Efficient Graph Rewriting System Using Local Event-driven Pattern Matching*

Bilal Said and Olivier Gasquet

Université Paul Sabatier, IRIT - LILaC,
118 route de Narbonne, F-31062 Toulouse cedex 9, France

Abstract. Pattern matching is the most time-cost expensive part of the graph rewriting process. When applying a given rewriting rule, most graph rewriting systems check the entire host graph for possible matches. In our modal logic theorem prover based on graph rewriting, LoTREC, we use the events that are emitted when the applied rules change locally the host graph, in order to reduce the effect of this problem by performing the match process on the relevant set of subgraphs.

Introduction

The large variety of graph transformation tools have mainly the same one-step rule application mechanism and usually differ by the techniques used for the graph pattern matching step, which is considered to be the most crucial in the overall performance of a graph transformation process. In fact, a naive implementation computes every possible mapping of the L nodes of the left-hand side of a rewriting rule to N nodes of the host graph, leading to $O(|N^L|)$ time complexity in general.

The classical approach, used in [4] for AGG [6], is to solve the pattern matching problem as a *constraint satisfaction problem*. Other systems, such as PROGRES [8]) and FUJABA [3], use *local search* techniques consisting of matching a single node by some heuristics and extending the matching step-by-step by neighboring nodes and edges. G. Varró and D. Varró introduce in [7] the *incremental update* technique that aims to keep track of all possible matchings identified by graph transformation rules in database tables, and update these tables incrementally to exploit the fact that rules typically perform only local modifications to graphs.

In the graph rewriting system of our theorem prover LoTREC, we implement an original technique that lies between these three approaches. Before running the rewriting process, we analyse the left-hand side conditions of every rule in order to precise the possible ways (i.e. search plans) a matching process could be established and achieved. During the rewriting process, we keep track of the local changes made by the rules at each step, using a special data structure called *events*. When it is called by the strategy, a rule uses these events to establish a

* This work has been partially supported by the project ARROWS of the French *Agence Nationale de la Recherche*

local pattern matching process, with respect to its different search plans. Finally, the established match is completed using a CSP like procedure that instantiates the rest of the pattern graph variables.

It is clear that this technique does not reduce the complexity of the matching process itself. Nevertheless, the pattern matching time-cost becomes $O(|k^L|)$ in the applications, where the rewriting rules are applied on at most k subgraphs at each step.

In section 1 we define our graph rewriting system. Then we present our event-driven pattern matching process in section 2. Finally we sum up with a discussion of our results.

1 The Graph Rewriting System of LoTREC

LoTREC is a theorem prover by tableau for modal logics [1], [2]. It allows the implementation of new user-defined logics, and it is capable of analysing a given input formula and building the models that verify it, and/or the counter-models that refute it. The key similarity with graph rewriting systems is that a (*Kripke*) model is very similar to attributed directed graphs, with formulas being the attributes for nodes and edges, and that tableau rules and their application are to models what rewriting rules and their application are to graphs.

In this paper, we simplify the definition of these graphs to the following:

Definition 1 (Graph). *A graph G is a tuple (V, E, F, s, t, f) where V is a finite set of nodes (or vertices) and E a finite set of arcs (or edges), $V \cap E = \emptyset$; s and t are two total mappings $s, t : E \to V$, denoting "source" and "target"; and F is a set of formulas labelling nodes and edges w.r.t. $f : V \cup E \to 2^F$.*

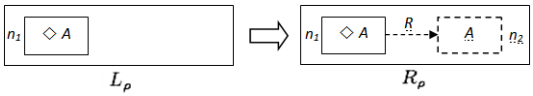
We use $S_G = V_G \cup E_G \cup F_G$ to denote the graph objects (symbols) of G, and $\mu_G \in \{s_G, t_G, f_G\}$ to designate one of the mapping functions of G.

Definition 2 (Graph morphism). *A graph morphism between two directed graphs L and G is a total mapping $\mathcal{M} : S_L \to S_G$ where the total mappings of L and G are preserved, i.e. $\forall s \in S_L : \mathcal{M}(\mu_L(s)) = \mu_G(\mathcal{M}(s))$. Note that $\mathcal{M}(S) = \{\mathcal{M}(s),\ s \in S\}$.*

Rule definition Using the Single Push Out (SPO) approach, a rewriting rule ρ is a single graph morphism that maps a *graph pattern* L_ρ, and a *replacement* graph R_ρ.

