Graph Transformation in Constant Time

Mike Dodds and Detlef Plump

Department of Computer Science
The University of York

Abstract. We present conditions under which graph transformation rules can be applied in time independent of the size of the input graph: graphs must contain a unique *root label*, nodes in the left-hand sides of rules must be reachable from the root, and nodes must have a bounded outdegree. We establish a constant upper bound for the time needed to construct all graphs resulting from an application of a fixed rule to an input graph. We also give an improved upper bound under the stronger condition that all edges outgoing from a node must have distinct labels. Then this result is applied to identify a class of graph reduction systems that define graph languages with a linear membership test. In a case study we prove that the (non-context-free) language of balanced binary trees with backpointers belongs to this class.

1 Introduction

A major obstacle to using graph transformation as a practical computation mechanism is its complexity. Finding a match for a rule r in a graph G requires time $O(\text{size}(G)^{\text{size}(L)})$, where L is the left-hand graph of r. This is too expensive for many applications, even if r is fixed (meaning that $\text{size}(L)$ is a constant). For example, Fradet and Le Metayer [8] and later Dodds and Plump [4] have proposed to extend the C programming language with graph transformation rules to allow the safe manipulation of pointers. To make such a language acceptable for programmers, rules must be applicable in constant time.

In [4], constant-time rule application is achieved by using a form of *rooted* graph transformation which is characterized by the presence of unique root nodes in rules and host graphs. These roots serve as entry points for the matching algorithm and ensure, under further assumptions on left-hand sides and host graphs, that all matches of a rule can be found in time independent of the size of the host graph. The purpose of this paper is twofold: to develop a general approach to rooted graph transformation in the setting of the double-pushout approach, and to demonstrate the expressive power of rooted graph transformation in a case study on graph recognition.

Our contributions are as follows. In Section 3, we present two axiomatic conditions each of which guarantees that rules can be applied in time independent of the size of host graphs. The first condition requires that graphs have a unique root, nodes in left-hand sides of rules are reachable from the root, and nodes in host graphs have a bounded outdegree. Under this condition, we establish a

constant upper bound for the time needed to construct all graphs resulting from an application of a fixed rule to a host graph. The second condition requires, in addition, that all edges outgoing from a node have distinct labels. We prove that this leads to a greatly reduced upper bound. Then, in Section 4, we introduce rooted *graph reduction specifications* for defining graph languages. We identify a class of graph reduction specifications that come with a linear membership test. In a case study we prove that the non-context-free language of balanced binary trees with backpointers belongs to this class. This is remarkable as the best known membership algorithm for context-free graph grammars (in the form of edge replacement grammars) needs cubic time when applied to languages with bounded node degree.

Our approach to rooted graph transformation is similar to Dörr's approach [5] in that he also requires unique root nodes to ensure constant-time application of rules. Instead of limiting outdegree, he aims at avoiding so-called strong V-structures in host graphs. This makes the approaches incomparable in terms of the strength of their assumptions. (We mention a few separating properties in Section 5.) Another major difference is that in [5], all rules are assumed to belong to a graph grammar which produces all host graphs (which allows to analyse the grammar for the impossibility of generating V-structures). We don't require any generation mechanism for host graphs. A final difference is that [5] is based on the algorithmic approach to graph transformation while we work in the setting of the double-pushout approach.

2 Graphs, Rules and Derivations

We review basic notions of the double-pushout approach to graph transformation, using a version that allows unlabelled nodes [12]. Rules with unlabelled nodes allow to relabel nodes and, in addition, represent sets of totally labelled rules because unlabelled nodes in the left-hand side act as placeholders for arbitrarily labelled nodes.

A *label alphabet* is a pair $\mathcal{C} = \langle \mathcal{C}_V, \mathcal{C}_E \rangle$ of finite sets $\mathcal{C}_V$ and $\mathcal{C}_E$. The elements of $\mathcal{C}_V$ and $\mathcal{C}_E$ serve as node labels and edge labels, respectively. For this section and the next, we assume a fixed alphabet $\mathcal{C}$.

A *graph* $G = \langle V_G, E_G, s_G, t_G, l_G, m_G \rangle$ consists of a finite set V_G of *nodes* (or *vertices*), a finite set E_G of *edges*, source and target functions $s_G, t_G \colon E_G \to V_G$, a partial node labelling function $l_G \colon V_G \to \mathcal{C}_V$[1] and an edge labelling function $m_G \colon E_G \to \mathcal{C}_E$. The *size* of G, denoted by $|G|$, is the number of its nodes and edges. The *degree* of a node v, denoted by $\mathrm{deg}_G(v)$, is the number of edges incident with v. The *outdegree* of a node v, denoted by $\mathrm{outdeg}_G(v)$, is the number of edges with source v. We write $\mathrm{outlab}_G(v)$ for $m_G(s_G^{-1}(v))$, the set of labels of all edges outgoing from v. A node v' is *reachable* from a node v if $v = v'$ or if there are edges $e_1, \ldots, e_n$ such that $s_G(e_1) = v$, $t_G(e_n) = v'$ and for $i = 1, \ldots, n-1$, $t_G(e_i) = s_G(e_{i+1})$.

[1] The domain of l_G is denoted by $\mathrm{Dom}(l_G)$. We write $l_G(v) = \perp$ to express that node v is in $V_G - \mathrm{Dom}(l_G)$.

A *graph morphism* $g\colon G \to H$ between two graphs G and H consists of two functions $g_V\colon V_G \to V_H$ and $g_E\colon E_G \to E_H$ that preserve sources, targets and labels: $s_H \circ g_E = g_V \circ s_G$, $t_H \circ g_E = g_V \circ t_G$, and $l_H(g_V(v)) = l_G(v)$ for all v in $\mathrm{Dom}(l_G)$. A morphism g is *injective* (*surjective*) if g_V and g_E are injective (surjective); it *preserves undefinedness* if $l_H(g(v)) = \bot$ for all v in $V_G - \mathrm{Dom}(l_G)$. Morphism g is an *isomorphism* if it is injective, surjective and preserves undefinedness. In this case G and H are *isomorphic*, which is denoted by $G \cong H$. Furthermore, g is an *inclusion* if $g(x) = x$ for all nodes and edges x in G. (Note that inclusions need not preserve undefinedness.)

