SAT-Based Loop Detection in Graph Rewriting

Marcus Ermler

University of Bremen, Department of Computer Science
P.O.Box 33 04 40, 28334 Bremen, Germany
`maermler@informatik.uni-bremen.de`

Abstract

In this paper, we propose an approach for detecting loops in derivations of graph rewriting systems via a translation of the derivation process and loop conditions into propositional formulas. A satisfying assignment represents a derivation with a detected loop and so it witnesses that the corresponding graph rewriting system does not terminate.

Keywords and phrases Non-termination, Loops, Graph Rewriting, SAT Encoding

1 Introduction

The question of termination or non-termination of graph rewriting systems seems to be an interesting issue, but is in general undecidable [5]. Nevertheless, techniques to prove termination of graph rewriting systems were introduced (cf. e.g. [4]). Termination of graph rewriting is defined in the following way (cf. [4]).

▶ **Definition 1** (Termination)**.** A graph rewriting system GRS is *terminating*, if it does not admit an infinite derivation.

The question of looping or non-looping has attracted much attention over the last years in the context of string and term rewriting systems, but is not so much studied in the area of graph rewriting. Also, SAT-based approaches are applied to string and term rewriting (cf. e.g. [6]). In this paper, we propose a new approach for detecting loops in derivations of graph rewriting systems via a translation of derivations and loop conditions into propositional formulas. The idea of translating graph rewriting into SAT was introduced in [3], the proposed technique for loop detection is supplemented to this approach. By using a translation to SAT, we want to benefit from fast solving techniques implemented in modern SAT solvers.

2 Graph Rewriting

We use *edge labeled directed graphs without multiple edges* and with a finite node set. For a finite set Σ of labels, such a graph is a pair $G = (V, E)$ where $V = \{1, \ldots, n\} = [n]$ for some $n \in \mathbb{N}$ is a finite set of *nodes*, numbered from 1 to n, and $E \subseteq V \times \Sigma \times V$ is a set of *labeled edges*. n is called the *size* of G. The components V and E are also denoted by V_G and E_G. We call an edge (v, x, v) a *sling* and an edge $(v, *, v')$ *unlabeled* and omit the label $*$ in drawings. Two edges (v_1, x, v_2) and (v_1', x', v_2') are considered as an *undirected* edge if $v_1 = v_2'$ and $v_2 = v_1'$. A special graph is the *empty graph* $\emptyset = (\emptyset, \emptyset)$. We call G a *subgraph* of H, denoted by $G \subseteq H$, if $V_G \subseteq V_H$ and $E_G \subseteq E_H$.

Furthermore, we use injective graph morphisms for the matching. Let G, H be two graphs as defined above. An *injective graph morphism* from $g\colon G \to H$ is an injective mapping $g_V\colon V_G \to V_H$, that is structure- and label-preserving, i.e. for each edge $(v, x, v') \in E_G$, $(g_V(v), x, g_V(v')) \in E_H$. An injective graph morphism $g\colon G \to H$ yields the image $g(G) = (g_V(V_G), g_E(E_G)) \subseteq H$ with $g_E(E_G)) = \{(g_V(v), x, g_V(v')) \mid (v, x, v') \in E_G\}$ called the

match of G in H. In the following, we will write $g(v)$ and $g(e)$ for nodes $v \in V_G$ and edges $e \in E_G$ because the type of the argument indicates the indices V and E.

▶ **Example 2.** Figure 1 shows the example graph G_0. Its nodes are numbered from 1 to 4 and all its edges except the slings are unlabeled and undirected. The slings are labeled with ∅ to denote that the corresponding nodes have not yet been chosen. The graph on the left-hand of the arrow in Figure 2, called L_{choose} in the following, is a subgraph of G_0. One can choose four injective mappings from L_{choose} to G_0: $g = \{1 \mapsto 1\}$, $g = \{1 \mapsto 2\}$, $g = \{1 \mapsto 3\}$, or $g = \{1 \mapsto 4\}$.

