Approximating Parikh Images for Generating Deterministic Graph Parsers

Frank Drewes[1], Berthold Hoffmann[2], and Mark Minas[3(✉)]

[1] Umeå Universitet, Umeå, Sweden
drewes@cs.umu.se
[2] Universität Bremen, Bremen, Germany
hof@informatik.uni-bremen.de
[3] Universität der Bundeswehr München, Neubiberg, Germany
mark.minas@unibw.de

Abstract. The Parikh image of a word abstracts from the order of its letters. Parikh's famous theorem states that the set of Parikh images of a context-free string language forms a semilinear set that can be effectively computed from its grammar. In this paper we study the computation of Parikh images for graph grammars defined by contextual hyperedge replacement (CHR). Our motivation is to generate efficient predictive top-down (PTD) parsers for a subclass of CHR grammars. We illustrate this by describing the subtask that identifies the nodes of the input graph that parsing starts with.

1 Introduction

The Parikh image of a word abstracts from the positions of letters in the word, by just counting how often these letters occur. Parikh's theorem states that the set of Parikh images of a context-free string language forms a semilinear set that can be effectively computed from its grammar [12]. Another way to put this is to say that, if the order of symbols in strings is disregarded, in effect turning every string into a multiset of symbols, then the context-free languages are effectively equal to the regular ones. This theorem is useful for studying properties of languages, e.g., for proving that some language is not context-free.

In this paper we study the computation of Parikh images for contextual hyperedge replacement (CHR) grammars. Our motivation is the automated generation of efficient parsers for these grammars. In [4], we have devised predictive top-down (PTD) parsers for a class of CHR grammars,[1] a technique similar to top-down $LL(1)$ string parsing. The complexity of PTD parsing is quadratic in general and linear in many practical cases, whereas that of general HR parsing (and thus of CHR parsing as well) is known to be NP-complete. Parikh images are the heart of the PTD parser generation, as they are used to make rule selection deterministic: imagine that the parser is in a situation where it has to expand a nonterminal hyperedge labelled A that is attached to a node v of

[1] Due to space restrictions, that paper describes only the HR case.

P. Milazzo et al. (Eds.): STAF 2016, LNCS 9946, pp. 112–128, 2016.
DOI: 10.1007/978-3-319-50230-4_9

the input graph, and there are two rules p, p' with the left-hand side A. Assume further that we have determined the semilinear sets U, V of terminal edge labels that derivations starting with p or p' can attach to v, and that these sets are disjoint. Then the parser can decide whether to apply p or p' by inspecting the part of the input still to be generated, and by checking whether the multiset of labels of edges actually attached to v belongs to U or to V.

Unfortunately, the exact computation of the required Parikh images is computationally far too expensive. Even more importantly, the resulting semilinear expressions become so huge that they cannot be handled in a reasonably efficient manner. Thus, one cannot hope to solve the problem by more efficient algorithms. We therefore propose a procedure which computes an over-approximation of the exact solution that is sufficiently close to the exact solution and sufficiently efficient to be used for PTD parser generation. We illustrate its use by considering the subtask of the parser generator that determines which nodes of the input graph have to be matched when parsing starts.

The remainder of this paper is structured as follows. In Sect. 2, we recall Parikh images, and discuss their exact computation for a given grammar. Since this algorithm is too inefficient for grammars of the size occurring in practical applications, we devise procedures that over-approximate Parikh images, in Sect. 3. That far, we discuss just the simple case of context-free string grammars. In Sect. 4, we introduce CHR grammars, and explain how the start nodes for PTD parsers of CHR grammars can be constructed with the help of the techniques developed in the earlier sections. Finally we mention some related and future work in Sect. 5.

2 Parikh Images and Grammar Graphs

Let Σ be a finite alphabet. We wish to count occurrences of terminal symbols in strings over Σ, but disregard the order of the occurrences of symbols. Thus, in effect, we want to work with finite multisets of elements of Σ rather than with strings. Instead of the usual free monoid Σ^* over Σ, we therefore consider the free *commutative* monoid $\Sigma^{\circledast}$ over Σ in which the monoid operator $\cdot$ is commutative, i.e., $a \cdot b = b \cdot a$. In other words, $\cdot$ is the union of multisets. If no confusion is likely to arise, we may drop the operator $\cdot$ in expressions, but the reader should keep in mind that the order of the symbols $a_1, \ldots, a_n$ in $a_1 \cdots a_n$ is irrelevant in this case, despite the string-like appearance of the expression. Note that, as a special case of this notation, a denotes the singleton multiset $\{a\}$. Finally, we define the partial ordering $\preceq$ on $\Sigma^{\circledast}$ to be multiset inclusion, i.e., $u \preceq v$ if v contains every symbol at least as many times as u does.

In the following, let $\Gamma = \langle \Sigma, \mathcal{N}, P, S \rangle$ be a fixed context-free Chomsky grammar with the sets Σ and $\mathcal{N}$ of terminal and nonterminal symbols, respectively, $\Sigma \cap \mathcal{N} = \varnothing$, $P \subseteq \mathcal{N} \times (\mathcal{N} \cup \Sigma)^{\circledast}$ the set of rules (or productions), and $S \in \mathcal{N}$ the start symbol. Note that we interpret the right-hand sides of rules as elements of $(\mathcal{N} \cup \Sigma)^{\circledast}$ rather than as strings. Accordingly, the language $\mathcal{L}(\Gamma)$ is a subset of $\Sigma^{\circledast}$, namely the Parikh image of the traditional string language generated by Γ.

In order to operate on languages $L \subseteq \Sigma^{\circledast}$, we make use of the so-called *counting semiring* over such languages with the addition $+$ of languages being their union and multiplication $\cdot$ being the extension of multiset union $\cdot$ to languages of multisets, i.e., $U \cdot V = \{u \cdot v \mid u \in U, v \in V\}$. Thus, the additive and multiplicative identities are the empty set and $\{\varepsilon\}$, respectively. Again, we write ε instead of $\{\varepsilon\}$. Note that the counting semiring is isomorphic to the one on sets of Parikh vectors counting occurrences of terminal symbols in words over Σ. We define the Kleene operator $\circledast$ on languages $L \subseteq \Sigma^{\circledast}$ as usual: $L^{\circledast}$ is the least set such that $L^{\circledast} = \varepsilon + L \cdot L^{\circledast}$. Since the semiring is commutative as well as idempotent, we have $(K + L)^{\circledast} = K^{\circledast} \cdot L^{\circledast}$ and $(K \cdot L^{\circledast})^{\circledast} = \varepsilon + K \cdot K^{\circledast} \cdot L^{\circledast}$.

Parikh's Theorem [12] states that the commutative language $\mathcal{L}(\Gamma)$ generated by Γ is semilinear, i.e., there are finitely many finite languages $A_1, \ldots, A_n \subseteq \Sigma^{\circledast}$ and $B_1, \ldots, B_n \subseteq \Sigma^{\circledast}$ such that $\mathcal{L}(\Gamma) = A_1 B_1^{\circledast} + \cdots + A_n B_n^{\circledast}$. The languages $A_1, B_1, \ldots, A_n, B_n$ can be effectively computed, e.g., using a generalization of Newton's method [5] but, as explained in the introduction, complexity issues prevent us from using this fact for PTD parser generation. Instead, we devise a procedure that over-approximates the exact Parikh image.

The idea underlying the procedure is to consider all possible derivation trees of Γ and to count the occurrences of terminal symbols in their leaves. We over-approximate the set of derivation trees, thus computing a semilinear set that contains the Parikh image of Γ but is sufficiently close to the exact solution.