A *rule* $r = \langle L \leftarrow K \rightarrow R\rangle$ consists of two inclusions $K \to L$ and $K \to R$ such that (1) for all $v \in V_L$, $l_L(v) = \bot$ implies $v \in V_K$ and $l_R(v) = \bot$, and (2) for all $v \in V_R$, $l_R(v) = \bot$ implies $v \in V_K$ and $l_L(v) = \bot$. We call L the *left-hand side*, R the *right-hand side* and K the *interface* of r.

A *direct derivation* from a graph G to a graph H via a rule $r = \langle L \leftarrow K \rightarrow R\rangle$, denoted by $G \Rightarrow_{r,g} H$ or just $G \Rightarrow_r H$, consists of two natural pushouts[2] as in Figure 1, where $g\colon L \to G$ is injective.

$$\begin{array}{ccccc} L & \leftarrow & K & \rightarrow & R \\ g\downarrow & (1) & \downarrow & (2) & \downarrow \\ G & \leftarrow & D & \rightarrow & H \end{array}$$

Fig. 1. A direct derivation

In [12] it is shown that for rule r and injective morphism g given, there exists such a direct derivation if and only if g satisfies the *dangling condition*: no node in $g(L) - g(K)$ must be incident to an edge in $G - g(L)$. If this condition is satisfied, then r and g determine D and H uniquely up to isomorphim and H can be constructed (up to isomorphism) from G as follows: (1) Remove all nodes and edges in $g(L) - g(K)$, obtaining a subgraph D'.[3] (2) Add disjointly to D' all nodes and edges from $R - K$, keeping their labels. For $e \in E_R - E_K$, $s_H(e)$ is $s_R(e)$ if $s_R(e) \in V_R - V_K$, otherwise $g_V(s_R(e))$. Targets are defined analogously. (3) For each node $g_V(v)$ in $g(K)$ with $l_L(v) \neq l_R(v)$, $l_H(g_V(v))$ becomes $l_R(v)$.

To keep the complexity considerations below independent of any type system imposed on graphs, we introduce abstract graph classes and rules preserving such classes. A *graph class* is a set $\mathbb{C}$ of graphs over $\mathcal{C}$. A rule r is $\mathbb{C}$-*preserving* if for every direct derivation $G \Rightarrow H$, $G \in \mathbb{C}$ implies $H \in \mathbb{C}$. We can now state the basic problem we are interested in.

---

[2] A pushout is *natural* if it is a pullback, too [12].

[3] D differs from D' in that nodes are unlabelled if they are the images of unlabelled nodes in K that are labelled in L. We do not need D to transform G into H though.

Graph Transformation Problem (GTP).
Given: A graph class $\mathbb{C}$ and a $\mathbb{C}$-preserving rule $r = \langle L \leftarrow K \rightarrow R \rangle$.
Input: A graph G in $\mathbb{C}$.
Output: The set $\{H \mid G \Rightarrow_r H\}$.

We consider the graphs in $\{H \mid G \Rightarrow_r H\}$ up to isomorphism, which makes this set finite (as there are only finitely many morphisms $L \rightarrow G$). The time complexity of GTP is dominated by the cost of finding the injective graph morphisms $L \rightarrow G$. This is because for each of these morphisms, checking the dangling condition and transforming G into H can be done in time independent of the size of G (assuming a suitable data structure for graphs). This leads us to the core problem to solve.

Graph Matching Problem (GMP).
Given: A graph class $\mathbb{C}$ and a $\mathbb{C}$-preserving rule $r = \langle L \leftarrow K \rightarrow R \rangle$.
Input: A graph G in $\mathbb{C}$.
Output: The set $\{g\colon L \rightarrow G \mid g \text{ is injective}\}$.

To solve the GMP in general requires time $|G|^{|L|}$ – better algorithms are not known. If we consider L as part of the input rather than as given, the GMP essentially becomes the *subgraph isomorphism problem* which is NP-complete [11]. (This is the problem to decide whether there exists an injection from L to G. In the worst case, if there is none, this is as expensive as finding all injections.)

3 Rooted Graph Transformation

We now present two conditions each of which ensures that the problems GTP and GMP can be solved in time independent of the size of the input graph G. Both conditions put restrictions on the rule r and the graph class $\mathbb{C}$.

Condition I.
There are $\varrho \in \mathcal{C}_V$ and an integer $b \geq 0$ such that

(1) L contains a unique ϱ-labelled node from which each node is reachable, and
(2) for every graph G in $\mathbb{C}$,
  (i) there is a unique ϱ-labelled node, and
  (ii) the outdegree of each node is bounded by b.

We call the distinguished node labelled with ϱ the *root*. The next condition differs from Condition I in clause (2)(ii).

Condition II.
There exists $\varrho \in \mathcal{C}_V$ such that

(1) L contains a unique ϱ-labelled node from which each node is reachable, and
(2) for every graph G in $\mathbb{C}$,
  (i) there is a unique ϱ-labelled node, and
  (ii) distinct edges outgoing from the same node have distinct labels.

Remark 1. Condition II implies Condition I, which can be seen by choosing bound b as the size of $\mathcal{C}_E$. The converse does not hold in general.

Remark 2. The conditions do not guarantee that r preserves the constraints on $\mathbb{C}$. To preserve property (2) of Condition I, it suffices that the right-hand side R of r contains a unique ϱ-labelled node, and that for each node v in K, $\mathrm{outdeg}_R(v) \leq \mathrm{outdeg}_L(v)$. The preservation of property (2) of Condition II will be discussed in Section 4.

The following algorithm solves the Graph Matching Problem: it constructs the set A_I of all total injections between the left-hand side of a rule and a target graph. The algorithm starts with the partial injection A_0 that is defined as matching the root only. Each iteration of the main loop extends the injections in the previous working set with a single edge and its target node, or a single edge, until the injections in the set are total. When an iteration adds some node or edge to the domain of the injections, we speak of it being *matched*. Edges are matched in an order which ensures that when an edge is matched its source must have been matched in some previous iteration. This ensures that edge matching is inexpensive. This algorithm is based on the similar isomorphism construction algorithm described in [5].