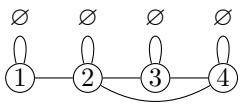

Figure 1 The graph G_0

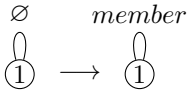

Figure 2 The rule *choose*

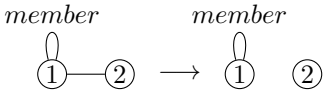

Figure 3 The rule *remEdge*

A *rule* $r = (L \supseteq K \subseteq R)$ consists of three graphs: the *left-hand side* L, the *gluing graph* K, and the *right-hand side* R. In our approach, we only consider rules with $E_K = \emptyset$ and an invariant node set, i.e. $V_L = V_K = V_R$. For that reason, we simplify the rule notation to $r = (L \to R)$, denote the set of nodes of a rule r by V_r and its size by $size(r) = size(L)$.

The application of a rule to a graph works as follows. Let $r = (L \to R)$ be a rule, G a graph, and $g\colon L \to G$ an injective graph morphism. Remove the edges in $g(L)$ from G yielding D and add R disjointly to D. Finally, glue R and D as follows. (1) Merge each $v \in V_R$ with $g(v)$. (2) If there is an edge $(v, x, v') \in E_R$ with $v, v' \in V_R$ and an edge $(g(v), x, g(v')) \in E_D$ then these edges are identified. The application of a rule r to a graph G with respect to an injective graph morphism g yielding a graph H is denoted by $G \underset{r,g}{\Longrightarrow} H$. This is called *rule application* or *direct derivation* and fits into the *double-pushout approach* (cf. [1]). The sequential composition $d = G_0 \underset{r_1,g_1}{\Longrightarrow} G_1 \underset{r_2,g_2}{\Longrightarrow} \cdots \underset{r_n,g_n}{\Longrightarrow} G_n$ of n direct derivations for some $n \in \mathbb{N}$ is called a *derivation*, shortly denoted by $G_0 \overset{*}{\underset{P}{\Longrightarrow}} G_n$ for $n \geq 0$ and $G_0 \overset{+}{\underset{P}{\Longrightarrow}} G_n$ for $n > 0$ if $r_1, \ldots, r_n \in P$.

▶ **Example 3.** In Figure 2, one can find an example for a graph rewriting rule which adds a *member* sling to a node. As described above, there are four mappings from L_{choose} to G_0. In Figure 4 a derivation is shown. The first graph G_1 of the derivation is the result of applying *choose* to G_0 twice by using the mappings $g = \{1 \mapsto 2\}$ and, then, $g = \{1 \mapsto 4\}$. Applying the rule *remEdge* from Figure 3 with the mapping $g = \{1 \mapsto 4, 2 \mapsto 3\}$ results in the second graph. Finally, a three time application of the rule *remEdge* yields the last graph.

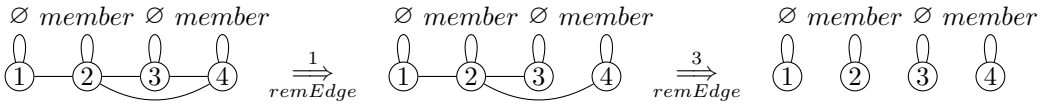

Figure 4 A sample derivation

A *graph grammar* is a system $GG = (G_0, P, \Delta)$ consisting of an *initial graph* G_0, a finite set P of graph rewriting rules, and a set $\Delta \subseteq \Sigma$ of *terminal symbols*. The graph grammar GG specifies all derivations from the initial graph G_0 to graphs labeled over Δ.

▶ **Example 4.** As an example, we consider the *vertex cover problem* that refers to the question, whether for a graph $G = (V, E)$, a subset $X \subseteq V$ exists, such that for all $(v, x, v') \in E$, $v \in X$ or $v' \in X$. The corresponding graph grammar for the graph G_0 from Figure 1 is $VC = (G_0, \{choose, remEdge\}, \{member, \varnothing\})$. The derivation in Figure 4 states the last part of a possible computation with G_0 as input. The two-time application of *choose* yields the first graph of the derivation as detailed above. A vertex cover is found, because the last graph of the derivation has only edges labeled with *member* or $\varnothing$.