A *graph* over our fixed context-free grammar Γ has a set $\dot{G}$ of nodes. Each node $v \in \dot{G}$ is labelled with $\ell(v) \in P \cup \mathcal{N} \cup \Sigma$, i.e., either a rule, a nonterminal, or a terminal symbol of Γ. Instead of explicitly representing edges, each node $v \in \dot{G}$ is assigned a multiset $\mathrm{children}(v) \in \dot{G}^{\circledast}$ of children nodes. A *tree* over Γ is just a graph over Γ that satisfies the usual requirements for trees. As a shorthand notation for a tree t we write $\alpha(t_1, \ldots t_n)$ if t has a root node v with label $\alpha = \ell(v)$ and $\mathrm{children}(v) = v_1 \cdots v_n$ such that v_i is the root node of the direct subtree t_i, for $i \in [n]$,[2] or just α if v is a leaf.

The set $\mathcal{D}(\alpha)$ of *derivation trees* of Γ with root label $\alpha \in \Sigma \cup \mathcal{N} \cup P$ is inductively defined. If $\alpha \in \Sigma$, the tree consists only of the root as its only node. If $\alpha \in \mathcal{N}$, the tree has a single direct subtree, being a derivation tree whose root is labeled with a rule applicable to α. If α is a rule in P, for each occurrence of a symbol in its right-hand side there is a subtree that is a derivation tree with that symbol as its root label. Formally,

$$\mathcal{D}(\alpha) = \begin{cases} \{\alpha\} & \text{if } \alpha \in \Sigma \\ \{\alpha(t) \mid \exists p = (\alpha, r) \in P \colon t \in \mathcal{D}(p)\} & \text{if } \alpha \in \mathcal{N} \\ \{\alpha(t_1, \ldots, t_n) \mid \forall i \in [n] \colon t_i \in \mathcal{D}(a_i)\} & \text{if } \alpha = (A, a_1 \cdots a_n) \in P. \end{cases}$$

We now define the Parikh image $\Psi(t)$ of a derivation tree t as the multiset of the terminal labels of its leaf nodes and the Parikh image $\psi(\alpha)$ of every $\alpha \in \Sigma \cup \mathcal{N} \cup P$ as the set of Parikh images of all derivation trees with root label α:

[2] $[n]$ denotes the set $\{1, \ldots, n\}$.

$$\Psi(t) = \begin{cases} a & \text{if } t = a \text{ and } a \in \Sigma \\ \Psi(t_1) \cdot \ldots \cdot \Psi(t_n) & \text{if } t = \alpha(t_1, \ldots, t_n) \text{ and } \alpha \in \mathcal{N} \cup P \end{cases}$$
$$\psi(\alpha) = \{\Psi(t) \mid t \in \mathcal{D}(\alpha)\}.$$

By this, $\psi(S)$ is obviously $\mathcal{L}(\Gamma)$.

In order to approximate the set of all derivation trees, we encode the rules of Γ in a graph G_Γ over Γ, called *grammar graph* of Γ: G_Γ has $\Sigma \cup \mathcal{N} \cup P$ as its node set, and each node is labelled with itself, $\ell(v) = v$ for each node v. The multiset of children of a nonterminal node A consists of just the rules with left-hand side A (in any order), while the multiset of children of a rule node is simply the right-hand side of that rule, and the multiset of children of a terminal node is the empty multiset ε:

$$\text{children}(v) = \begin{cases} p_1 \cdots p_n \text{ if } v \in \mathcal{N} \text{ and } p_1 \cdots p_n \text{ is the multiset of all rules} \\ \qquad\qquad\qquad\qquad \text{with left-hand side } v \\ a_1 \cdots a_n \text{ if } v = (A, a_1 \cdots a_n) \in P \\ \varepsilon \qquad\quad \text{if } v \in \Sigma \end{cases}$$

Clearly, for all $\alpha \in \Sigma \cup \mathcal{N} \cup P$ the derivation trees with root α can be read off G_Γ by starting at the node α and recursively selecting one of its children if $\alpha \in \mathcal{N}$ (thus choosing a rule for α) or all of them if $\alpha \in P$ (thus building sub-derivations that correspond to the nonterminals in the right-hand side of α).

Our aim is to compute $\mathcal{L}(\Gamma) = \psi(S)$ by counting terminal leaves of the derivation trees with root S. Recall that the strongly connected components (SCCs) of the grammar graph are just the maximal subgraphs in which every node can be reached from every other node on a directed path. Thus, nodes belonging to the same cycle are in the same SCC. We identify an SCC with the set C of its nodes. The problem of counting terminal leaves of complete derivation trees can thus be broken down into the simpler problem of considering each SCC of the grammar graph separately, solving the problem for the (possibly infinite) set of all partial derivation trees that can be "read off" this individual SCC, and combining these solutions to obtain one for the derivation trees of Γ.

We now define how to read off trees from C. These trees are called the *component trees* of C. For this, let us call a node v a *successor* of an SCC C if v is a child of a node $u \in C$ but $v \notin C$. We denote the set of all successors of C by $\operatorname{succ} C$. Note that successors can be terminals, nonterminals, as well as rules. The set $\text{trees}_C(v)$ of all component trees that can be read off an SCC C starting at $v \in C \cup \operatorname{succ} C$ is defined as follows:

$$\text{trees}_C(v) = \begin{cases} \{v\} & \text{if } v \in \Sigma \cap C \text{ or } v \in \operatorname{succ} C \\ \{v(t) \mid t \in \text{trees}_C(v_i) \text{ for some } i \in [k]\} & \\ \qquad\qquad \text{if } v \in \mathcal{N} \cap C \text{ and } \text{children}(v) = v_1 \cdots v_k & \\ \{v(t_1, \ldots, t_k) \mid t_i \in \text{trees}_C(v_i) \text{ for each } i \in [k]\} & \\ \qquad\qquad \text{if } v \in P \cap C \text{ and } \text{children}(v) = v_1 \cdots v_k & \end{cases}$$

A derivation tree t can be composed from component trees t_1, t_2 if t_1 has a leaf v with the same label as the root r of t_2, i.e., $\ell(v) = \ell(r) = u \in C_2 \cap \operatorname{succ} C_1$ for SCCs C_1, C_2. Then t is obtained from t_1 and t_2 by merging v and r.

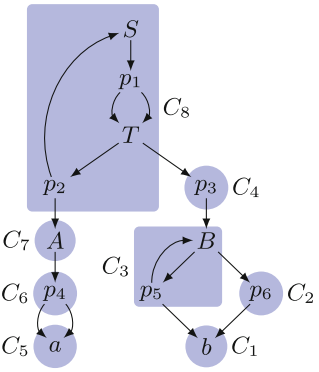

Fig. 1. Grammar graph of the grammar in Example 1 with indicated strongly connected components.

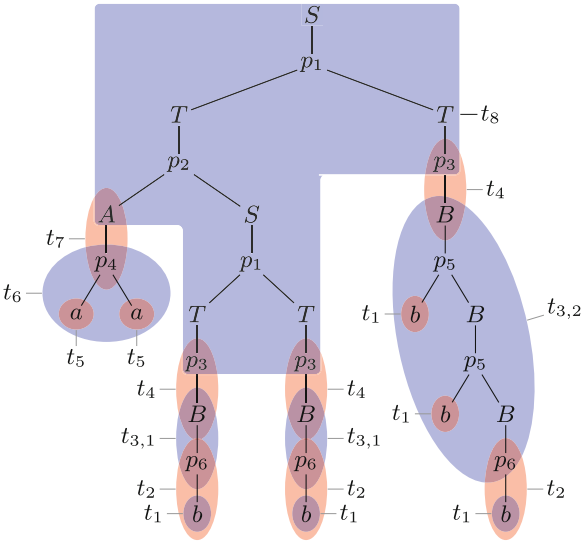

Fig. 2. Derivation tree of a^2b^5.

