabelled nodes $v_0, \ldots, v_n$ and edges $e_1, \ldots, e_n$, where $e_i$ points from $v_{i-1}$ to $v_i$ and is labelled with $b_i$.

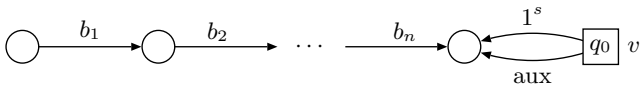
**Theorem 2.** *Every recursively enumerable string language can be generated by an adaptive star grammar.*

*Sketch of Proof.* Consider any recursively enumerable string language $L$, and let $M$ be an OCM recognizing $L$. Without loss of generality, we may assume that $L$ does not contain the empty string. An adaptive star grammar generating $L$ may work as follows. In a preprocessing phase, it generates an arbitrary string $w \in \mathcal{A}^+$ in a nondeterministic fashion. At the same time, the subgraph $gr(\mathit{initial}_M(w))$ is built. In the second phase, the star rules in $P_M$ are used to simulate $M$.

To see that the first phase can really be implemented, note that we can turn $n$ edges labeled with 1 into $n^m + j$ such edges using the rule




Using such rules, we can generate exactly the graphs $G$ of the form




with $b_1, \ldots, b_n \in \mathcal{A}$ and $s$ the representation of $b_1 \cdots b_n$ in base-$m$ notation for some $n \geqslant 1$. Thus, the star occurrence $G(v)$ equals $gr(\mathit{initial}_M(b_1 \cdots b_n))$. Together with Lemma 3, this proves the theorem. □

If adaptive star grammars shall be practically used, it is necessary to come up with restrictions that guarantee the decidability of the membership problem. Therefore, we now study a reasonably restricted class of star grammars that allows to decide this question. We consider *simple adaptive star grammars* first. In the following, let us call an edge $e$ in a graph $G$ terminal if it is not incident with a nonterminal node (i.e., if $\ell_G \circ s_G(e) \notin N$); otherwise, $e$ is nonterminal.

**Definition 12 (Simple adaptive star grammar).** An adaptive star grammar $\Gamma = \langle \Sigma, N, P, Z \rangle$ and its set $P$ of rules are called *simple* if $P$ does not contain any rule whose right-hand side is either just its set of border nodes or contains a non-straight star.

Following Corollary 1 (p. 9), we restrict derivations to the set $nf(P)$ of rules without multiple nodes and to graphs without multiple nodes. Simple adaptive star grammars cannot produce parallel nonterminal edges as right-hand sides do not contain non-straight stars. Hence, in the following, we can ignore rules in $nf(P)$ that are obtained by taking quotients. Let $\tilde{nf}(P)$ be the corresponding set of rules.

**Lemma 4.** *There is an algorithm that decides whether $G \Rightarrow^*_{\tilde{nf}(P)} G'$ for every finite set $P$ of simple star rules and all graphs $G$ and $G'$ without multiple nodes.*

*Proof.* We measure the size of a graph $G$ by $\tau(G) = |\dot{G}| + |\{e \in \bar{G} \mid e \text{ terminal}\}|$. Each derivation $H \Rightarrow_{\tilde{nf}(P)} H'$ removes a nonterminal node, but adds at least one other node or a terminal edge, i.e., $\tau(H) \leqslant \tau(H')$. We prove the lemma by showing that the set of all graphs $G'$ with $G \Rightarrow^*_{\tilde{nf}(P)} G''$ and $\tau(G'') \leqslant \tau(G')$ is finite. The number of graphs $\tilde{G}$ that can be derived from another graph in a single step such that $\tau(\tilde{G}) \leqslant \tau(G)$ is finite. We, therefore, have to show that there is no infinite derivation sequence $G = G_0 \Rightarrow_{p_0} G_1 \Rightarrow_{p_1} \ldots$ such that $G_i \ncong G_j$ for all $i \neq j$ and $p_i \in \tilde{nf}(P)$, $\tau(G_i) \leqslant \tau(G')$ for each $i$. If we assume that there is such an infinite derivation sequence, there must be an index $s$ such that, for each $i \geqslant s$, $\tau(G_i) = \tau(G_s)$ and $p_i \in \tilde{nf}(P)$ is obtained from a rule whose right-hand side is a straight star as $\Gamma$ is simple. Hence, each graph $G_i$ for $i \geqslant s$ has the same number of nodes resp. edges and, as a consequence, there must exist two indices $i, j \geqslant s$, $i \neq j$ such that $G_i \cong G_j$. □