3 Detecting Loops in Graph Rewriting via SAT

Every propositional formula with variable set $\{edge(v, a, v', k) \mid (v, a, v') \in [n] \times \Sigma \times [n], k \in [m]\}$ represents a sequence $G_1, \ldots, G_m$ of graphs for each variable assignment f satisfying the formula, i.e. the graph G_k contains the edge (v, a, v') if and only if $f(edge(v, a, v', k)) = TRUE$. Please note, that this translation does not allow node addition or deletion. A single initial graph G in the kth derivation step can be described by the formula

$$\mathsf{graph(G)(k)} = \bigwedge_{(v,a,v') \in E_G} edge(v, a, v', k) \wedge \bigwedge_{(v,a,v') \in ([n] \times \Sigma \times [n]) - E_G} \neg edge(v, a, v', k).$$

▶ **Example 5.** Then, the graph G_0 in Figure 1 is expressed via the following formula

$$\mathsf{graph(G_0)(0)} = \bigwedge_{(v,a,v') \in E_0} edge(v, a, v', 0) \wedge \bigwedge_{(v,a,v') \in ([n] \times \Sigma \times [n]) - E_0} \neg edge(v, a, v', 0)$$

where $E_0 = \{(1, *, 2), (2, *, 1), (2, *, 3), (3, *, 2), (3, *, 4), (4, *, 3), (2, *, 4), (4, *, 2), (1, \varnothing, 1), (2, \varnothing, 2), (3, \varnothing, 3), (4, \varnothing, 4)\}$. Please note, that in case of undirected edges both corresponding directed edges have to occur in the formula.

For a rule $r = (L \to R)$ the set of injective graph morphisms from $[size(r)]$ to the set of nodes $[n]$ is denoted by $\mathcal{M}(r, n)$. Let $k \in \mathbb{N}$ be a derivation step, G_{k-1} be a graph with the node set $[n]$, $r = (L \to R)$ be a rule, and $g \in \mathcal{M}(r, n)$. The application of r to G_{k-1} with respect to g is then expressed by the following formulas

- $\mathsf{morph(r, g, k)} = \bigwedge_{(v,a,v') \in E_L} edge(g(v), a, g(v'), k - 1)$,
- $\mathsf{rem(r, g, k)} = \bigwedge_{(v,a,v') \in E_L - E_R} \neg edge(g(v), a, g(v'), k)$,
- $\mathsf{add(r, g, k)} = \bigwedge_{(v,a,v') \in E_R} edge(g(v), a, g(v'), k)$,
- $\mathsf{keep(r, g, k)} = \bigwedge_{(v,a,v') \notin g(E_L \cup E_R)} \big(edge(v, a, v', k - 1) \leftrightarrow edge(v, a, v', k)\big)$ where $g(E_L \cup E_R) = \{(g(v), a, g(v')) \mid (v, a, v'), \in E_L \cup E_R\}$,
- $\mathsf{apply(r, g, k)} = \mathsf{morph(r, g, k)} \wedge \mathsf{rem(r, g, k)} \wedge \mathsf{add(r, g, k)} \wedge \mathsf{keep(r, g, k)}$.

The formula morph describes that g is a graph morphism from L to G_{k-1}. The removal of the images of every edge of the left-hand side L from G_{k-1} is expressed by rem. The addition of edges of the right-hand side R is described by add. That edges that have been neither deleted nor added must be kept, corresponds to the formula keep. Finally, apply describes the whole application of r to G_{k-1} with respect to g. The following theorem states that a satisfying assignment to apply corresponds to a direct derivation.

▶ **Theorem 6.** *$G_{k-1} \underset{r,g}{\Longrightarrow} G_k$ if and only if there is a satisfying assignment to the formula* $\mathsf{graph(G_{k-1})(k-1)} \wedge \mathsf{apply(r, g, k)} \wedge \mathsf{graph(G_k)(k)}$.

▶ **Example 7.** For the rule *remEdge* in Figure 3, the first graph G_1 in Figure 4, and the graph morphism $g = \{1 \mapsto 4, 2 \mapsto 3\}$ one gets the following formulas: $\mathsf{morph(remEdge, g, 2)} = edge(4, member, 4, 1) \wedge edge(3, *, 4, 1) \wedge edge(4, *, 3, 1)$, $\mathsf{rem(remEdge, g, 2)} = \neg edge(3, *, 4, 2) \wedge \neg edge(4, *, 3, 2)$, and $\mathsf{add(remEdge, g, 2)} = edge(4, member, 4, 2)$.