In defining the algorithm, we use some extra notions. Given partial functions $f, f' \colon A \to B$, we write $f \mathrm{\ ext\ } f' \mathrm{\ by\ } Z$ if $\mathrm{Dom}(f) \supseteq \mathrm{Dom}(f')$, $\mathrm{Dom}(f) - \mathrm{Dom}(f') = Z$ and for each $x \in \mathrm{Dom}(f')$, $f(x) = f'(x)$. A *partial graph morphism* $f \colon G \xrightarrow{par} H$ is a graph morphism from some subgraph of G to H. For some graph G and label ϱ we describe a list of edges $e_1 \ldots e_K \in E_G$ as an *edge enumeration* if every edge in E_G appears exactly once in $e_1 \ldots e_K$, and for every edge e_i, either $l_G(s_G(e_i)) = \varrho$ or $\exists j < i \colon s_G(e_i) = t_G(e_j)$.

Algorithm 1 (Graph Matching Algorithm). The algorithm works for a rule $r = \langle L \leftarrow K \rightarrow R \rangle$, an input graph $G \in \mathbb{C}$, as stated in the Graph Matching Problem, and it assumes that Condition I is satisfied.

$e_1 \ldots e_{|E_L|} \Leftarrow$ edge enumeration for L and ϱ
Attach tag to the unique ϱ-labelled node in L
$A_0 \Leftarrow \{h \colon L \xrightarrow{par} G \mid h \text{ is injective} \wedge \mathrm{Dom}(h_V) = l_L^{-1}(\varrho) \wedge \mathrm{Dom}(h_E) = \emptyset\}$
for $i = 1$ to $|E_L|$ **do**
  if $t_L(e_i)$ is not tagged **then**
    attach tag to $t_L(e_i)$
    $A_i \Leftarrow \{h \colon L \xrightarrow{par} G \mid h \text{ is injective} \wedge \exists h' \in A_{i-1} \colon$
      $(h_E \mathrm{\ ext\ } h'_E \mathrm{\ by} \{e_i\} \wedge h_V \mathrm{\ ext\ } h'_V \mathrm{\ by} \{t_L(e_i)\})\}$
  else
    $A_i \Leftarrow \{h \colon L \xrightarrow{par} G \mid h \text{ is injective} \wedge \exists h' \in A_{i-1} \colon$
      $(h_E \mathrm{\ ext\ } h'_E \mathrm{\ by} \{e_i\} \wedge h_V = h'_V)\}$
  end if
end for
return $A_I = A_{|E_L|}$

Proposition 1 (Correctness of Algorithm 1). *Algorithm 1 solves the Graph Matching Problem.*

Proof. We have to show that the returned set A_I is the set of all total injective graph morphisms from L to G.

Soundness. We first show that A_I contains only total injections $L \to G$. By construction of the sets A_i, it is clear that these sets contain partial injections from L to G, so the same holds for A_I. It therefore suffices to show that all morphisms in A_I are total. From the definition of an edge enumeration it follows every edge $e \in E_L$ will be picked by some iteration of the loop. At this point there are three cases: (1) e and $t_L(e)$ are added to the domain of one or more morphisms h in A_{i-1}, where h is already defined for $s_L(e)$, and A_i becomes the set of all extended morphisms. (2) e is added to the domain of one or more morphisms h in A_{i-1}, where h is already defined for both $s_L(e)$ and $t_L(e)$, and A_i becomes the set of all extended morphisms. (3) Matching of e fails because either A_{i-1} is empty or there is no counterpart of e in G. Then A_i will be empty and hence A_I will be empty, too. As a consequence, if A_I is not empty upon termination of the loop, all its morphisms must be defined for all edges in L and their incident nodes. Hence, by the structure of L, they are total morphisms.

Completeness. From the definition of an edge enumeration it follows that every edge in E_L will be picked by some iteration of the loop. Without loss of generality, let e_i be the i-th element of the edge enumeration $e_1 \ldots e_{|E_L|}$, and so the edge picked in the i-th iteration. Also, let L_i be the subgraph of L consisting of the edges $e_1, \ldots, e_i$ and their incident nodes. Then a straightforward induction on i proves that

$$\{h\colon L \xrightarrow{par} G \mid \ h \text{ is injective and } \mathrm{Dom}(h) = L_i\} \subseteq A_i.$$

Since $L_{|E_L|} = L$ by the structure of L, it follows that A_I contains all total injections from L to G. □

Theorem 1 (Complexity of Algorithm 1). *Algorithm 1 requires time* $\Sigma_{i=0}^{|E_L|} b^i$ *at most. The maximal size for the resulting set* A_I *is* $b^{|E_L|}$.

Proof. From the definition of an edge enumeration, it follows that a run of the algorithm involves $|E_L|$ iterations of the loop. In each iteration, one of the cases (1) to (3) of that proof applies. In finding the maximal running time, case (3) can be ignored as A_{i+1} is just set to $\emptyset$. To consider the costs in (1) and (2), let e be the edge picked in the body of the loop.

Case (1): Both e and $t_L(e)$ are added to the domain of one or more morphisms h in A_{i-1}, where h is already defined for $s_L(e)$, and A_i becomes the set of all extended morphisms. By Condition I, there are at most b edges outgoing from $h_V(s_L(e))$. It follows $|A_i| \leq b|A_{i-1}|$. Hence the maximal time needed to update A_i is $b|A_{i-1}|$.

Case (2): Only e is added to the domain of one or more morphisms h in A_{i-1}, where h is already defined for both $s_L(e)$ and $t_L(e)$, and A_i becomes the set of

all extended morphisms. For the same reason as in Case (1), the time needed to update A_i is $b|A_{i-1}|$.

The initialisation of the algorithm constructs the set A_0 which, by Condition I, contains exactly one morphism and therefore can be constructed in time 1 (assuming a suitable data structure for graphs in $\mathbb{C}$). No time is neede for the construction of the edge enumeration $e_1 \dots e_{|E_L|}$, as it can be pre-processed. By the above case analysis, executing the body of the loop takes time $b|A_{i-1}|$ at most. Thus we obtain the following bound for the overall running time:

$$1 + b|A_0| + b|A_1| + \ldots + b|A_{|E_L|}|.$$

By recursively expanding each term $|A_i|$ to its maximal size, we arrive at the expression

$$1 + b + b^2 + \ldots + b^{|E_L|} = \sum_{i=0}^{|E_L|} b^i.$$

This expansion also shows that maximal size of A_I is $b^{|E_L|}$. □

For the rule $r = \langle L \leftarrow K \rightarrow R \rangle$ of the GMP and the GTP, we define size by $|r| = |L| + |K| + |R|$.