We define L_ρ and R_ρ using an appropriate simple declarative language (figure 1). In this language, we use two sets of symbols: $\mathcal{T}$, a set of *terms* (or *variable* symbols), and $\mathcal{C}$, a set of *constant* symbols. The graph objects of L_ρ and R_ρ are represented by symbols from a set $S_\rho = T_\rho \cup C_\rho$, where $T_\rho \subseteq \mathcal{T}$, and $C_\rho \subseteq \mathcal{C}$. We denote by S_{L_ρ} and S_{R_ρ} the subsets of S_ρ appearing in L_ρ and R_ρ. We also use $\mathcal{K}^n$ to denote the set of first order predicates of arity n defined over $\mathcal{T} \cup \mathcal{C}$.

The objects of S_{L_ρ} are related by a set of constraint predicates, called *elementary conditions*. A set of *elementary actions* describes both the changes that should be made on these objects, and the new objects which should be created and added to L_ρ, in order to obtain the *replacement* graph R_ρ. Formally:



LHS_ρ RHS_ρ

If	Then
c_1: `hasElement` n_1 $\Diamond A$	a_1: `createNewNode` n_1 n_2
	a_2: `link` n_1 n_2 R
	a_3: `add` n_2 A

Fig. 1. LoTREC declarative language for rewriting rules vs usual graphical notation

Definition 3 (Elementary Conditions). *An elementary condition c of a given rule ρ is a first order predicate of the form* `keyword`$(p_1, \ldots, p_n)$ *where* `keyword` $\in \mathcal{K}^n$ *and $p_i \in Param(c)$ s.t. $Param(c) \subseteq S_{L_\rho}$, for $i \in \{1, \ldots, n\}$.*

An action a is defined in the same way. We omit the parentheses and commas from these definitions for the sake of simplicity. The complete list of LoTREC predefined conditions and actions is given in [5].

Example 1. The condition c_1 given in figure 1, `hasElement` n_1 $\Diamond A$, is a well defined condition in LoTREC. It takes in two parameters: a node (n_1) and a formula ($\Diamond A$). Intuitively, such a condition is used to ensure that, in a matching subgraph g, the condition $\Diamond A \in f_g(n_1)$ is satisfied.

Definition 4 (Rewriting rule). *A rewriting rule ρ is a pair of ordered sets: $LHS_\rho = (c_1, \ldots, c_l)$ of* elementary conditions*, and $RHS_\rho = (a_1, \ldots, a_r)$ of* elementary actions*.*

Note that $|LHS_\rho|$ is equivalent to $|L_\rho|$, since n conditions at most are sufficient to instantiate n different variables in LoTREC.

In order to allow the verification of a condition c on a given graph g, a subset of $Param(c)$ should be already instantiated by elements of g. We denote this set by $Activation(c)$. During this verification, the remaining parameters will also be instantiated. The set of these parameters is called $Update(c)$. It is clear that $Param(c) = Activation(c) \cup Update(c)$, and $Activation(c) \cap Update(c) = \emptyset$.

Example 2. Considering the condition c_1, of figure 1, the instantiation of $\Diamond A$ should be preceded by the assignment of a concrete instance node to n_1. Thus it is obvious that $Activate(c_1) = \{n_1\}$ and $Update(c_1) = \{\Diamond A\}$.

Definition 5 (Local search plan). *A local search plan for a rule ρ is an ordered set $p = (c_1, \ldots, c_n)$ defined over the set of conditions LHS_ρ, such that: $n = |LHS_\rho|$, i.e. all the conditions of ρ are included in p; and $\forall\ i \in \{2, \ldots, n\}$, $Activate(c_i) \subseteq \cup_{k=2}^{i-1} Update(c_k)$, i.e. that the needed parameters for a given condition c_i are assured by the former conditions c_k s.t. $k < i$.*

We call c_1 the head of p, while $(c_2, \ldots, c_n)$ is called the tail of p (see figure 2). Two plans are equivalent if they have the same head condition, regardless the order of the tail conditions. They are called distinct otherwise.

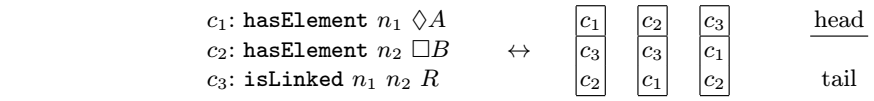


Fig. 2. Rules compilation: ordered lists of conditions are the local search plans

Rules Compilation The compilation of a rule ρ is the computation of the set of all possible distinct local search plans of ρ. This set is constructed in polynomial time before starting the rewriting process, since a rule ρ has at most $|LHS_\rho|$ plans, i.e. $|L_\rho|$ plans.