For each rule $r \in P$ and each mapping $g \in \mathcal{M}(r,n)$, the possible applications of r to G_{k-1} are expressed via $\mathsf{step(k)} = \bigvee_{r\in P, g\in M(r,n)} \mathsf{apply(r,g,k)}$. All derivation starting in G_0 of length m are described by $\mathsf{der(G_0,m)} = \mathsf{graph(G_0)(0)} \wedge \bigwedge_{k=1}^{m} \mathsf{step(k)}$. The following theorem states the connection between der and successful derivations.

▶ **Theorem 8.** *Let G_0 and G_T be two graphs of size n, P be a rule set, and $m \in \mathbb{N}$ be the number of derivation steps. Then there is a successful derivation $G_0 \overset{m}{\underset{P}{\Longrightarrow}} G_T$ if and only if there is a satisfying assignment to $\mathsf{der(G_0,m)} \wedge \mathsf{graph(G_T)(m)}$.*

Let n be the size of G_0 and $p(n)$ be a polynomial bound. Then all derivations starting in G_0 with a length of at most $p(n)$ are expressed via $\mathsf{all\_der(G_0,p(n))} = \bigvee_{m=0}^{p(n)} \mathsf{der(G_0,m)}$.

▶ **Example 9.** A satisfying assignment to the following subformula yields the derivation detailed in Example 3 where c is an abbreviation for choose and rE for remEdge
$\mathsf{graph(G_0)(0)} \wedge \mathsf{apply(c,\{1\mapsto 2\},1)} \wedge \mathsf{apply(c,\{1\mapsto 4\},2)} \wedge \mathsf{apply(rE,\{1\mapsto 4, 2\mapsto 3\},3)} \wedge$
$\mathsf{apply(rE,\{1\mapsto 2, 2\mapsto 1\},4)} \wedge \mathsf{apply(rE,\{1\mapsto 2, 2\mapsto 3\},5)} \wedge \mathsf{apply(rE,\{1\mapsto 4, 2\mapsto 2\},6)}$.

Let G_0 and H_0 be two graphs and $g\colon G_0 \to H_0$ be an injective graph morphism. Then the JOIN-Theorem in ([2], p. 101) states the conditions under which a derivation $G_0 \overset{n}{\Longrightarrow} G_n$ can be extended to a derivation $H_0 \overset{n}{\Longrightarrow} H_n$. Loosely speaking, $H_0 - G_0$ is joined with each G_i. This result can be used to find derivations of the form $G_0 \overset{n}{\Longrightarrow} G_n \Longrightarrow H_0 \supsetneq G_0$, i.e. to detect loops in derivations. According to the JOIN-Theorem, the derivation $G_0 \overset{n}{\Longrightarrow} G_n$ would be extended to a derivation $H_0 \overset{n}{\Longrightarrow} H_n$. Important here is that nodes connecting $g(G_0)$ and $H_0 - g(G_0)$ are not deleted during the derivation. In this paper, we only consider rules that do not delete nodes but this is still a special case. Thus, we use the term *simple looping* instead of looping and define it as follows.

▶ **Definition 10** (Simple Loops)**.** A graph rewriting system with an initial graph G_0 and a rule set P containing no node deletion rules is called *simple looping*, if there are graphs G, H and an injective graph morphism $g\colon G \to H$ such that $G_0 \overset{*}{\underset{P}{\Longrightarrow}} G \overset{+}{\underset{P}{\Longrightarrow}} H$.

▶ **Example 11.** Let us consider again Example 4 and let us assume that *choose* has been defined in a wrong way (see Figure 5). The application of $\overline{choose}$ to G_0 yields the last graph in Figure 6 where G_0 is isomorphic to a subgraph of it, i.e. the derivation contains a simple loop. The rule $\overline{choose}$ can be applied infinitely often to node 2.

