Technical report IRIT/RT–2009-2–FR A semantics for an event based generic tableau prover (long version)

Olivier Gasquet, Bilal Said and François Schwarzentruber

Université Paul Sabatier
Institut de recherche en informatique de Toulouse

Abstract. Unless one strictly uses standard modal logics, there is a need for a generic tool allowing easy implementation of provers or model builders for most of multi modal logics. This is what LoTREC[1] aims at. It offers a high-level language for describing tableaux calculi specified by means of rules rewriting patterns together with a strategy specifying the order in which rules have to be applied. However, a naive implementation of the search of patterns to be rewritten would be sub-optimal. LoTREC's engine has a built-in optimised way of finding them. In this paper we give the semantics of rule application and prove the correctness of their optimised treatment.

Introduction

In this paper we formally define the semantics of the language of the generic prover LoTREC ([8, 7, 10]) which uses an event-driven pattern detection, and second we prove that running with or without this optimisation is equivalent in terms of possible computations.

LoTREC is a generic platform aiming at the quick design of model builders mainly for modal logics. The only existing similar tool being TWB [2] we must specifically underline what make them different. We will not discuss proper graph rewriting tools like [16] and [9], they are definitely not easily adaptable to treat problems involving logical aspects.

People working on a modal formalisation of such and such concept are likely to be interested in the possibility of quickly implementing a prover for their logics. As an example, it could be possible for the authors of [6] to implement easily a prover for their logic of subinterval structures. LoTREC allows to handle almost any logic having a Kripke semantics, as long as a model builder for it can be expressed as a graph rewriting system. This is the main feature of LoTREC: to offer a high-level language for describing rewrite rules over a graph, together with a very simple language for describing strategies: descriptions of iterations of rules applications. This allows LoTREC to be used easily by those who model

[1] www.irit.fr/Lotrec

applications in modal logics and makes it easier to non-computer scientists to implement their own logics since only small programming skills are necessary. LoTREC may be used for "model transformation" for logics such as Public Announcement Logic [4] for example, or for model checking.

LoTREC has also great pedagogical features: on the one hand its very user-friendly interface allows to use it quickly and easily, and on the other hand, its language is so simple that it is used in the Master courses "Formalisation of Interaction with Modal Logics" to design model builders for some logics.

LoTREC is a model builder and most existing tools are satisfiability checkers. Though LoTREC can be used simply as a decision procedure, it is clear that when building models, it cannot run in optimal space at least for PSPACE logics!

Nevertheless, in the domain of graph rewriting, there is one source of time consumption: that of finding the set of patterns that can be rewritten by some given rule. Standard graph tools search them in the whole current structure at each iteration, and since in modal logics the current structures are often of exponential size w.r.t. the input formula, this search is very expensive.

In this paper, we present an optimised search for patterns that really improves the naive one; as such it has been presented at [14]. It consists essentially in computing after each rule application the set of "objects" (nodes, edges, formulas) that allows such or such rule to be applied at a further step. Note that we keep in the memory only elements (nodes, edges, formulas) of the patterns, and not the whole set of "patterns" (like done in the "incremental update" of [15]). This is achieved in JAVA by using the *event dispatching* paradigm [11].

An important aspect of programming tableaux concerns the "distance" between their theoretical formulations on paper and their implementations involving programming tricks and various optimizations that are not in the initial formulation. This may raise some doubts on completeness and correctness proofs that usually are given using the theoretical formulation. We claim that LoTREC allows to keep the tableau method implementation closer to the theoretical formulation due to its declarative language. Of course, using events in fact makes this "distance" greater, but we show that rewriting with events preserves completeness, soundness and termination.

In the first section we consider tableau rules as rewriting rules. We define the notions of rules applicability and application both in a naive system and in LoTREC and prove their equivalence. In the second section, we extend those notions and equivalence to strategies. Finally we give some experimental results.

1 Modelling tableau rules with term rewriting

In this section, we give a representation of tableau method based on term rewriting. The benefit is to have a simple general framework. First, we recall some basic notions about terms. You can also refer to [3].

1.1 Recalling terms

Definition 1 (set of terms). *Let L be a set of symbols. We define the set of terms $\mathcal{T}(L)$ over L as the smallest set such that:*

- $L \subseteq \mathcal{T}(L)$;
- *for all* $n \in \mathbb{N}$, *for all* $t_1, \ldots, t_n \in \mathcal{T}(L)$, $(t_1, \ldots, t_n) \in \mathcal{T}(L)$.

$\mathcal{C}$ denotes the infinitely countable set of constant symbols and $\mathcal{V}$ the infinitely countable set of variable symbols. We define $\mathcal{GT} = \mathcal{T}(\mathcal{C})$ the set of *ground terms* and $\mathcal{VT} = \mathcal{T}(\mathcal{C} \cup \mathcal{V})$ the set of *variable terms*.