Invariant, Added and Removed Symbols We use in the sequel $Inv_\rho = S_{L_\rho} \cap S_{R_\rho}$ to denote the set of *invariant* symbols in ρ, i.e. the objects that are preserved during ρ application; $Add_\rho = S_{R_\rho} \setminus Inv_\rho$ to denote the set of newly created symbols i.e. the objects that will be *added* to the host graph; and $Rem_\rho = L_\rho \setminus Inv_\rho$ to denote the set of existing objects that will be *removed*. In general $R_\rho = (L_\rho \setminus Rem_\rho) \cup Add_\rho$, whereas in LoTREC, $R_\rho = L_\rho \cup Add_\rho$ since there are no *remove* actions, i.e. for every rule ρ, $Inv_\rho = L_\rho$.

Rule application A *transformation step* of a rewriting rule is accomplished, as shown in figure 3, by first matching L_ρ against a subgraph of the current working graph called the *redex*, and then replacing it by a copy of R_ρ.

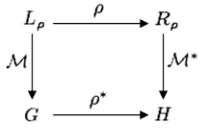


Fig. 3. Rewriting rule application in Single Push-Out approach

Definition 6 (Redex). *The redex of L_ρ w.r.t. $\mathcal{M}$ is $g \subseteq G$ s.t. $\mathcal{M}(L_\rho) = g$.*

Property 1. A redex g verifies the conditions of LHS_ρ w.r.t. the variable instantiation $\mathcal{M} : S_{L_\rho} \to S_G$, namely $g \models_{\mathcal{M}} LHS_\rho$.

Definition 7 (Replacement). *The replacement g' of a redex g, $g' = \rho^*(g) = \mathcal{M}^*(R_\rho)$, is computed w.r.t $\mathcal{M}^* : S_{R_\rho} \to S_H$; namely $g' \models_{\mathcal{M}^*} RHS_\rho$, such that: $\mathcal{M}^*(s) = \mathcal{M}(s)$, for every $s \in S_{L_\rho}$, otherwise $\mathcal{M}^*(s) = C_s$ where C_s is a new (constant) object introduced to S_H.*

It our case $g \subseteq g'$, and H can be viewed as G doted by new objects created and added by ρ, i.e. $H = G \cup \mathcal{M}^*(Add_\rho)$. However, we can extend this definition, in order to deal with *remove actions*, to : $H = (G \setminus \mathcal{M}^*(Rem_\rho)) \cup \mathcal{M}^*(Add_\rho)$ by using partial morphisms instead of total ones.

Example 3. Consider the subgraph $g = (\{N_1\}, \emptyset, \{\Diamond\Box q\}, s, t, f)$ where $f(N_1) = \{\Diamond\Box q\}$. It consists of a single node N_1 containing the formula $\Diamond\Box q$. The application of the rule of figure 1 on g leads to the graph $g' = (\{N_1, N_2\}, \{E\}, \{\Diamond\Box q, \Box q, R\}, s', t', f')$ where N_2 is a new created node, E is a new created edge, $s'(E) = N_1$, $t'(E) = N_2$, $f'(N_1) = f(N_1)$, $f'(N_2) = \{\Box q\}$, $f'(E) = \{R\}$.

Strategies In LoTREC, we make no choice among matching subgraphs, so that a transformation step consists of applying a given rule wherever it is possible in the host graph. However, the choice of the rule is made by a user-defined strategy. A strategy can call a set of rules, other strategies or a set of simple control structures, called *routines*, such as `Repeat`, `All-Rules` and `First-Rule` (for more details see [2]).

2 Event-driven Pattern Matching Process

Events When an elementary action is applied on a subgraph of the host graph, it launches an *event* of a corresponding type which embeds information about the action parameters. Thus the events are defined similarly to conditions and actions (c.f. definition 3). The graph objects embedded in an event can be viewed as *pins* used to fix the value of one or more variables of the LHS. In this way, the other variables are instantiated by searching, locally around these pins, for their convenient matches.