Corollary 1 (GTP under Condition I). *Under Condition I, the Graph Transformation Problem can be solved in time* $\Sigma_{i=0}^{|E_L|} b^i + 4|r|b^{|E_L|}$.

Proof. Recall from Section 2 that constructing a derivation $G \Rightarrow_r H$ consists of four stages: (1) Constructing an injective morphism $L \rightarrow G$ that satisfies the dangling condition. (2) Removing nodes and edges. (3) Inserting nodes and edges. (4) Relabelling nodes. To extend this to the GTP problem, we modify stage (1) to: Constructing the set A of *all* injections $L \rightarrow G$ that satisfy the dangling condition. We then perform stages (2) – (4) for all members of the set A.

The dangling condition can be decided for an injection $h\colon L \rightarrow G$ in time $|V_L|$. This is because the condition holds if and only if for all nodes v in $V_L - V_K$, $\deg_L(v) = \deg_G(h(v))$. We assume a graph representation such that the degree of any node can be retrieved in constant time, so we can compare $\deg_L(v)$ with $\deg_G(h(v))$ for all v in L in time $|V_L|$.

We construct the set A_I of all injections between L and G using Algorithm 1. We then complete stage (1) by filtering this set for those morphisms that satisfy the dangling condition.

Given a morphism h, it is obvious that stage (2) can be executed in time $|L - K|$, and stage (3) can be done in time $|R - K|$. Stage (4) requires time $|V_K|$ at most. In the worst case, stages (2) to (4) must be completed for every injection in the set A_I. This results in the time bound

$$\sum_{i=0}^{|E_L|} b^i + b^{|E_L|}(|V_L| + |L - K| + |R - K| + |V_K|).$$

As $|V_L|$, $|L|$, $|R|$, $|K|$ and $|V_K|$ are all bounded by $|r|$, we can estimate the expression from above by $\Sigma_{i=0}^{|E_L|} b^i + 4|r|b^{|E_L|}$. □

Note that according to the GTP and Condition I, $|r|$, $|E_L|$ and b are constants and hence the above time bound is a constant—albeit a possibly large one. The next theorem and the subsequent corollary show that under Condition II, the constant time bounds for the GMP and the GTP decrease from being exponential in $|E_L|$ down to being linear in $|E_L|$ or $|r|$.

Theorem 2 (Complexity of Algorithm 1 under Condition II). *Under Condition II, Algorithm 1 requires time $2|\mathcal{C}_E||E_L|+1$ at most. The resulting set A_I contains at most one injection.*

Proof. Under Condition II, outgoing edges from a node must be distinctly labelled. Hence a partial morphism in A_n can be extended by an edge outgoing from a matched node in at most one way. Since $|A_0| = 1$, it follows that $|A_n| \leq 1$ for every iteration n. In particular, $A_I \leq 1$.

In the time bound of the algorithm established in the proof of Theorem 1, we can therefore replace each $|A_i|$ with 1. We also replace b with $|\mathcal{C}_E|$ (the number of edge labels in the given alphabet). This gives

$$1 + |\mathcal{C}_E| + |\mathcal{C}_E| + \ldots + |\mathcal{C}_E| = |\mathcal{C}_E||E_L| + 1.$$

□

Corollary 2 (GTP under Condition II). *Under Condition II, the Graph Transformation Problem can be solved in time $|\mathcal{C}_E||E_L| + 4|r| + 1$.*

Proof. By Theorem 2, Algorithm 1 will need time at most $|\mathcal{C}_E||E_L| + 1$ under Condition II. The proof of Corollary 1 shows that applying r for a found morphism can be done in time $|V_L| + |L - K| + |R - K| + |V_K|$. Hence we obtain the overall bound

$$|\mathcal{C}_E||E_L| + 1 + |V_L| + |L - K| + |R - K| + |V_K|.$$

As before, $|V_L|$, $|L|$, $|R|$, $|K|$ and $|V_K|$ are bounded by $|r|$ and hence we can estimate the bound from above as $|\mathcal{C}_E||E_L| + 4|r| + 1$. □

4 Efficient Recognition of Graph Languages

In this section we apply the results of the previous section to show that graph languages specified by rooted graph reduction systems of a certain form come with an efficient membership test. This is in sharp contrast to the situation for graph grammars where even context-free languages can be NP-complete.

We define graph reduction languages by adapting the approach of [2] to the setting of rooted graph transformation.

Definition 1 (Signature and Σ-graph). A *signature* $\Sigma = \langle \mathcal{C}, \varrho, \text{type} \rangle$ consists of a label alphabet $\mathcal{C} = \langle \mathcal{C}_V, \mathcal{C}_E \rangle$, a *root label* $\varrho \in \mathcal{C}_V$, and a mapping type: $\mathcal{C}_V \to \mathcal{C}_E$ that assigns to each node label a set of edge labels. A graph G over $\mathcal{C}$ is a *Σ-graph* if it contains a unique ϱ-labelled node, the *root* of G, and if for each node v, (1) $l_V(v) \neq \perp$ implies $\text{outlab}_G(v) \subseteq \text{type}(l_G(v))$ and (2) distinct edges outgoing from v have distinct labels. The set of all Σ-graphs is denoted by $\mathcal{G}(\Sigma)$.

Next we define a class of rules that preserve Σ-graphs.

Definition 2 (Σ-rule). A rule $r = \langle L \leftarrow K \rightarrow R \rangle$ is a *Σ-rule* if L, K and R are Σ-graphs and for each node v in K,

(1) $l_L(v) = \perp = l_R(v)$ implies $\text{outlab}_L(v) = \text{outlab}_R(v)$ and
(2) $l_R(v) \neq \perp$ implies $(\text{outlab}_R(v) \cap \text{type}(l_L(v))) \cup (\text{type}(l_L(v)) - \text{type}(l_R(v))) \subseteq \text{outlab}_L(v)$.

Conditions (1) and (2) ensure that r can add outgoing edges to a node only if the node is relabelled and the edge labels do not belong to the node's old type. Also, outgoing edges of a relabelled node are deleted if their labels do not belong to the node's new type.

Proposition 2 (Σ-rules preserve Σ-graphs). *Let $G \Rightarrow_r H$ such that G is a Σ-graph and r a Σ-rule. Then H is a Σ-graph.*

We now define a 'rooted' version of the graph reduction specifications of [2].