Given a variable term $u \in \mathcal{VT}$, $Var(u)$ denotes the set of variables occurring in u. Given a set of variable terms U, $Var(U) = \bigcup_{u \in U} Var(u)$.

Definition 2 (substitution). *A substitution σ is a mapping from $\mathcal{V}$ to $\mathcal{GT}$ which extends as usual to elements of $\mathcal{VT}$. We denote by Σ the set of all possible substitutions.*

Example 1. If $W, \varphi \in \mathcal{V}, \sigma(W) = (1\ 2), \sigma(\varphi) = (p \vee q)$, then we have $\sigma'((W((\Diamond\varphi) \wedge \varphi))) = ((1\ 2)((\Diamond(p \vee q)) \wedge (p \vee q)))$.

1.2 Encoding a premodel with terms

In [7], it is shown that tableaux can be represented as graphs (w.r.t. Kripke semantics). For the sake of simplicity, we abstractly represent a graph by a set of ground terms that we call *premodel*. We introduce a set of constant symbols $\mathcal{C}$ containing symbols to deal with formulas as $\{\Diamond, \Box, \wedge, \vee, p, q, \ldots\} \subseteq \mathcal{C}$ and symbols as $\{world, link, containsformula, included \ldots\} \subseteq \mathcal{C}$ for encoding the graph structure.

Definition 3 (premodel). *A premodel is a finite set of ground terms. The set of all premodels in noted $\mathcal{PM}$.*

Example 2. $\{((world\ 1), (world\ 2), (containsformula\ 1\ (\Box p)), (1\ link\ 2)\}$ encodes the premodel $\bullet_1^{\{\Box p\}} \rightarrow \bullet_2^{\emptyset}$.

For the sake of brevity, we settle for this example and we omit the details of how we can represent a premodel by a set of terms.

1.3 Encoding tableau rule as term rewriting rule

As shown in [7], a tableau rule consists of conditions (describing a graph pattern) and actions (describing new graph objects to be added[2]). If the graph pattern is found in the premodel, then actions are performed. Conditions are modeled by two sets of variable terms: positive conditions (PC) and negative conditions (NC). PC is a set of variable terms to be unified with a subset of the premodel.

[2] In [7], we only use monotonic rules.

NC is a set of variable terms which have to be not unifiable. We may model actions by a set of variable terms (A) denoting new ground terms to be added to the premodel when the rule is applied. Instead we use $\mathcal{A} = \{A_1, \ldots, A_n\}$ since our rules can be non deterministic, i.e. the system makes a choice between different sets of terms to be added. When the rule is deterministic $card(\mathcal{A}) = 1$. Formally:

Definition 4 (Rule). *A rule r is a triple $(PC, NC, \mathcal{A})$ where:*

- $\emptyset \subsetneq PC \subseteq \mathcal{VT}$;
- $NC \subseteq \mathcal{VT}$;
- $Var(NC) \subseteq Var(PC)$;
- *a non empty finite set $\mathcal{A} \subseteq 2^{\mathcal{VT}}$ such that for all $A \in \mathcal{A}$, $Var(A) \subseteq Var(PC)$.*

We write it $r = \frac{PC,NC}{\mathcal{A}}$. Given a rule r, PC_r, NC_r and $\mathcal{A}_r$ denote respectively positive conditions set, negative conditions set and actions set of r.

Note that $PC \neq \emptyset$: we only apply rules on non empty patterns. As our rules only add elements, only positive conditions are used to instantiate variables. Negative conditions are only verified on these instances. And that is why we have the third condition $Var(NC) \subseteq Var(PC)$. We also impose $Var(A) \subseteq Var(PC)$. Indeed, new objects are denoted by skolem terms whose arguments are terms of PC (see examples 5 and 6).

In the following examples W, V, B and C denote variables. Here are two typical examples of a non-deterministic rule:

Example 3. $r = \frac{\{(\text{formula } W\ (B \vee C))\},\emptyset}{\{\{(\text{formula } W\ B)\},\{(\text{formula } W\ C)\}\}}$ is the rule dealing with the "or" operator.

Example 4. $r = \frac{\{(world\ V),(world\ W)\}}{\{\{(link\ W\ V)\},\{(link\ V\ W)\}\}}$ is the rule corresponding to the connectedness property (for instance in the modal logic S4.3).

The following example gives an incorrect encoding of the $\Diamond$-rule:

Example 5. $r = \frac{\{(world\ W),(containsformula\ W\ (\Diamond B))\},\emptyset}{\{\{(world\ V),(link\ W\ V),(containsformula\ V\ B)\}\}}$ is not a correct rule because "$Var(A) \nsubseteq Var(PC)$".