Example 4. The application of a_1, a_2 and a_3, of the example 3, generate respectively the events $E_1 = \texttt{NodeCreated}(N_1, N_2)$, $E_2 = \texttt{NodesLinked}(N_1, N_2, E)$ and $E_3 = \texttt{FormulaAdded}(N_2, \Box q)$.

Events Dispatching Once launched due to a given rule application, an event is dispatched at run-time to every rewriting rule, including the rule that launches it. Each rule has its own *events queue*, which is simply an ordered set of events. The events received by a given rule are enqueued at the end of its events queue.

Local Redex Initialisation and Completion When it is called by the strategy, a rule treats the events, stored in its queue since the last strategy call, with respect to their arrival order. Given an event, exploring a local search plan consists of processing the event with the head condition to establish an partial redex. Partial redexes are completed during the verification of the tail conditions.

Example 5 (Redex initialization). Processing the event E_3 of the example 4 with the head condition c_2:`hasElement` n_2 $\Box B$ of the middle search plan of figure 2, succeeds in initializing the partial redex ($n_2 = N_2$ and $B = q$) in the graph g' of example 3. Verifying C_3 on this redex is possible with ($n_1 = N_1$). Verifying C_1 on this last partial redex also succeeds, and completes it with ($A = \Box q$).

Discussion and Conclusion

A main difference between this system and general purpose systems is that the formula instantiation is integrated within the pattern matching process. This makes the experimental benchmarks difficult to design, and thus only theoretical results are discussed.

Comparing to PROGRES [8], the compilation of a rule ρ leads to $|L_\rho|!$ different plans, while in LoTREC it is bound by the size of its left-hand side $|L_\rho|$.

In addition, although PROGRES supports an incremental technique called attributes update, this technique detects only the invalidation of possible variable assignments. Thus it does not exploit the whole information embedded in the changes made on the graphs. In LoTREC we use these information to detect new possible valid assignments.

Similarly to incremental update [7], our method avoids restarting already-done pattern matching processes. However, our technique is less space and time-consuming. In fact, in our system, a rule keeps only the events occurred since its last application, and releases them all after being applied once again. Whereas the incremental update needs to keep successful matches in tables during the whole run-time. Furthermore, these tables need a considerable amount of pre-processing at initialization time and they are maintained by continuous updates, while the events stored temporary in our rules need neither initialization nor update, and thus have no additional time-cost.

In this paper we present a graph rewriting system adapted to theorem proving by tableau for modal logics. We propose a local and event-driven mechanism for limiting the effect of the graph pattern matching problem. At a given rewriting step, we establish the match process only on the local subgraphs changed by the rules during the previous step. It is clear that this technique has no advantage in specific applications where the redexes matched at each step are as numerous as possible. However, in the applications where the rules are applied alternatively in k redexes at each step where k is likely to be far less than N, our technique reduces the N factor of $O(|N^L|)$ time complexity of this problem to k.

Acknowledgements We would like to thank the anonymous reviewers for their helpful comments. We would also like to thank David Fauthoux who established the conception of the LoTREC rewriting system.

References

1. M. A. Castilho, L. Farinas del Cerro, O. Gasquet, and A. Herzig. *Modal Tableaux with Propagation Rules and Structural Rules.* Fundamenta Informaticae, 1997
2. L. F. del Cerro, D. Fauthoux, O. Gasquet, A. Herzig, D. Longin, and F. Massacci. *Lotrec: The Generic Tableau Prover for Modal and Description Logics.* Lecture Notes In Computer Science, vol. 2083. Springer-Verlag, 2001
3. T. Fischer, J. Niere, L. Torunski, and A. Zündorf. *Story Diagrams: A New Graph Rewrite Language Based on the Unified Modeling Language and Java.* Lecture Notes In Computer Science. Springer-Verlag, 1998
4. M. Rudolf. *Utilizing Constraint Satisfaction Techniques for Efficient Graph Pattern Matching.* Lecture Notes In Computer Science, Springer-Verlag, 2000
5. M. Sahade. *The conditions and actions in LoTREC language.* Tech. report, 2004
6. G. Taentzer. *Adding visual rules to object-oriented modeling techniques.* In Proceedings of Technology of Object-Oriented Languages and Systems, 1999.
7. G. Varró and D. Varró. *Graph Transformation with Incremental Updates.* Electronic Notes in Theoretical Computer Science, 2004
8. A. Zündorf. *Graph Pattern Matching in* PROGRES. Lecture Notes In Computer Science. Springer-Verlag, 1996