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,...,n} = [n] for some n ∈ N is a finite set of nodes, numbered from 1 to n, and E ⊆ V × Σ × V is a set of labeled edges. n is called the size of G. The components V and E are also denoted by VG and EG. We call an edge (v, x, v) a sling and an edge (v, *, v′) unlabeled and omit the label * in drawings. Two edges (v1, x, v2) and (v′1, x′, v′2) are considered as an undirected edge if v1 = v′2 and v2 = v′1. A special graph is the empty graph ∅ = (∅, ∅). We call G a subgraph of H, denoted by G ⊆ H, if VG ⊆ VH and EG ⊆ EH.

Furthermore, we use injective graph morphisms for the matching. Let G, H be two graphs as defined above. An injective graph morphism from g: G → H is an injective mapping gv: VG → VH, that is structure- and label-preserving, i.e. for each edge (v, x, v′) ∈ EG, (gv(v), x, gv(v′)) ∈ EH. An injective graph morphism g: G → H yields the image g(G) = (gv(VG), gE(EG)) ⊆ H with gE(EG) = {(gv(v), x, gv(v′)) | (v, x, v′) ∈ EG} 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 $\emptyset$ to denote that the corresponding nodes have not yet been chosen. The graph on the left-hand of the arrow in Figure 2, called $L_{\text{choose}}$ in the following, is a subgraph of $G_0$. One can choose four injective mappings from $L_{\text{choose}}$ to $G_0$: $g = \{1 \mapsto 1\}$, $g = \{1 \mapsto 2\}$, $g = \{1 \mapsto 3\}$, or $g = \{1 \mapsto 4\}$.

![Figure 1](https://example.com/image1.png) ![Figure 2](https://example.com/image2.png) ![Figure 3](https://example.com/image3.png)

**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 $L_{\text{choose}}$ to $G_0$. In Figure 4 a derivation is shown. The first graph $G_1$ of the derivation is the result of applying $\text{choose}$ to $G_0$ twice by using the mappings $g = \{1 \mapsto 2\}$ and, then, $g = \{1 \mapsto 4\}$. Applying the rule $\text{remEdge}$ 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 $\text{remEdge}$ yields the last graph.

![Figure 4](https://example.com/image4.png)

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 $\Delta$. 
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 $VC = (G_0, \{\text{choose}, \text{remEdge}\}, \{\text{member}, \emptyset\})$. 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 $\emptyset$.

3 Detecting Loops in Graph Rewriting via SAT

Every propositional formula with variable set $\{\text{edge}(v, a, v', k) | (v, a, v') \in [n] \times [n] \times [n], k \in [m]\}$ 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 $f(\text{edge}(v, a, v', k)) = \text{TRUE}$. Please note, that this translation does not allow node addition or deletion. A single initial graph $G$ in the $k$th derivation step can be described by the formula

$$\text{graph}(G)(k) = \bigwedge_{(v, a, v') \in E_G} \text{edge}(v, a, v', k) \land \bigwedge_{(v, a, v') \in ([n] \times [n]) \setminus E_G} \neg \text{edge}(v, a, v', k).$$

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

$$\text{graph}(G_0)(0) = \bigwedge_{(v, a, v') \in E_0} \text{edge}(v, a, v', 0) \land \bigwedge_{(v, a, v') \in ([n] \times [n]) \setminus E_0} \neg \text{edge}(v, a, v', 0)$$

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

For a rule $r = (L \rightarrow R)$ the set of injective graph morphisms from $[\text{size}(r)]$ 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 $[n]$, $r = (L \rightarrow R)$ 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

- morph$(r, g, k) = \bigwedge_{(v, a, v') \in E_L} \text{edge}(g(v), a, g(v'), k - 1),$
- rem$(r, g, k) = \bigwedge_{(v, a, v') \in E_{L-R}} \neg \text{edge}(g(v), a, g(v'), k),$
- add$(r, g, k) = \bigwedge_{(v, a, v') \in E_R} \text{edge}(g(v), a, g(v'), k),$
- keep$(r, g, k) = \bigwedge_{(v, a, v') \in g(E_L \cup E_R)} \{\text{edge}(g(v), a, g(v'), k - 1) \leftrightarrow \text{edge}(v, a, v', k)\}$
  where $g(E_L \cup E_R) = \{(g(v), a, g(v')) | (v, a, v') \in E_L \cup E_R\},$
- apply$(r, g, k) = \text{morph}(r, g, k) \land \text{rem}(r, g, k) \land \text{add}(r, g, k) \land \text{keep}(r, g, k).$

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. $G_{k-1} \Rightarrow G_k$ if and only if there is a satisfying assignment to the formula $\text{graph}(G_{k-1})(k - 1) \land \text{apply}(r, g, k) \land \text{graph}(G_k)(k)$.