Example 1. As an example we consider the grammar with $\Sigma = \{a, b\}$, $\mathcal{N} = \{S, A, B, T\}$, and rules $p_1 = \langle S, TT\rangle, p_2 = \langle T, AS\rangle, p_3 = \langle T, B\rangle, p_4 = \langle A, aa\rangle, p_5 = \langle B, bB\rangle, p_6 = \langle B, b\rangle$. Figure 1 shows its grammar graph with indicated strongly connected components and Fig. 2 shows a derivation tree of a^2b^5. The derivation tree is made up of (copies of) the following nine component trees: $t_1 \in \text{trees}_{C_1}(b)$, $t_2 \in \text{trees}_{C_2}(p_6)$, $t_{3,1}, t_{3,2} \in \text{trees}_{C_3}(B)$, $t_4 \in \text{trees}_{C_4}(p_3)$, $t_5 \in \text{trees}_{C_5}(a)$, $t_6 \in \text{trees}_{C_6}(p_4)$, $t_7 \in \text{trees}_{C_7}(A)$, and $t_8 \in \text{trees}_{C_8}(S)$. ♢

The procedure for computing Parikh images (and approximated Parikh images later) of derivation trees runs in three basic steps:

1. The SCCs of the grammar graph are computed.
2. A DAG of SCCs is obtained by contracting each SCC to a single node.
3. This DAG is evaluated in a bottom-up fashion, processing each SCC in turn as described in the following.

The processing of a SCC results in a Parikh image $\psi(v)$ associated with each node v of the SCC. Let C be the next SCC to be processed, assuming that all SCCs containing children of C have already been processed. We determine $\psi(v)$ for each $v \in C$. If v is terminal then $C = \{v\}$ and $\psi(v)$ is a singleton. Hence, the case of $v \in \Sigma \cap C$ does not need to be considered anymore below. For $v \in (\mathcal{N} \cup P) \cap C$, we can determine $\psi(v)$ by collecting the Parikh images of all component trees of C, letting the successor nodes of C act as terminal symbols. Clearly, this

component-specific Parikh image of a component tree $t \in \text{trees}_C(v)$ is

$$\Psi_C(t) = \begin{cases} a & \text{if } t = a \in \text{succ}\, C \\ \Psi_C(t_1) \cdot \ldots \cdot \Psi_C(t_k) & \text{if } t = \alpha(t_1, \ldots, t_k) \text{ and } \alpha \in (\mathcal{N} \cup P) \cap C. \end{cases}$$

The set of component-specific Parikh images of a node $v \in C$ is then defined as

$$\psi_C(v) = \{\Psi_C(t) \mid t \in \text{trees}_C(v)\}.$$

The actual Parikh image $\psi(v)$ for a node $v \in C$ can then be obtained from $\psi_C(v)$ by substituting each occurrence of any node $u \in \text{succ}\, C$ by $\psi(u)$, which has been determined previously.

One can compute $\psi_C(v)$ by interpreting the subgraph induced by $C \cup \text{succ}\, C$ as a system of equations to be solved. Nodes in $\text{succ}\, C$ are constants representing given Parikh images (that have been determined previously in the bottom-up process). Each node $v \in (\mathcal{N} \cup P) \cap C$ with its children $v_1, \ldots, v_k$ stands for a variable defined by an equation. If v is a rule, the equation is $v = v_1 \cdot \ldots \cdot v_k$, otherwise it is $v = v_1 + \cdots + v_k$.

Example 2. The system of equations for SCC C_3 of the grammar graph shown in Fig. 1 is

$$B = p_5 + p_6 \qquad p_5 = bB.$$

It has the solution $B = p_6 b^{\circledast}$, $p_5 = p_6 b b^{\circledast}$. The system of equations for C_8 is

$$S = p_1 \qquad p_1 = TT \qquad T = p_2 + p_3 \qquad p_2 = AS$$

with the solution $S = p_1 = p_3 p_3 (Ap_3)^{\circledast}$, $T = p_3 (Ap_3)^{\circledast}$, $p_2 = Ap_3 p_3 (Ap_3)^{\circledast}$. ♢

Such a system of equations is called linear if there are no products (by $\cdot$) of more than one variable, i.e., if no rule node in C has more than one child in C (e.g., C_3 in the example above). Therefore we call such an SCC *linear*, too. Each linear system of equations corresponds to a finite automaton and can be algebraically solved using Brzozowski's method [1]. If it is non-linear, i.e., if there is a term involving a product of two variables (e.g., C_8 in the example above), one can solve it using a generalization of Newton's method [5].

3 Approximating Parikh Images

Let us call a tree *repetitive* if there are two distinct but equally labeled nodes on a path from the root to a leaf, and *non-repetitive* otherwise. We now show how to compute approximated Parikh images $\psi'(v)$ instead of $\psi(v)$ by computing $\psi'_C(v)$ as an approximation of $\psi_C(v)$ according to (1) when SCC C is processed:

$$\psi'_C(v) = A_C(v) \cdot (\text{rep}\, C)^{\circledast} \tag{1}$$

where

$$A_C(v) = \{\Psi_C(t) \mid t \in \text{trees}_C(v) \text{ and } t \text{ is non-repetitive}\} \tag{2}$$

$$\text{rep}\, C = \begin{cases} \varnothing & \text{if } |C| = 1 \\ (\text{succ}\, C) \backslash P & \text{if } |C| > 1 \text{ and } C \text{ is linear} \\ \text{succ}\, C & \text{if } |C| > 1 \text{ and } C \text{ is not linear} \end{cases} \tag{3}$$

Table 1. Approximated Parikh images for the grammar graph in Fig. 1.

v	SCC	type	rep C	non-repetitive trees	$A_C(v)$	$\psi'_C(v)$	$\psi'(v)$	$\psi''(v)$
b	C_1	elem.	$\varnothing$	$\{b\}$	b	b	b	b
p_6	C_2	elem.	$\varnothing$	$\{p_6(b)\}$	b	b	b	b
B	C_3	linear	$\{b\}$	$\{B(p_6)\}$	p_6	$p_6 b^{\circledast}$	$bb^{\circledast}$	$bb^{\circledast}$
p_3	C_4	elem.	$\varnothing$	$\{p_3(B)\}$	B	B	$bb^{\circledast}$	$bb^{\circledast}$
a	C_5	elem.	$\varnothing$	$\{a\}$	a	a	a	a
p_4	C_6	elem.	$\varnothing$	$\{p_4(a,a)\}$	aa	aa	aa	aa
A	C_7	elem.	$\varnothing$	$\{A(p_4)\}$	p_4	p_4	aa	aa
S	C_8	non-lin.	$\{A, p_3\}$	$\{S(p_1(T(p_3), T(p_3)))\}$	p_3p_3	$p_3p_3A^{\circledast}p_3^{\circledast}$	$bb(aa)^{\circledast}b^{\circledast}$	$bba^{\circledast}b^{\circledast}$

Example 3. Table 1 summarizes the results when applying the procedure of computing the approximated Parikh images for the grammar in Example 1. The table shows in each row a node v of the grammar graph in Fig. 1, its SCC C and the type of C where linear SCCs are distinguished from non-linear and elementary ones, the latter being those with $|C| = 1$. Set $\operatorname{rep} C$ of the SCC and the set of all non-repetitive component trees with root label v follow. The three remaining entries are the sets $A_C(v)$, $\psi'_C(v)$, and $\psi'(v)$, written as algebraic terms. $\psi'(v)$ is obtained from $\psi'_C(v)$ by substituting each occurrence of any node $u \in \operatorname{succ} C$ by $\psi'(u)$, which has been determined previously. (The last column will be explained in Example 4 below.)