Next, we consider *straight adaptive star grammars*. An adaptive star grammar is called straight if its rules are straight, meaning that their right-hand sides do not contain non-straight stars. Hence, each simple adaptive star grammar is a straight one, but the converse does not hold. The method body grammar in Figure 1 is an example of a straight grammar that is not simple. In order to show that membership is decidable for straight adaptive star grammars, we will use the following lemma:

**Lemma 5.** *There is an algorithm that decides whether* $G \Rightarrow^*_{\tilde{nf}(P)} G \setminus \{x\}$ *for every finite set* $P$ *of straight star rules and all straight stars* $G$ *without multiple nodes, where* $x$ *is the center node of* $G$*.*

*Proof.* The existence of a derivation $G \Rightarrow^*_{\tilde{nf}(P)} G \setminus \{x\}$ requires that no applied rule adds either a new terminal node or a terminal edge. Hence, the number of terminal nodes remains constant in the derivation, and the derivation does not contain graphs containing terminal edges.

Let $P_n$ be the (finite) subset of all rules $(S ::= R) \in \tilde{nf}(P)$ such that $|\dot{S}| \leqslant n$ and $R$ contains neither terminal edges nor terminal nodes that are not border nodes of $S$. Obviously, $G \Rightarrow^*_{\tilde{nf}(P)} G \setminus \{x\}$ iff $G \Rightarrow^*_{P_n} G \setminus \{x\}$ where $n = |\dot{G}|$.

Now, $G \Rightarrow^*_{P_n} G \setminus \{x\}$ is equivalent to the (decidable) question whether an appropriately constructed context-free Chomsky grammar $G'$ generates the empty string. To see this, construct $G'$ by using as nonterminals the set of all isomorphism classes $S$ such that $S$ is a star occurring in one of the rules in $P_n$. Now, let $G'$ contain the rule $[S] \to [S_1] \cdots [S_k]$ if $P_n$ contains a rule $S ::= R$, where $S_1, \ldots, S_k$ are the stars occurring in $R$. It should be clear that $G'$ generates the empty string if and only if there is a derivation $G \Rightarrow^*_{P_n} G \setminus \{x\}$. □

This result allows to prove the following theorem:

**Theorem 3.** *The (uniform) membership problem is decidable for straight adaptive star grammars, i.e., there is an algorithm deciding whether* $G \in \mathcal{L}(\Gamma)$ *for every straight adaptive star grammar* $\Gamma$ *and every graph* $G$*.*

*Proof.* For $\Gamma = \langle \Sigma, N, P, Z \rangle$, we construct a new rule set $P'$ iteratively, as follows. Initially, $P'$ is the (finite) set of all rules in $\tilde{nf}(P)$ whose left-hand sides consist of not more than $|\dot{G}| + 1$ nodes. Now, if $P'$ contains a rule $S ::= R$ such that there occurs a star with center node $x$ in $R$, then the rule $S ::= R \setminus \{x\}$ is added to $P'$ provided that $R(x) \Rightarrow^*_{\tilde{nf}(P)} R(x) \setminus \{x\}$ (which can be decided by Lemma 5). This process is repeated until no new rule can be added to $P'$. Finally, each rule is removed from $P'$ whose right-hand side is just its set of border nodes. Obviously, for every nonempty graph $G$, we have $G \in \mathcal{L}(\Gamma)$ iff $Z \Rightarrow^+_{P'} G$. The result follows by Lemma 4 if $G$ is not empty (as $P'$ is simple), and from Lemma 5 otherwise. □