Example 7. For the rule remEdge in Figure 3, the first graph $G_1$ in Figure 4, and the graph morphism $g = \{(1 \rightarrow 4, 2 \rightarrow 3)\}$ one gets the following formulas: morph$(\text{remEdge}, g, 2) = \text{edge}(4, \text{member}, 4, 1) \land \text{edge}(3, *, 4, 1) \land \text{edge}(4, *, 3, 1), \text{rem}(\text{remEdge}, g, 2) = \neg \text{edge}(3, *, 4, 2) \land \neg \text{edge}(4, *, 3, 2),$ and add$(\text{remEdge}, g, 2) = \text{edge}(4, \text{member}, 4, 2).$
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 $\text{step}(k) = \bigvee_{r \in P, g \in \mathcal{M}(r, n)} \text{apply}(r, g, k)$. All derivation starting in $G_0$ of length $m$ are described by $\text{der}(G_0, m) = \text{graph}(G_0)(0) \land \bigwedge_{k=1}^m \text{step}(k)$. The following theorem states the connection between $\text{der}$ 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 $G_0 \xrightarrow{m} P G_T$ if and only if there is a satisfying assignment to $\text{der}(G_0, m) \land \text{graph}(G_T)(m)$.

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 $\bigvee_{m=0}^{p(n)} \text{der}(G_0, m)$.

**Example 9.** A satisfying assignment to the following subformula yields the derivation detailed in Example 3 where $c$ is an abbreviation for $\text{choose}$ and $rE$ for $\text{remEdge}$

\[
\text{graph}(G_0)(0) \land \text{apply}(c, \{1 \mapsto 2\}, 1) \land \text{apply}(c, \{1 \mapsto 4\}, 2) \land \text{apply}(rE, \{1 \mapsto 4, 2 \mapsto 3\}, 3) \land \text{apply}(rE, \{1 \mapsto 2, 2 \mapsto 3\}, 4) \land \text{apply}(rE, \{1 \mapsto 4, 2 \mapsto 2\}, 5).
\]

Let $G_0$ and $H_0$ be two graphs and $g: G_0 \rightarrow H_0$ be an injective graph morphism. Then the JOIN-Theorem in [2], p. 101 states the conditions under which a derivation $G_0 \xrightarrow{n} G_n$ can be extended to a derivation $H_0 \xrightarrow{n} H_n$. Loosely speaking, $H_0 - G_0$ is joined with each $G_i$. This result can be used to find derivations of the form $G_0 \xrightarrow{n} G_n \xrightarrow{k} G_0$, i.e. to detect loops in derivations. According to the JOIN-Theorem, the derivation $G_0 \xrightarrow{n} G_n$ would be extended to a derivation $H_0 \xrightarrow{n} H_n$. 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 $g: G \rightarrow H$ such that $G_0 \xrightarrow{\ast} P G \xrightarrow{\ast} P H$.

**Example 11.** Let us consider again Example 4 and let us assume that $\text{choose}$ has been defined in a wrong way (see Figure 5). The application of $\overline{\text{choose}}$ 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 $\overline{\text{choose}}$ can be applied infinitely often to node 2.

\[
\begin{array}{c}
\emptyset \\
\downarrow \\
\emptyset
\end{array} \quad \rightarrow \quad \begin{array}{c}
\emptyset \\
\downarrow \\
\text{member}
\end{array}
\]

\[\begin{array}{c}
\emptyset \\
\downarrow \\
\emptyset \\
\downarrow \\
\emptyset \\
\downarrow \\
\emptyset \\
\downarrow \\
\emptyset
\end{array} \quad \xrightarrow{\text{choose}} \quad \begin{array}{c}
\emptyset \\
\downarrow \\
\emptyset \\
\downarrow \\
\emptyset \\
\downarrow \\
\emptyset \\
\downarrow \\
\emptyset
\end{array}
\]

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

\[
\text{loop}(m) = \bigvee_{k=1}^{m-1} \bigvee_{g \in \mathcal{M}(n, n)} \bigvee_{(v, a, v') \in [n] \times \Sigma \times [n]} \bigvee_{(e, a, v', k) \rightarrow (e, g(v), a, g(v'), m)}
\]

Please note, that this formula generates up to $n^m$ 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 \( \varnothing \)-slings as node types such that member-nodes could only match member-nodes (the same for \( \varnothing \)-nodes).

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

\[
\text{loop\_detection}(G_0, p(n)) = \bigvee_{m=0}^{p(n)} (\text{der}(G_0, m) \land \text{loop}(m)).
\]

Simple looping corresponds to a satisfying assignment for \( \text{loop\_detection} \).

**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 \( \text{loop\_detection}(G_0, p(n)) \).

**Example 13.** A satisfying assignment to the subformula \( \text{der}(G_0, 1) \land \text{loop}(1) \) 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 \( G_0 \Rightarrow G \Rightarrow H \) where a subgraph \( \mathcal{G} \) 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