Note that the computed approximated Parikh image is $\psi'(S) = bb(aa)^{\circledast}b^{\circledast} = \{a^{2i}b^j \mid i \geqslant 0 \wedge j \geqslant 2\}$ whereas the exact set, as one can see, is $\psi(S) = bb(aab)^{\circledast}b^{\circledast} = \{a^{2i}b^{i+j} \mid i \geqslant 0 \wedge j \geqslant 2\}$, i.e., $\psi'(S) = \psi(S) + aa(aa)^{\circledast}(aab)^{\circledast}$. ♢

We now examine how precise the approximation is. It is immediately clear that $\psi_C(v) = \psi'_C(v)$ if C is elementary, i.e., $|C| = 1$. We now show that the exact component-specific Parikh images are subsets of their approximations (Lemma 1). But the approximation does not contain elements completely unrelated to the exact solution; instead, each approximated element can be extended to one contained in the exact Parikh image (Lemma 2).

Lemma 1. *$\psi_C(v) \subseteq \psi'_C(v)$ for each SCC C and $v \in C$.*

Proof. We presume an arbitrary SCC C, node $v \in C$, and tree $t \in \operatorname{trees}_C(v)$, and show that $\Psi_C(t) \in A_C(v) \cdot (\operatorname{rep} C)^{\circledast}$. The proof is by induction on the size of t. Thus, assume that $\Psi_C(t') \in A_C(v) \cdot (\operatorname{rep} C)^{\circledast}$ for all $t' \in \operatorname{trees}_C(v)$ such that t' is smaller than t. We distinguish three cases.

Case 1 (t is non-repetitive). The proposition follows from $\Psi_C(t) \in A_C(v)$.

Case 2 (t is repetitive and C is linear). As t is repetitive, there are two nodes v_1 and v_2 on a path in t such that $\ell(v_1) = \ell(v_2)$. This decomposes t into three trees t_1, t_2, t_3 as shown in Fig. 3. By the linearity of C all leaves of t_2 except v_2 are in $(\operatorname{succ} C) \setminus P = \operatorname{rep} C$, which means that $\Psi_C(t_2) \in (\operatorname{rep} C)^{\circledast}$. Moreover, the

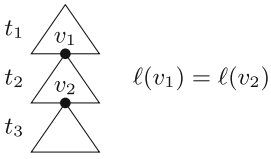

Fig. 3. Construction for Lemma 1

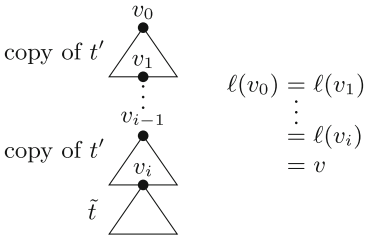

Fig. 4. Construction for Lemma 2

tree t' obtained from t_1 and t_3 by identifying v_1 and v_2 is in $\mathrm{trees}_C(v)$ and is smaller than t. Hence, the induction hypothesis yields

$$\Psi_C(t) = \Psi_C(t') \cdot \Psi_C(t_2) \in A_C(v) \cdot (\operatorname{rep} C)^{\circledast} \cdot (\operatorname{rep} C)^{\circledast} = A_C(v) \cdot (\operatorname{rep} C)^{\circledast}.$$

Case 3 (t is repetitive and C is not linear). Decompose t as in the previous case. Again, $\Psi_C(t_2) \in (\operatorname{rep} C)^{\circledast}$, this time because all leaves of t_2 except v_2 are in $\operatorname{succ} C = \operatorname{rep} C$. Consequently, the same argument as above applies. □

Lemma 2. *For each SCC C, $v \in C$, and $\alpha \in \psi'_C(v)$, there are $\grave{}\alpha, \alpha' \in \psi_C(v)$ such that $\grave{}\alpha \preceq \alpha \preceq \alpha'$.*

Proof. We presume an arbitrary SCC C, node $v \in C$, and $\alpha \in \psi'_C(v)$. By (1) and (2), there is a non-repetitive tree $\tilde{t} \in \mathrm{trees}_C(v)$ and a $\beta \in (\operatorname{rep} C)^{\circledast}$ such that $\alpha = \Psi_C(\tilde{t}) \cdot \beta$ and, therefore, $\Psi_C(\tilde{t}) \preceq \alpha$. In other words, $\grave{}\alpha \preceq \alpha$ for $\grave{}\alpha = \Psi_C(\tilde{t})$. It remains to be shown that there is a tree $t \in \mathrm{trees}_C(v)$ such that $\alpha \preceq \Psi_C(t)$. We distinguish the three cases in Eq. (3) above.

Case 1 ($|C| = 1$). In this case $\alpha \in A_C(v)$, and thus $\alpha = \Psi_C(t)$ for a tree $t \in \mathrm{trees}_C(v)$.

Case 2 ($|C| > 1$ and C is linear). C is strongly connected, i.e., C has a cycle containing each node in C. By following this cycle once, starting at v, one creates a tree t' with both the root and a leaf labelled by v, and such that each node $v \in \operatorname{rep} C$ occurs as the label of at least one node. For each $i \geqslant 0$, construct $t_i \in \mathrm{trees}_C(v)$ from $\tilde{t}$ and i isomorphic copies of t' as shown in Fig. 4. By choosing i to be the maximum multiplicity of elements in β, one obtains $\alpha = \Psi_C(\tilde{t}) \cdot \beta \preceq \Psi_C(t_i)$ because $\beta \in (\operatorname{rep} C)^{\circledast}$ and $\operatorname{rep} C \preceq \Psi_C(t')$.

Case 3 ($|C| > 1$ and C is not linear). Let n be the maximum multiplicity of elements in α, i.e., $\alpha \preceq (\operatorname{succ} C)^n$. By the non-linearity of C, there is a path from v to a rule node r having two distinct nodes $u, u' \in C$ among its children. But there is also a path from u' back to v, which by iteration yields a path starting at v and containing k occurrences of r, for any k. Moreover, again since C is strongly connected, there are $t_1, \ldots, t_k \in \mathrm{trees}_C(u)$ (for a sufficiently large k) such that every node in $\operatorname{succ} C$ occurs in at least n of the t_i. Putting these pieces together, we obtain a tree $t \in \mathrm{trees}_C(v)$ such that $\alpha \preceq (\operatorname{succ} C)^n \preceq \Psi_C(t)$. □

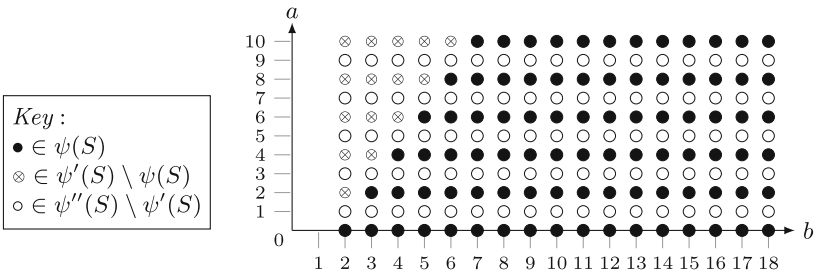

Fig. 5. Parikh images for the grammar in Examples 1–4

The following corollary is an immediate consequence of Lemmas 1 and 2:

Corollary 1. *For each $v \in \Sigma \cup \mathcal{N} \cup P$, the following holds:*

- $\psi(v) \subseteq \psi'(v)$
- *For each $\alpha \in \psi'(v)$, there are $\lhd\alpha, \alpha' \in \psi(v)$ such that $\lhd\alpha \preceq \alpha \preceq \alpha'$.*

In particular, each least element of $\psi'(v)$ is also a least element in $\psi(v)$.