## 7 Conclusions

Adaptive star grammars are more expressive than context-free graph grammars while retaining a context-free flavour. The extended expressive power is indis-

pensable for generating structures such as object-oriented program models. In this paper the authors have joined their earlier work: The mechanisms presented here are much simpler than those proposed in [12]. Future work will investigate how star grammars can be used in graph transformation rules to define parts of a rule that may be variable, but have a fixed shape. This is useful for modeling refactorings rules, in which complex, variable structures like syntax trees are manipulated as atomic parts. All these concepts will be implemented in the graph transformation language and tool Dia**plan** [13].

## References


1. Tom Mens, Serge Demeyer, and Dirk Janssens. Formalising behaviour-preserving transformation. In Andrea Corradini, Hartmut Ehrig, Hans-Jörg Kreowski, and Grzegorz Rozenberg, editors, *First International Conference on Graph Transformation (ICGT'02)*, number 2505 in LNCS, pages 286–301. Springer, 2002.
2. Annegret Habel. *Hyperedge Replacement: Grammars and Languages*. Number 643 in LNCS. Springer, 1992.
3. Frank Drewes, Annegret Habel, and Hans-Jörg Kreowski. Hyperedge replacement graph grammars. In Rozenberg [14], chapter 2, pages 95–162.
4. Joost Engelfriet and Grzegorz Rozenberg. Node replacement graph grammars. In Rozenberg [14], chapter 1, pages 1–94.
5. Andy Schürr, Andreas Winter, and Albert Zündorf. The PROGRES approach: Language and environment. In Gregor Engels, Hartmut Ehrig, Hans-Jörg Kreowski, and Grzegorz Rozenberg, editors, *Handbook of Graph Grammars and Computing by Graph Transformation. Vol. II: Applications, Languages, and Tools*, chapter 13, pages 487–550. World Scientific, Singapore, 1999.
6. Jörg Niere and Albert Zündorf. Using fujaba for the development of production control systems. *LNCS*, 1779:181–191, 2000.
7. Shane Sendall. Combining generative and graph transformation techniques for model transformation: An effective alliance? In *Proc. OOPSLA'03-Workshop on Generative Techniques in the Context of MDA*, 2003. URL: `www.softmetaware.com/oopsla2003/mda-workshop.html`.
8. Berthold Hoffmann, Dirk Janssens, and Niels Van Eetvelde. Cloning and expanding graph transformation rules for refactoring. *ENTCS*, 152(4), 2006. Proc. Graph and Model Transformation Workshop (GRAMOT'05).
9. Hartmut Ehrig. Introduction to the algebraic theory of graph grammars. In V. Claus, Hartmut Ehrig, and Grzegorz Rozenberg, editors, *Graph Grammars and Their Application to Computer Science and Biology*, number 73 in LNCS, pages 1–69. Springer, 1979.
10. Berthold Hoffmann and Niels Van Eetvelde. A graph grammar for program graphs. Technical report, University of Antwerp, March 2006. UA WIS/INF 2006.
11. John E. Hopcroft and Jeffrey D. Ullman. *Introduction to Automata Theory, Languages and Computation*. Addison-Wesley, Reading, Massachusetts, 1979.
12. Berthold Hoffmann. Graph transformation with variables. In H.-J. Kreowski et al., editors, *Formal Methods in Software and System Modeling*, volume 3393 of *LNCS*, pages 101–115. Springer, 2005.
13. Frank Drewes, Berthold Hoffmann, Raimund Klein, and Mark Minas. Rule-based programming with diaplan. *ENTCS*, 117(1), 2005.
14. Grzegorz Rozenberg, editor. *Handbook of Graph Grammars and Computing by Graph Transformation, Vol. I: Foundations*. World Scientific, Singapore, 1997.