Definition 3 (Graph reduction specification). A *graph reduction specification* $S = \langle \Sigma, \mathcal{C}_N, \mathcal{R}, Acc \rangle$ consists of a signature $\Sigma = \langle \mathcal{C}, \varrho, \text{type} \rangle$, a set $\mathcal{C}_N \subseteq \mathcal{C}_V$ of *nonterminal* labels, a finite set $\mathcal{R}$ of Σ-rules and an $\mathcal{R}$-irreducible[4] Σ-graph Acc, the *accepting graph*, such that in Acc and in all left-hand sides of rules in $\mathcal{R}$, each node is reachable from the root. The graph language specified by S is $\text{L}(S) = \{G \in \mathcal{G}(\Sigma) \mid G \Rightarrow^*_{\mathcal{R}} Acc \text{ and } l_G(V_G) \cap \mathcal{C}_N = \emptyset\}$.

We often abbreviate 'graph reduction specification' by GRS. The following simple example of a GRS specifies cyclic lists as used in pointer data structures.

Example 1 (Cyclic lists). The GRS CL $= \langle \Sigma_{\text{CL}}, \emptyset, \mathcal{R}_{\text{CL}}, Acc_{\text{CL}} \rangle$ has the signature $\Sigma_{\text{CL}} = \langle \{\varrho, E\}, \{p, n\}, \varrho, \{\varrho \mapsto \{p\}, E \mapsto \{n\}\} \rangle$. The accepting graph Acc_{CL} and the rules $\mathcal{R}_{\text{CL}}$ are shown in Figure 2, where the unique ϱ-labelled node is drawn as a small grey node and the label p of its outgoing edge is omitted.

The language L(CL) consists of cyclic lists built up from E-labelled nodes and n-labelled edges, and a distinguished root pointing to any node in the list. For a proof, we have to show soundness (every graph in L(CL) is a cyclic list) and completeness (every cyclic list is in L(CL)). Soundness follows from the fact that for every inverse[5] r^{-1} of a rule r in $\mathcal{R}_{\text{CL}}$, and every cyclic list G, $G \Rightarrow_{r^{-1}} H$

[4] A graph G is *$\mathcal{R}$-irreducible* if there is no step $G \Rightarrow_{\mathcal{R}} H$.
[5] The *inverse* of a rule is obtained by swapping left- and right hand sides together with the inclusion morphisms.

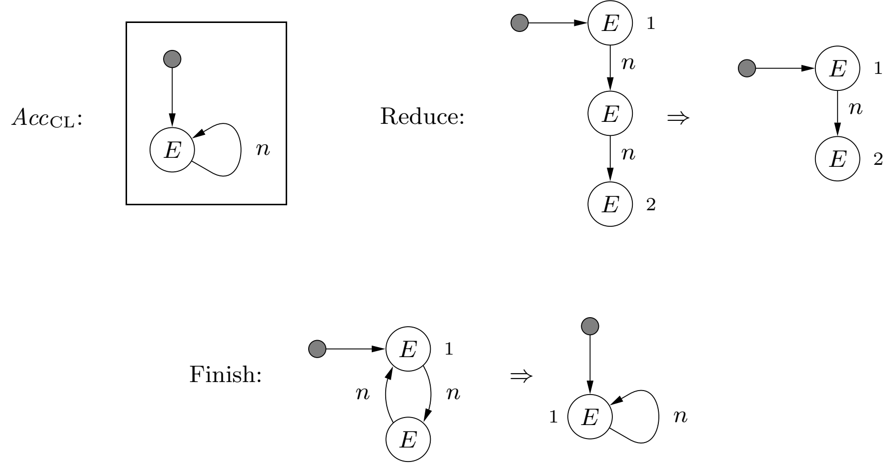

Fig. 2. GRS CL for recognising cyclic lists

implies that H is a cyclic list. For, every reduction $G \Rightarrow^* Acc$ via $\mathcal{R}_{\mathrm{CL}}$ gives rise to a derivation $Acc \Rightarrow^* G$ via $\mathcal{R}_{\mathrm{CL}}^{-1}$ and hence G is a cyclic list. Completeness is shown by induction on the number of E-labelled nodes in cyclic lists. The cyclic list with one E-labelled node is Acc_{CL}, which belongs to L(CL). If G is a cyclic list with at least two E-labelled nodes, then there is a unique injective morphism from the left-hand side of either Reduce (if G has more than two E-labelled nodes) or Finish (if G has exactly two E-labelled nodes) to G. Hence there is a step $G \Rightarrow_{\mathcal{R}_{\mathrm{CL}}} H$, and it is easily seen that H is a cyclic list that is smaller than G. Hence, by induction, there is a derivation $H \Rightarrow^*_{\mathcal{R}_{\mathrm{CL}}} Acc$ and thus $G \in \mathrm{L(CL)}$.

Two properties of CL allow to test graphs in $\mathcal{G}(\Sigma_{\mathrm{CL}})$ efficiently for membership in L(CL). Firstly, reduction sequences terminate after a linear number of steps because both rules reduce the size of a graph. Secondly, reduction is deterministic as Σ_{CL}-graphs contain a unique root and the left-hand sides of the two rules do not overlap. Σ_{CL}-graphs can therefore be tested for membership in L(CL) by a straightforward reduction algorithm: apply the rules of CL as long as possible and check if the resulting graph is isomorphic to Acc_{CL}. □

The properties of CL allowing efficient membership checking can be generalized to obtain a class of GRSs whose languages can be recognised in linear time.

Definition 4 (Linear GRS). A GRS $\langle \Sigma, \mathcal{C}_N, \mathcal{R}, Acc \rangle$ is *linearly terminating* if there is a natural number c such that for every derivation $G \Rightarrow_{\mathcal{R}} G_1 \Rightarrow_{\mathcal{R}} \ldots \Rightarrow_{\mathcal{R}} G_n$ on Σ-graphs, $n \leq c|G|$. It is *closed* if for every step $G \Rightarrow_{\mathcal{R}} H$ on

Σ-graphs, $G \Rightarrow^*_{\mathcal{R}} Acc$ implies $H \Rightarrow^*_{\mathcal{R}} Acc$. A linearly terminating and closed GRS is a *linear* GRS.