Approximating $\psi(v)$ by $\psi'(v)$ has turned out to be still too inefficient when used for generating parsers for CHR grammars. The most expensive operation is to compute $\psi'(v)$ from $\psi'_C(v)$ by substituting $\psi'(u)$ for each occurrence of any node $u \in \operatorname{succ} C$. This, however, becomes manageable when semilinear sets are approximated by simple semilinear sets. We call a semilinear set M *simple* if there are finitely many finite sets $A_1, \ldots, A_n \subseteq \Sigma^\circledast$ and $B_1, \ldots, B_n \subseteq \Sigma$ such that $M = A_1 B_1^\circledast + \cdots + A_n B_n^\circledast$. (Note that, in contrast to general semilinear sets, $B_1, \ldots, B_n$ are now subsets of Σ rather than of $\Sigma^\circledast$.)

Our approximation of Parikh images using simple semilinear sets now works exactly as before, except for an additional simplification step that lets us compute *simple semilinear sets* $\psi''(v)$ instead of $\psi'(v)$: first, in $\psi'_C(v)$, substitute each occurrence of any node $u \in \operatorname{succ} C$ by the recursively computed $\psi''(u)$. Since $\cdot$ distributes over $+$, we can write the resulting expression in the form $A_1 C_1 + \cdots + A_n C_n$, where the A_i are products not containing $\circledast$ and the C_i are products and sums of expressions of the form $E^\circledast$. Finally, $\psi''(v) = A_1 B_1^\circledast + \cdots A_n B_n^\circledast$, where $B_i = \{a \in \Sigma \mid a \text{occurs in} C_i\}$. One can now verify that Lemmas 1, 2, and Corollary 1 still hold if ψ' is replaced by ψ''.

Example 4. The last column in Table 1 above summarizes the results when applying the procedure of computing the Parikh images for the grammar in Example 1 approximated by simple semilinear sets.

Note that the approximated Parikh image of the generated language is now $\psi''(S) = bba^\circledast b^\circledast = \{a^i b^j \mid i \geqslant 0 \wedge j \geqslant 2\} = \psi'(S) + abb(aa)^\circledast b^\circledast$, where $\psi'(S) = bb(aa)^\circledast b^\circledast$ and $\psi(S) = bb(aab)^\circledast b^\circledast$ (see Example 3). Figure 5 shows elements of the Parikh images in the (a, b) plane. ♢

4 Application to Deterministic Graph Parsing

In order to illustrate how the approximation of Parikh images can be used to generate predictive top-down (PTD) graph parsers, we recall contextual hypergraph replacement (CHR) as far as it is needed to understand the example. (See [3,4] for details of CHR grammars and PTD parsing, resp.)

We consider a ranked labeling alphabet Σ that comes with an *arity* function $arity\colon \Sigma \to \mathbb{N}$. A *hypergraph* $G = \langle \dot{G}, \bar{G}, att_G, \ell_G \rangle$ *over* Σ (*graph*, for short) consists of disjoint finite sets $\dot{G}$ of *nodes* and $\bar{G}$ of *hyperedges* (*edges*, for short) respectively, a function $att_G\colon \bar{G} \to \dot{G}^*$ that *attaches* sequences of nodes to edges, and a *labeling* function $\ell_G\colon \bar{G} \to \Sigma$ so that $|att_G(e)| = arity(\ell_G(e))$ for every edge $e \in \bar{G}$. The set of all graphs over Σ is denoted by $\mathcal{G}_\Sigma$. For a graph G and an edge $e \in \bar{G}$, we denote by $G - e$ the graph obtained by removing e from G.

For graphs G and H, a *morphism* $m\colon G \to H$ is a pair $m = \langle \dot{m}, \bar{m} \rangle$ of functions $\dot{m}\colon \dot{G} \to \dot{H}$ and $\bar{m}\colon \bar{G} \to \bar{H}$ preserving labels and attachments: $\ell_H \circ \bar{m} = \ell_G$, and $att_H \circ \bar{m} = \dot{m}^* \circ att_G$; m is *injective* if both $\dot{m}$ and $\bar{m}$ are injective.

We consider edges labeled with a distinguished subset $X \subseteq \Sigma$ as *nonterminals*. A *contextual rule* (*rule*, for short) $L ::= R$ consists of graphs L and R over Σ such that (1) the *left-hand side* L contains exactly one edge x, which is required to be a nonterminal (i.e., $\bar{L} = \{x\}$ with $\bar{\ell}_L(x) \in X$) and (2) the *right-hand side* R is a supergraph of $L - x$. Nodes in L that are not attached to x are the *contextual nodes* of L (and of r); r is *context-free* if it has no contextual nodes.

Let r be a contextual rule as above, and consider some graph G. If there is an injective morphism $m\colon L \to G$, the *replacement* of $m(x)$ by R (via m) is given as the graph H obtained from the disjoint union of $G - m(x)$ and R by identifying every node $v \in \dot{L}$ with $m(v)$. We then write $G \Rightarrow_r H$.

Let $\mathcal{R}$ be a finite set of contextual rules. We write $G \Rightarrow_{\mathcal{R}} H$ if $G \Rightarrow_r H$ for some rule $r \in \mathcal{R}$, and denote the transitive-reflexive closure of $\Rightarrow_{\mathcal{R}}$ by $\Rightarrow^*_{\mathcal{R}}$.

A *contextual hyperedge-replacement graph grammar* (*CHR grammar*, for short) is a triple $\Gamma = \langle \Sigma, \mathcal{R}, Z \rangle$ consisting of a finite labeling alphabet Σ, a finite set $\mathcal{R}$ of contextual rules, and a start graph $Z \in \mathcal{G}_\Sigma$ consisting of a single nonterminal without any attached nodes. The language of terminal graphs generated by Γ is given by $\mathcal{L}(\Gamma) = \{G \in \mathcal{G}_{\Sigma \setminus X} \mid Z \Rightarrow^*_{\mathcal{R}} G\}$.

Below, following [4], we denote graphs as multisets of literals $a(v_1, \ldots, v_k)$. Such a literal represents an edge that carries a k-ary label $a \in \Sigma$ and connects nodes $v_1, \ldots, v_k$. An isolated node x (such as a context node) is represented by the literal (x).

Example 5 (Flowcharts). A flowchart graph represents the control flow of a program, where nodes (circles) represent program states that are connected by edges representing decisions (diamonds), activities (rectangles), and gotos (thick arrows). An example is the graph in Fig. 6 which, if represented by means of edge literals as introduced above, would be denoted textually as $dec(a, b, c)$ $goto(b, d)$ $act(c, d)$. (Recall that this should be read as a multiset of literals, despite its string-like appearance.)

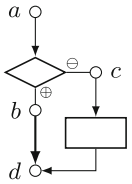

Fig. 6. A flowchart graph.