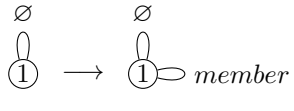

Figure 5 The altered rule $\overline{choose}$

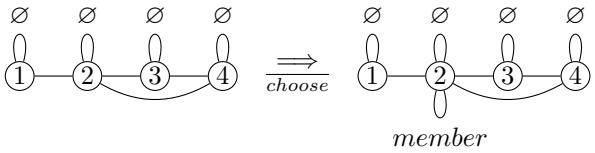

Figure 6 A detected loop

For some graph in a derivation of length m isomorphic to a subgraph of the last graph of this derivation, we define the following propositional formula

$$\mathsf{loop(m)} = \bigvee_{k=1}^{m-1} \bigvee_{g\in\mathcal{M}(n,n)} \bigwedge_{(v,a,v')\in[n]\times\Sigma\times[n]} \Big(edge(v,a,v',k) \to edge(g(v),a,g(v'),m) \Big).$$

Please note, that this formula generates up to n^n possible morphisms ($g \in \mathcal{M}(n,n)$) and has to be restricted in some way. An idea could be to use node types to reduce the possible

matchings. In the derivation from Example 3, we could use the *member*- and ∅-slings as node types such that *member*-nodes could only match *member*-nodes (the same for ∅-nodes).

Detecting a simple loop in all derivations up to a length of $p(n)$ is defined as follows

$$\mathsf{loop\_detection}(\mathsf{G_0}, \mathsf{p(n)}) = \bigvee_{m=0}^{p(n)} \big(\mathsf{der}(\mathsf{G_0}, \mathsf{m}) \wedge \mathsf{loop}(\mathsf{m})\big).$$

Simple looping corresponds to a satisfying assignment for loop_detection.

▶ **Theorem 12.** *Let GRS be a graph rewriting system with an initial graph G_0 of size n and P be a rule set containing no node deletion rules. Then GRS is simple looping if and only if there is a polynomial $p(n)$ such that there is a satisfying assignment to* $\mathsf{loop\_detection}(\mathsf{G_0}, \mathsf{p(n)})$.

▶ **Example 13.** A satisfying assignment to the subformula $\mathsf{der}(\mathsf{G_0}, 1) \wedge \mathsf{loop}(1)$ would yield the detected simple loop from Example 11.

4 Conclusion

In this paper, we have introduced a SAT-based approach for detecting simple loops in derivations of graph rewriting systems. In future, we want to investigate looping for subgraphs, i.e. we want to find derivations $G_0 \overset{*}{\Longrightarrow} G \overset{+}{\Longrightarrow} H$ where a subgraph $\overline{G}$ of G is isomorphic to a subgraph of H. This idea would cover simple looping. Moreover, we want to devise a translation to SAT for graph rewriting systems with node deletion rules.

Acknowledgements The author is grateful to Johannes Waldmann for his suggestion that the SAT-based approach to graph rewriting could be used for loop detection, to Hans-Jörg Kreowski for his suggestion that the JOIN-Theorem could be useful for Definition 10, and to the anonymous referees for their helpful comments.

References

1 Andrea Corradini, Hartmut Ehrig, Reiko Heckel, Michael Löwe, Ugo Montanari, and Francesca Rossi. Algebraic approaches to graph transformation part I: Basic concepts and double pushout approach. In Grzegorz Rozenberg, editor, *Handbook of Graph Grammars and Computing by Graph Transformation, Vol. 1: Foundations*, pages 163–245. World Scientific, 1997.

2 Hans-Jörg Kreowski. *Manipulationen von Graphmanipulationen.* PhD thesis, TU Berlin, 1978.

3 Hans-Jörg Kreowski, Sabine Kuske, and Robert Wille. Graph transformation units guided by a SAT solver. In Hartmut Ehrig, Arend Rensink, Grzegorz Rozenberg, and Andy Schürr, editors, *Proc. 5th Intl. Conference on Graph Transformations (ICGT 2010)*, volume 6372 of *Lecture Notes in Computer Science*, pages 27–42. Springer, 2010.

4 Detlef Plump. On termination of graph rewriting. In Manfred Nagl, editor, *Proc. Graph-Theoretic Concepts in Computer Science*, volume 1017 of *Lecture Notes in Computer Science*, pages 88–100, 1995.

5 Detlef Plump. Termination of graph rewriting is undecidable. *Fundamenta Informaticae*, 33(2):201–209, 1998.

6 Harald Zankl, Christian Sternagel, Dieter Hofbauer, and Aart Middeldorp. Finding and certifying loops. In Jan van Leeuwen, Anca Muscholl, David Peleg, Jaroslav Pokorný, and Bernhard Rumpe, editors, *Proc. 36th Conference on Current Trends in Theory and Practice of Computer Science (SOFSEM 2010)*, volume 5901 of *Lecture Notes in Computer Science*, pages 755–766. Springer, 2010.