The *recognition problem* (or *membership problem*) for GRS languages is defined as follows:

Given: A GRS $S = \langle \Sigma, \mathcal{C}_N, \mathcal{R}, Acc \rangle$.
Instance: A Σ-graph G.
Question: Does G belong to $\mathrm{L}(S)$?

Theorem 3 (Linear recognition). *For linear GRSs, the recognition problem is decidable in linear time.*

Proof. Consider a GRS $S = \langle \Sigma, \mathcal{C}_N, \mathcal{R}, Acc \rangle$. Membership of a Σ-graph G in $\mathrm{L}(S)$ is tested as follows: (1) Check that G contains no node labels from $\mathcal{C}_N$. (2) Apply the rules of $\mathcal{R}$ (nondeterministically) as long as possible. (3) Check that the resulting graph is isomorphic to Acc.

Phase (2) of this procedure terminates in a linear number of reduction steps as S is linearly terminating. By Corollary 2, each step can be performed in constant time. So the time needed for phases (1) and (2) is linear. Phase (3) amounts to checking (i) if there is an injective morphism $Acc \to H$, where H is the graph resulting from the reduction of G, and (ii) if $|Acc| = |H|$. Part (i) requires the same time as the Graph Matching Problem under Condition II (note that Acc is a fixed Σ-graph) and hence, by Theorem 2, can be done in constant time. It follows that phase (3) requires only constant time.

The procedure is correct by the fact that S is closed: if $G \Rightarrow^*_{\mathcal{R}} H$ such that H is $\mathcal{R}$-irreducible and $H \ncong Acc$, then there is no derivation $G \Rightarrow^*_{\mathcal{R}} Acc$. This is shown by a simple induction on the length of $G \Rightarrow^*_{\mathcal{R}} H$. □

To demonstrate the expressive power of linear GRSs, we show that the non-context-free graph language of *balanced binary trees with back-pointers* (BBTBs for short) can be specified by a linear GRS.[6] A BBTB consists of a binary tree built up from nodes labelled B, U and L such that all paths from the tree root to leaves have the same length. Each node has a back-pointer to its parent node, the back-pointer of the tree root is a loop. Nodes labelled with B have two children to which edges with labelled l and r are pointing; nodes labelled with U have one child to which a c-labelled edge points; nodes labelled with L have no children. In addition to this tree, a BBTB has a unique root node whose outgoing edge points to any node in the tree.

The GRS $\mathrm{BB} = \langle \Sigma_{\mathrm{BB}}, \{B', U'\}, \mathcal{R}_{\mathrm{BB}}, Acc_{\mathrm{BB}} \rangle$ is shown in Figure 3, where $\mathrm{type}(B) = \mathrm{type}(B') = \{l, r, b\}$, $\mathrm{type}(U) = \mathrm{type}(U') = \{c, b\}$ and $\mathrm{type}(L) = \{b\}$. Note that B' and U' are nonterminal labels. As in Example 1, we draw the root of a BBTB (not to be confused with the tree root) as a small grey node and omit the label of its unique outgoing edge. We also omit the label b of back-pointers and draw them as dashed edges.

[6] The language of BBTBs is not context-free in the sense of either hyperedge replacement grammars [6] or node replacement grammars [7].

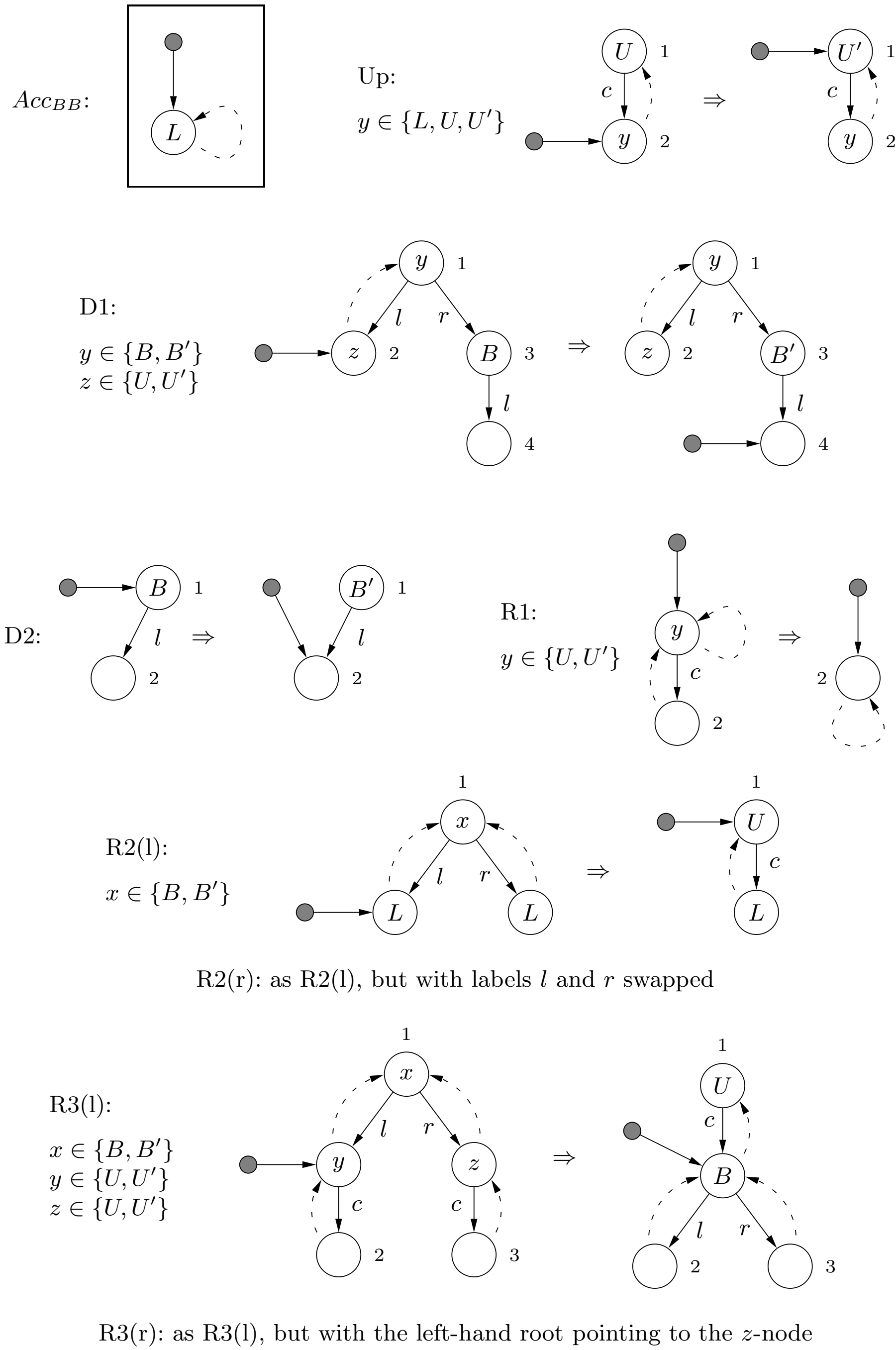

Fig. 3. GRS BB for recognising balanced binary trees with back-pointers

Proposition 3 (Correctness). L(BB) *is the set of all balanced binary trees with back-pointers.*

The proof of Proposition 3 is given in the Appendix.