Now consider the rules

$$S() ::\overset{\mathsf{i}}{=} D(x) \qquad D(x) ::\overset{\mathsf{a}}{=} act(x,y)\ D(y) \qquad D(x) ::\overset{\mathsf{h}}{=} (x)$$
$$D(x) ::\overset{\mathsf{b}}{=} dec(x,y,z)\ D(y)\ D(z) \qquad D(x)\ (y) ::\overset{\mathsf{g}}{=} goto(x,y)$$

The context-free rules i, a, b, h generate flow *trees* of decisions and activities; the contextual rule, g, inserts gotos to a program state generated elsewhere. Note that these rules can generate unstructured "Spaghetti code"; this cannot be achieved by context-free rules alone.

The flowchart in Fig. 6 is generated as follows:

$$\begin{aligned} S() &\Rightarrow_{\mathsf{i}} D(a) \\ &\Rightarrow_{\mathsf{b}} dec(a,b,c)\ D(b)\ D(c) \\ &\Rightarrow_{\mathsf{a}} dec(a,b,c)\ D(b)\ act(c,d)\ D(d) \\ &\Rightarrow_{\mathsf{h}} dec(a,b,c)\ D(b)\ act(c,d) \\ &\Rightarrow_{\mathsf{g}} dec(a,b,c)\ goto(b,d)\ act(c,d) \end{aligned}$$

Note that the derivation using rule h does not produce an isolated node d. Therefore, literal (d) is omitted. ♢

One obvious task a graph parser must be able to perform is to identify the nodes at which the processing starts, i.e., which nodes correspond to those in the right-hand side of the initial rule applied. For PTD parsing some (or all) of these nodes – they are called *start nodes* in the following – must be determined in a syntactically correct graph by their neighborhood, i.e., their incident edges. We now describe how this can be done using approximated Parikh images.

Let us introduce the notion of neighborhoods first. Given a graph H and any node v (not necessarily of H), the *neighborhood* of v in H is obtained by merging all nodes except v into one. When H is represented by literals, this neighborhood graph $[H]_v$ is obtained by replacing each occurrence of v in a literal by a unique new node $\bullet$, and all other nodes by a "don't care" node $\circ$. Isolated nodes are removed, i.e., only edge literals are kept.

It is important to note that the set of literals $a(v_1, \ldots, v_k)$ that appear in $[H]_v$ (for a given CHR grammar) is finite, because a is taken from the finite set of terminal edge labels and $v_1, \ldots, v_k \in \{\bullet, \circ\}$. Thus we can view the set of these literals as a complex but finite alphabet Δ, and every $[H]_v$ as an element of $\Delta^{\circledast}$.

Hence every set of such graphs $[H]_v$ becomes a commutative language over Δ, which allows us to apply the results of Sects. 2 and 3 to these languages.

Example 6. The neighborhood of node d in the graph H shown in Fig. 6 is $[H]_d = dec(\circ,\circ,\circ)\ goto(\circ,\bullet)\ act(\circ,\bullet)$. It represents the fact that d has just one incoming *goto* edge and one incoming *act* edge, and there is just one *dec* edge which is not incident with d. ♢

Given a syntactically correct graph H, we want to determine for each $v \in \dot{H}$, by just considering its neighborhood, which rule $p = (L ::= R)$ has generated v in the derivation $Z \Rightarrow^* H$. If p generates not just a single node, we also want to know which node $u \in \dot{R}\backslash\dot{L}$ actually corresponds to v. This can sometimes be done by computing, for each rule $p = (L ::= R)$ and node $u \in \dot{R}\backslash\dot{L}$,

$$nh(p,u) = \{[H]_v \in \Delta^{\circledast} \mid H \text{ is terminal, } Z \Rightarrow^* G \Rightarrow_p G' \Rightarrow^* H, \\ v \in \dot{H}, \text{ and } v \text{ is the image of } u \text{ created in } G \Rightarrow_p G'\}.$$

The set $nh(p,u)$ contains all possible neighborhoods of copies v of u created by applying p in a derivation of a syntactically correct graph.

The parser can identify node $v \in \dot{H}$ as a start node corresponding to a node u in start rule p if $[H]_v \in nh(p,u)$, but $[H]_v \notin nh(p',u')$ for each $(p',u') \neq (p,u)$. The parser can identify the corresponding start node of every syntactically correct graph if $nh(p,u) \cap nh(p',u') = \varnothing$ for each $(p',u') \neq (p,u)$.

In order to compute neighborhoods, we use the fact that each CHR derivation corresponds to a derivation of neighborhoods. To see this, let $[p]_u$, for any CHR rule $p = (L ::= R)$ and for any node u (not necessarily in $\dot{L} \cup \dot{R}$), be the context-free rule $[L]_u ::= [R]_u$ over Δ. It is clear that $[p]_u$ is context-free because $[L]_u$ is a single literal, and that, for all graphs G, G' with $G \Rightarrow_p G'$ and each node v, there is a node x and a multiset α of literals such that $[G]_v = \alpha[L]_x$ and $[G']_v = \alpha[R]_x$, i.e., a context-free derivation $[G]_v \Rightarrow_{[p]_x} [G']_v$.

Define sets of mapped rules $P^\circ = \{[p]_x \mid p = (L ::= R) \in \mathcal{R} \text{ and } x \notin \dot{R}\}$ and $P^\bullet = \{[p]_x \mid p = (L ::= R) \in \mathcal{R} \text{ and } x \in \dot{L}\}$. Thus, P° consists of all rules not containing $\bullet$ at all and $P^\bullet$ consists of those containing $\bullet$ in both the left-hand side and right-hand side (except for contextual rules where a contextual node x is mapped to $\bullet$, as in this case $[p]_x$ contains $\bullet$ only in its right-hand side).

Now, consider any CHR rule $p = (L_p ::= R_p)$ and any node $u \in \dot{R}_p\backslash\dot{L}_p$. Let G, G' be graphs such that $Z \Rightarrow^* G \Rightarrow_p G'$ and $v \in \dot{G}'$ a node that has been created in the last step, being the image of $u \in \dot{R}\backslash\dot{L}$. Then this means that there is a multiset α of literals, called a *vicinity multiset* of $[L_p]_u$, such that $[Z]_v \Rightarrow^*_{P^\circ} [G]_v = \alpha[L_p]_u$. A neighborhood of v is obtained when G' is derived to a terminal graph H and $[G']_v = \alpha[R_p]_u$ is derived in a corresponding way. Clearly, for every terminal graph H with $G' \Rightarrow^* H$, there are multisets α', α'' such that $[G']_v = \alpha[R_p]_u \Rightarrow^*_{P'} \alpha'\alpha'' = [H]_v$ with $\alpha \Rightarrow^*_{P'} \alpha'$ and $[R_p]_u \Rightarrow^*_{P'} \alpha''$ using the set of rules $P' = P^\circ \cup P^\bullet$. Note that the crucial middle step, which uses the rule $[p]_u$ to create $\bullet$ is not covered by the rules in P'.