Proposition 4 (Linearity). *GRS* BB *is linearly terminating: the length of any derivation $G \Rightarrow^*_{\mathcal{R}_{\mathrm{BB}}} H$ on Σ_{BB}-graphs is at most $|G| + |V_G|$.* BB *is also closed and hence is a linear GRS.*

Proof. For every Σ_{BB}-graph G, define $T(G) = |G| + |l_G^{-1}(\mathcal{C}_V - \mathcal{C}_N)|$ where $|l_G^{-1}(\mathcal{C}_V - \mathcal{C}_N)|$ is the number of nodes not labelled with B' or U'. We show that for every step $G \Rightarrow_{\mathcal{R}_{\mathrm{BB}}} H$ on Σ_{BB}-graphs, $T(G) > T(H)$. This implies the bound in the proposition since $|l_G^{-1}(\mathcal{C}_V - \mathcal{C}_N)| \leq |V_G|$.

Rules Up, D1 and D2 preserve the size of graphs but decrease the number of terminally labelled nodes, hence they decrease T's value. Rules R1, R2(l) and R2(r) decrease size without increasing the number of terminal node labels, so they decreasse T's value too. Rules R3(l) and R3(r) decrease size by three and increase the number of terminal node labels by at most two, thus they also decreasse T's value.

BB is closed because for every Σ_{BB}-graph G, the set

$$\{g\colon L \to G \mid L \text{ is a left-hand side in } \mathcal{R}_{\mathrm{BB}} \text{ and } g \text{ is injective}\}$$

contains at most one morphism. This can easily be checked by inspecting the rules of $\mathcal{R}_{\mathrm{BB}}$, keeping in mind that distinct outedges of a node always have distinct labels. □

That the non-context-free language of BBTBs is definable by a linear GRS is quite remarkable. The best known membership test for context-free graph grammars (in the form of edge replacement grammars) needs cubic time when applied to languages with bounded node degree [14]. Context-free graph languages with unbounded node degree can even be NP-complete [13].

5 Related Work

In addition to the remarks made in the Introduction on the relation of our approach to Dörr's work [5], we mention a few separating properties resulting from the different assumptions. While our approach is restricted to graphs of bounded outdegree, [5] allows an unbounded outdegree as long as outgoing edges form permitted V-structures. On the other hand, we allow parallel edges with the same label which are forbidden in Dörr's approach. Moreover, under Condition I, our only constraint on labels is that there is a uniquely labelled root, while all other items may have the same label. This is not possible with the V-structure approach which needs more structure in the labelling of graphs. Another difference is that rule applications in [5] are always deterministic while rules conforming to our Condition I may be nondeterministic.

Some authors have considered rooted graph matching under severe structural restrictions on host graphs, usually resulting from particular application areas. For example, [9, 10] consider graphs representing infinite trees by 'unrolling' from some root. These graphs permit a linear matching algorithm.

There is also a large body of work on recognising graph languages which relates to our work on graph reduction specifications. For example, in [1] it is shown that all graph language expressible in monadic second-order logic (MSOL) can be recognised in linear time if the language has bounded treewidth. But our example of balanced binary trees with back-pointers is not expressible in MSOL and hence outside the scope of this approach.

An efficient linear-time algorithm for recognising such bounded-treewidth languages is given in [3]. This algorithm achieves linear-time termination even though individual reductions may require worse than constant time by recording which potential application areas have already been searched. The algorithm works for *special reduction systems*, which are size-reducing graph reduction systems similar a linear GRSs without roots, but with several other restrictions.

References

1. S. Arnborg, B. Courcelle, A. Proskurowski, and D. Seese. An algebraic theory of graph reduction. *Journal of the ACM*, 40(5):1134–1164, 1993.
2. A. Bakewell, D. Plump, and C. Runciman. Specifying pointer structures by graph reduction. *Mathematical Structures in Computer Science*. To appear. Preliminary version available as Technical Report YCS-2003-367, University of York, 2003.
3. H. L. Bodlaender and B. van Antwerpen-de Fluiter. Reduction algorithms for graphs of small treewidth. *Inf. Comput.*, 167(2):86–119, 2001.
4. M. Dodds and D. Plump. Extending C for checking shape safety. In *Proceedings Graph Transformation for Verification and Concurrency*, Electronic Notes in Theoretical Computer Science. Elsevier, 2005. To appear.
5. H. Dörr. *Efficient Graph Rewriting and its Implementation*, volume 922 of *Lecture Notes in Computer Science*. Springer-Verlag, 1995.
6. F. Drewes, A. Habel, and H.-J. Kreowski. Hyperedge replacement graph grammars. In G. Rozenberg, editor, *Handbook of Graph Grammars and Computing by Graph Transformation. Volume I: Foundations*, chapter 2, pages 95–162. World Scientific, 1997.
7. J. Engelfriet and G. Rozenberg. Node replacement graph grammars. In G. Rozenberg, editor, *Handbook of Graph Grammars and Computing by Graph Transformation. Volume I: Foundations*, chapter 1, pages 1–94. World Scientific, 1997.
8. P. Fradet and D. L. Métayer. Shape types. In *Proceedings of the 1997 ACM Symposium on Principles of Programming Languages*, pages 27–39. ACM Press, 1997.
9. J. J. Fu. Linear matching-time algorithm for the directed graph isomorphism problem. In *Proceedings of the 6th International Symposium on Algorithms*, volume 1004 of *Lecture Notes in Computer Science*, pages 409–417. Springer-Verlag, 1995.
10. J. J. Fu. Pattern matching in directed graphs. In *Proc. Combinatorial Pattern Matching*, volume 937 of *Lecture Notes in Computer Science*, pages 64–77. Springer-Verlag, 1995.