Example 7. The sets P° and $P^\bullet$ of context-free rules for the flowcharts rules shown in Example 5 are

$$P^\circ = \left\{ \begin{array}{ll} S() ::\overset{[\mathsf{i}]_\circ}{=} D(\circ) & \\ D(\circ) ::\overset{[\mathsf{a}]_\circ}{=} act(\circ,\circ)\ D(\circ) & D(\circ) ::\overset{[\mathsf{b}]_\circ}{=} dec(\circ,\circ,\circ)\ D(\circ)\ D(\circ) \\ D(\circ) ::\overset{[\mathsf{g}]_\circ}{=} goto(\circ,\circ) & D(\circ) ::\overset{[\mathsf{h}]_\circ}{=} \varepsilon \end{array} \right\}$$

$$P^\bullet = \left\{ \begin{array}{ll} D(\bullet) ::\overset{[\mathsf{a}]_x}{=} act(\bullet,\circ)\ D(\circ) & D(\bullet) ::\overset{[\mathsf{b}]_x}{=} dec(\bullet,\circ,\circ)\ D(\circ)\ D(\circ) \\ D(\bullet) ::\overset{[\mathsf{g}]_x}{=} goto(\bullet,\circ) & D(\circ) ::\overset{[\mathsf{g}]_y}{=} goto(\circ,\bullet) \\ D(\bullet) ::\overset{[\mathsf{h}]_x}{=} \varepsilon & \end{array} \right\}$$

Let us consider rule a and its generated node y in the example derivation shown in Example 5. Rule a is applied to graph $G = dec(a,b,c)\ D(b)\ D(c)$, resulting in graph $G' = dec(a,b,c)\ D(b)\ act(c,d)\ D(d)$ and, after continuing the derivation, in graph $H = dec(a,b,c)\ goto(b,d)\ act(c,d)$, i.e., node y in rule a corresponds to node d in G' and also in H. The neighborhood of d in H, therefore, is derived as follows:

$$\begin{aligned} S() &\Rightarrow_{[\mathsf{i}]_\circ} D(\circ) \\ &\Rightarrow_{[\mathsf{b}]_\circ} \underbrace{dec(\circ,\circ,\circ)\ D(\circ)}_{\alpha}\ \underbrace{D(\circ)}_{[L_\mathsf{a}]_y} \\ &\Rightarrow_{[\mathsf{a}]_y} \underbrace{dec(\circ,\circ,\circ)\ D(\circ)}_{\alpha}\ \underbrace{act(\circ,\bullet)\ D(\bullet)}_{[R_\mathsf{a}]_y} \\ &\Rightarrow_{[\mathsf{h}]_x} dec(\circ,\circ,\circ)\ D(\circ)\ act(\circ,\bullet) \\ &\Rightarrow_{[\mathsf{g}]_y} \underbrace{dec(\circ,\circ,\circ)\ goto(\circ,\bullet)}_{\alpha'}\ \underbrace{act(\circ,\bullet)}_{\alpha''} \end{aligned}$$

Note the correspondence of applied context-free rules and the CHR rules applied in Example 5. Note also that the vicinity multiset $\alpha = dec(\circ,\circ,\circ)\ D(\circ)$ does not contain $\bullet$, but its derived multiset $\alpha' = dec(\circ,\circ,\circ)\ goto(\circ,\bullet)$ does. This is so because rule g uses d as a context node in the CHR derivation. ♢

We now show that the set of all possible vicinity multisets α of $[L_p]_u$ is actually a context-free language. To see this, we consider the context-free derivation sequence $[Z]_v \Rightarrow^*_{P^\circ} [G]_v = \alpha[L_p]_u$. $[Z]_v$ and $[L_p]_u$ are nonterminal literals. Therefore, there is a context-free derivation sequence $A_0 \Rightarrow_{P^\circ} \alpha_1 A_1 \Rightarrow_{P^\circ} \alpha_1\alpha_2 A_2 \Rightarrow_{P^\circ} \cdots \Rightarrow_{P^\circ} \alpha_1 \cdots \alpha_n A_n$ with nonterminal literals $A_0, \ldots, A_n$ and $A_0 = [Z]_v$ as well as $A_n = [L_p]_u$, $(A_i ::= \alpha_{i+1} A_{i+1}) \in P^\circ$ for each i, and $\alpha_1 \cdots \alpha_n \Rightarrow^*_{P^\circ} \alpha$. We introduce a new nonterminal symbol A^v for each nonterminal literal A and define the set P^v of *vicinity rules* as

$$P^\mathrm{v} = \{(B^\mathrm{v} ::= \gamma A^\mathrm{v}) \mid (A ::= \gamma B) \in P^\circ \text{ and } B \text{ is nonterminal}\} \cup \{[Z]^\mathrm{v}_\circ ::= \varepsilon\}.$$

Note once more that γB is a multiset, so B may be any nonterminal in the right-hand side of $A ::= \gamma B$. Intuitively, if a derivation tree t over P contains a

nonterminal node u labelled by B, then P^{v} allows to derive from B the "context" of u in t, yielding the multiset that consists of all literals generated by t, except for the subtree rooted at u.

One can now verify that the set of all vicinity multisets just consists of each multiset α such that $[L_p]_u^{\mathrm{v}} \Rightarrow_{P''}^* \alpha$ with $P'' = P^{\circ} \cup P^{\mathrm{v}}$, and, therefore

$$nh(p,u) = \{\alpha \mid [R_p]_u[L_p]_u^{\mathrm{v}} \Rightarrow_{P^{\mathrm{s}}}^* \alpha \text{ and } \alpha \text{ contains terminal literals only}\}$$

where $P^{\mathrm{s}} = P^{\circ} \cup P^{\bullet} \cup P^{\mathrm{v}}$. The set of all neighborhoods can thus be computed as the Parikh image of a new nonterminal symbol S_p^u

$$nh(p,u) = \psi(S_p^u)$$

using the set $P^{\mathrm{s}} \cup \{S_p^u ::= [R_p]_u[L_p]_u^{\mathrm{v}}\}$ of rules.

Example 8. The set P^{v} of vicinity rules for flowcharts (Example 5) is

$$P^{\mathrm{v}} = \left\{ \begin{array}{ll} S()^{\mathrm{v}} ::= \varepsilon & D(\circ)^{\mathrm{v}} ::=^{\mathsf{a}^{\mathrm{v}}} act(\circ,\circ)\ D(\circ)^{\mathrm{v}} \\ D(\circ)^{\mathrm{v}} ::=^{\mathsf{i}^{\mathrm{v}}} S()^{\mathrm{v}} & D(\circ)^{\mathrm{v}} ::=^{\mathsf{b}^{\mathrm{v}}} dec(\circ,\circ,\circ)\ D(\circ)\ D(\circ)^{\mathrm{v}} \end{array} \right\}$$

The only rules in Example 5 that generate any nodes are rules i, a, and b, generating nodes x (rule i), y (rules a and b), and z (rule b). Therefore, we must determine $nh(\mathsf{i},x)$, $nh(\mathsf{a},y)$, $nh(\mathsf{b},y)$, and $nh(\mathsf{b},z)$, which requires the additional nonterminals S_{i}^x, S_{a}^y, S_{b}^y, and S_{b}^z together with the following rules:

$$\begin{array}{ll} S_{\mathsf{i}}^x ::= D(\bullet)\ S()^{\mathrm{v}} & S_{\mathsf{b}}^y ::= dec(\circ,\bullet,\circ)\ D(\bullet)\ D(\circ)\ D(\circ)^{\mathrm{v}} \\ S_{\mathsf{a}}^y ::= act(\circ,\bullet)\ D(\bullet)\ D(\circ)^{\mathrm{v}} & S_{\mathsf{b}}^z ::= dec(\circ,\circ,\bullet)\ D(\circ)\ D(\bullet)\ D(\circ)^{\mathrm{v}} \end{array}$$

The approximated Parikh images over-approximate the corresponding sets of possible neighborhoods:

$$\begin{aligned} \psi''(S_{\mathsf{i}}^x) &= \varepsilon + goto(\bullet,\circ) + \big(dec(\bullet,\circ,\circ) + act(\bullet,\circ)\big) \cdot U \\ \psi''(S_{\mathsf{a}}^y) &= act(\circ,\bullet) \cdot Q \\ \psi''(S_{\mathsf{b}}^y) &= dec(\circ,\bullet,\circ) \cdot Q \\ \psi''(S_{\mathsf{b}}^z) &= dec(\circ,\circ,\bullet) \cdot Q \end{aligned}$$

where

$$\begin{aligned} Q &= \big(\varepsilon + goto(\bullet,\circ) + dec(\bullet,\circ,\circ) + act(\bullet,\circ)\big) \cdot U \\ U &= goto(\circ,\bullet)^{\circledast}\, dec(\circ,\circ,\circ)^{\circledast}\, goto(\circ,\circ)^{\circledast}\, act(\circ,\circ)^{\circledast} \end{aligned}$$

A careful look at these sets reveals that $\psi''(S_{\mathsf{i}}^x) \cap \big(\psi''(S_{\mathsf{a}}^y) \cup \psi''(S_{\mathsf{b}}^y) \cup \psi''(S_{\mathsf{b}}^z)\big) = \varnothing$, i.e., the start node for parsing H is the unique node $v \in \dot{H}$ whose neighborhood is contained in $\psi''(S_{\mathsf{i}}^x)$. It is the node which has no other incoming edges than *goto* edges. ♢

In general, an analysis such as the one above can easily be made automatically once the simple semilinear sets have been computed, because expressions using union and intersection of such sets can easily be checked for emptiness. If the intersection is nonempty, PTD parsing is not possible, because the parser cannot determine unique start nodes for every input graph.

The neighborhood of a node v contains a literal for each edge in the graph, even for those edges that are not incident with v and, therefore, do not contain $\bullet$ in their literals (e.g., $act(\circ, \circ)$ in Example 8). At the expense of loosing some information, one can omit such literals from the neighborhood and use this modified definition of neighborhoods instead. For the flowchart example, therefore, one can determine the start node of a graph as the node with the following approximated set of (modified) neighborhoods, obtained from $\psi''(S_{\mathsf{i}}^x)$:

$$\varepsilon + dec(\bullet, \circ, \circ)goto(\circ, \bullet)^{\circledast} + act(\bullet, \circ)goto(\circ, \bullet)^{\circledast} + goto(\bullet, \circ).$$

This simple semilinear set determines the start node as the node without any incident edges (first subterm), as the node with a leaving *act* or *dec* edge, any number of incoming *goto* edges, but no other edge (second and third subterm), or as the node that has a leaving *goto* edge, but does not have any other incident edge (fourth subterm).[3] The parser can actually determine all start nodes in linear time in the number of edges and nodes of the graph when using modified neighborhoods and when storing graphs with adjacency lists. This is so because the parser must check for each node whether it is one of the start nodes. To this end it must visit each of the incident edges and compute the neighborhood by counting the occurrences of literals. Using the resulting representation of the multiset as a tuple of natural numbers, membership of the neighborhood in a simple semilinear set can be checked in constant time. The proposition follows from the fact that each edge is visited as often as indicated by its arity.

5 Conclusions

In this paper we have devised a procedure for approximating Parikh images, and we have shown how this can be used to find the start nodes for PTD parsers of CHR grammars; the procedure is also used for another property of PTD parsers, called neighbor-determined rule choice in [4].

Semilinear sets are studied and applied in various fields such as complexity and computational theory [9,11], formal verification [13], and program analysis [5]. The membership problem for a fixed semilinear set is of linear time complexity [6], but the constants involved would become huge even for small grammars. In fact, the *uniform* membership problem for semilinear

[3] Note that a node with just a leaving *goto* edge can actually not be a starting node although this is indicated by $\psi''(S_{\mathsf{i}}^x)$. The reason for this over-approximation is that rule $[\mathsf{g}]_x$ can be be applied to $D(\bullet)$ even if there is no additional node that could be used as a context node, which is actually necessary for applying CHR rule g.

sets is NP-complete [8] even if the sets are represented explicitly in the form $A_1 B_1^{\circledast} + \cdots + A_n B_n^{\circledast}$. Furthermore, extracting this explicit form from a context-free Chomsky grammar creates an exponential blow-up in itself. This makes further simplifications mandatory. In practice, it seems that simple approximated Parikh images provide a reasonable compromise between generality and computational efficiency (cf. the experimental evaluation reported in [4]).

Early work on parsing graphs has used little static analysis of grammars [7,10], and the parser generator for positional grammars [2] defers many decisions to parser execution time, and leaves the determination of start nodes to the users of the parsers.

The results of this paper will not only allow us to give a precise definition of the parser generation for CHR grammars; it will also be essential for our future work on generating deterministic bottom-up parsers for CHR grammars, which work analogously to $LR(1)$ string parsers.

Acknowledgements. We thank the anonymous reviewers for the valuable comments.

References

1. Brzozowski, J.A.: Derivatives of regular expressions. J. ACM **11**(4), 481–494 (1964)
2. Costagliola, G., Chang, S.K.: Using linear positional grammars for the LR parsing of 2-D symbolic languages. Grammars **2**(1), 1–34 (1999)
3. Drewes, F., Hoffmann, B.: Contextual hyperedge replacement. Acta Informatica **52**(6), 497–524 (2015)
4. Drewes, F., Hoffmann, B., Minas, M.: Predictive top-down parsing for hyperedge replacement grammars. In: Parisi-Presicce, F., Westfechtel, B. (eds.) ICGT 2015. LNCS, vol. 9151, pp. 19–34. Springer, Heidelberg (2015). doi:10.1007/978-3-319-21145-9_2
5. Esparza, J., Kiefer, S., Luttenberger, M.: Newton's method for *Omega*-continuous semirings. In: Aceto, L., Damgård, I., Goldberg, L.A., Halldórsson, M.M., Ingólfsdóttir, A., Walukiewicz, I. (eds.) ICALP 2008. LNCS, vol. 5126, pp. 14–26. Springer, Heidelberg (2008). doi:10.1007/978-3-540-70583-3_2
6. Fischer, P.C., Meyer, A.R., Rosenberg, A.L.: Counter machines and counter languages. Math. Syst. Theor. **2**, 265–283 (1968)
7. Franck, R.: A class of linearly parsable graph grammars. Acta Informatica **10**(2), 175–201 (1978)
8. Huynh, T.-D.: The complexity of semilinear sets. In: Bakker, J., Leeuwen, J. (eds.) ICALP 1980. LNCS, vol. 85, pp. 324–337. Springer, Heidelberg (1980). doi:10.1007/3-540-10003-2_81
9. Ibarra, O.H., Seki, S.: Characterizations of bounded semilinear languages by one-way and two-way deterministic machines. Int. J. Found. Comput. Sci. **23**(6), 1291–1305 (2012)
10. Kaul, M.: Practical applications of precedence graph grammars. In: Ehrig, H., Nagl, M., Rozenberg, G., Rosenfeld, A. (eds.) Graph Grammars 1986. LNCS, vol. 291, pp. 326–342. Springer, Heidelberg (1987). doi:10.1007/3-540-18771-5_62

11. Lavado, G.J., Pighizzini, G., Seki, S.: Converting nondeterministic automata and context-free grammars into Parikh equivalent deterministic automata. In: Yen, H.-C., Ibarra, O.H. (eds.) DLT 2012. LNCS, vol. 7410, pp. 284–295. Springer, Heidelberg (2012). doi:10.1007/978-3-642-31653-1_26
12. Parikh, R.J.: On context-free languages. J. ACM **13**(4), 570–581 (1966)
13. To, A.W.: Model checking infinite-state systems: generic and specific approaches. Ph.D. thesis, School of Informatics, University of Edinburgh, August 2010