11. M. R. Garey and D. S. Johnson. *Computers and Intractability.* W.H. Freeman and Company, 1979.
12. A. Habel and D. Plump. Relabelling in graph transformation. In *Proc. International Conference on Graph Transformation (ICGT 2002)*, volume 2505 of *Lecture Notes in Computer Science*, pages 135–147. Springer-Verlag, 2002.
13. K.-J. Lange and E. Welzl. String grammars with disconnecting or a basic root of the difficulty in graph grammar parsing. *Discrete Applied Mathematics*, 16:17–30, 1987.
14. W. Vogler. Recognizing edge replacement graph languages in cubic time. In *Proc. Workshop on Graph Grammars and Their Application to Computer Science*, volume 532 of *Lecture Notes in Computer Science*, pages 676–687, 1991.

Appendix

Proof of Proposition 3. Call a graph an NBBTB if it can be obtained from a BBTB by relabelling any number of B-nodes into B'-nodes, and any number of U-nodes into U'-nodes. Call the single edge with a ϱ-labelled node as its source the *root pointer*. Given an NBBTB in which the root pointer does not point to the tree root, call a node a *root-pointer-predecessor* if it is on the unique path of non-back-pointer edges from the tree root to the parent of the root-pointer target. Call a graph an EBBTB if it is an NBBTB satisfying the following conditions: (1) root-pointer-predecessors are not labelled U' and (2) all nodes labelled B' are root-pointer-predecessors. Note that every BBTB it is also an EBBTB.

Soundness. We will show that every Σ_{BB}-graph reducible to Acc_{BB} is an NBBTB, implying that every graph in L(BB) is a BBTB. It suffices to show that the inverses of the rules in $\mathcal{R}_{\mathrm{BB}}$ preserve NBBTBs, as then a simple induction on the length of reductions to Acc gives the desired result.

The inverses of the rules Up, D1 and D2 are clearly NBBTB-preserving as they only relabel nonterminal into terminal nodes and redirect the root pointer to some other node in the tree. The inverse of rule R1 can only be applied at the tree root because this is the only node with a loop attached to it. Hence the rule adds a new tree root, which preserves balance. The inverses of rules R2(l) and R2(r) are also NBBTB-preserving: replacing a U-node pointing to a leaf with a B-node pointing to two leaves preserves balance. Similarly, the inverses of R3(l) and R3(r) preserve balance and the other NBBTB properties.

Completeness. We will show that every EBBTB is reducible to Acc_{BB}, implying that every BBTB is in L(BB). In Proposition 4 it is shown that every $\mathcal{R}_{\mathrm{BB}}$-derivation sequence terminates, so it suffices to show that (1) Acc_{BB} is the only $\mathcal{R}_{\mathrm{BB}}$-irreducible EBBTB and (2) applying any rule in $\mathcal{R}_{\mathrm{BB}}$ to an EBBTB results in an EBBTB.

We first show that every non-Acc_{BB} EBBTB is reducible by $\mathcal{R}_{\mathrm{BB}}$, by enumeration of all possible cases. If the root pointer points to a U or U'-node, we know from Σ_{BB} that it must have an outgoing edge labelled c. If it has no incoming edges, it is the treeroot and we can reduce it using rule R1. If this node has an incoming edge, it must be labelled c, l or r. If c, from the signature and the definition of an EBBTB we know that the source must be a U node, and

so rule Up applies. If the incoming edge is labelled l or r, its source must be a B or B' node and it must have a sibling r or l edge. From the graph balance property, the other edge must point to another node with an outgoing edge – either a U, U', B, or B'. B' is excluded by the definition of an EBBTB. If U or U' rule R3(l) or R3(r) applies. If a B, by Σ_{BB} it must have outgoing l and r edges, and so rule D1 applies.

If the root pointer points to a B-node, by Σ_{BB} it must have an outgoing edge labelled l, so the D2 rule applies. By the definition of an EBBTB, the root pointer cannot point to an B'-node.

If the root pointer points to an L-node, either there are no incoming edges (aside from a backpointer loop) so the graph is Acc_{BB}, or it must have a single incoming edge labelled l, r, or c. If c, by Σ_{BB} and definition of EBBTB the source must be a U and the graph can be reduced by the Up rule. If the incoming edge is labelled l or r, its source must be labelled B and it must have a sibling edge labelled r or l. We know by the balance property that the target of this edge must be another L-node, so either R2(l), or R2(r) applies. This completes the proof that every EBBTB apart from Acc_{BB} can be reduced.

We now show that all rules preserve EBBTBs, once again by case enumeration. Rule Up moves the root pointer from its current position to the node's parent and relabels this parent from U to U'. As the rest of the graph is preserved, this preserves EBBTB condition (1). Rules D1 and D2 move the root pointer and relabel a single B-node to B'. In both cases the B'-node is added to the root-pointer-predecessors, so EBBTB condition (2) is satisfied. No B' nodes are added, so EBBTB condition (1) is satisfied. Rules Up, D1 and D2 only relabel nodes and move the root pointer, so balance is preserved.

Rule R1 deletes a U or U'-node and moves the root pointer to the child of the current position. We know from the EBBTB conditions that this child cannot be labelled B', and so the EBBTB conditions are satisfied. Rule R1 can only delete the treeroot, as this is the only non-L-node that can be safely deleted, and so it preserves balance. Rules R2(l) and R2(r) replace a B or B'-node with a U-node and move the root pointer. Replacing a B-node and two leaves with a U-node and one leaf preserves balance. The EBBTB conditions are satisfied, as the new target of the root pointer is a U-node and the root-pointer-predecessors are otherwise preserved. Rules R3(l) and R3(r) replace two U or U'-nodes with a B node. The rule preserves balance because the distance from the 'top' of the rule to the 'bottom' is preserved. These rules replace a single B or B'-node on the path to the treeroot with a U-node, and the root-pointer-predecessors are otherwise unaltered. No B'-nodes are added, so both EBBTB conditions are satisfied. This completes the proof that rules in $\mathcal{R}_{\mathrm{BB}}$ are EBBTB-preserving. □