Instead of dealing with a new node V, we name it with the skolem term $(r_\Diamond\ W\ (\Diamond B))$ encoding the fact that the node has been created by the rule $r_\Diamond$ applied on the node W with the formula $\Diamond B$.

Example 6. A correct encoding of the $\Diamond$-rule shall be:
$r_\Diamond = \frac{\{(world\ W),(containsformula\ W\ (\Diamond B)\},\emptyset}{\{\{(world\ (r_\Diamond\ W\ (\Diamond B))),(link\ W\ (r_\Diamond\ W\ (\Diamond B))),(containsformula\ (r_\Diamond\ W\ (\Diamond B))\ B)\}\}}$

1.4 Rewriting rule application

Applying a rule r on a premodel T consists in:

1. finding a substitution σ unifying positive conditions of r with a subset of T;
2. verifying that negative conditions are respected;
3. and finally performing the actions when 1. and 2. succeed.

When a given substitution σ succeeds on 1. and 2., we usually say that r is applicable on T and the application of r on T will result in a new $T' = \sigma(A) \cup T$. If r is applied on T' by considering the same substitution σ, then T' will remain unchanged. While practically, we are rather interested in applying r using a different substitution σ', if there are any. This can be achieved by keeping a history of found substitutions. Nevertheless, we guarantee it by testing that the skolem terms added by the application of r and σ on T have not been already added: for all $A \in \mathcal{A}$, $\sigma(A) \not\subseteq T$. That's why we tune the usual definitions of rule *applicability* and rule *application* to ensure that a new different substitution is considered when the same rule is applied sequentially twice.

Definition 5 (rule applicability with the substitution σ). *Let $T \subseteq \mathcal{GT}$, $r = \frac{PC,NC}{\mathcal{A}}$ and $\sigma \in \Sigma$. We say that r is applicable on T with the substitution σ, denoted by $r \downarrow_\sigma T$ iff*

- $\sigma(PC) \subseteq T$;
- $\sigma(NC) \not\subseteq T$;
- *and for all $A \in \mathcal{A}$, $\sigma(A) \not\subseteq T$.*

Definition 6 (rule applicability). *We say that r is applicable on T, denoted by $r \downarrow T$ iff there exists $\sigma \in \Sigma$ such that $r \downarrow_\sigma T$.*

Definition 7 (rule application). *Let $T, T' \subseteq \mathcal{GT}$ and $r = \frac{PC,NC}{\mathcal{A}}$. We write $T \xrightarrow{r} T'$ iff there exists $\sigma \in \Sigma$, $A \in \mathcal{A}$ such that $r \downarrow_\sigma T$ and $T' = T \cup \sigma(A)$.*

Example 7. Let us consider $r_\Diamond$ from example 6 and the following premodels:

$T = \{ (world\ 1), (containsformula\ 1\ (\Diamond(\Diamond p))) \}$;

$T_2 = \{ (world\ 1), (containsformula\ 1\ (\Diamond(\Diamond p))),$
$(world\ 2), (link\ 1\ 2), (containsformula\ 2\ (\Diamond p)) \}$;

$T_3 = \{ (world\ 1), (containsformula\ 1\ (\Diamond(\Diamond p))),$
$(world\ 2), (link\ 1\ 2), (containsformula\ 2\ (\Diamond p)),$
$(world\ 3), (link\ 2\ 3), (containsformula\ 3\ p) \}$;

where $2 = (r_\Diamond\ 1\ (\Diamond(\Diamond p)))$ and $3 = (r_\Diamond\ 2\ (\Diamond p))$.

We have $T \xrightarrow{r_\Diamond} T_2 \xrightarrow{r_\Diamond} T_3$. But we do not have $T \xrightarrow{r_\Diamond} T_2 \xrightarrow{r_\Diamond} T_2$.

The following proposition recalls that our rules are monotonic and it shall be used later in the proof of theorem 1.

Proposition 1. *Let $T, T' \subseteq \mathcal{GT}$ and a rule r: $T \xrightarrow{r} T'$ implies $T \subseteq T'$.*

1.5 Discussion about pattern matching process

Let us consider Example 7. A naive rewriting system will achieve $T \xrightarrow{r_\Diamond} T_2$ by using a substitution σ such that $\sigma(W) = 1$ and $\sigma(A) = (\Diamond p)$. Then, at the rewriting step $T_2 \xrightarrow{r_\Diamond} T_3$, the system will consider again the same substitution σ, before it realizes that it will not succeed. This is because a naive system would browse the whole premodel T_2 looking up for a possible match.

Due to the high time cost of the matching process, we would rather want to reduce the number of established substitutions. And since new successful patterns may only rise from new added objects, we should keep track of these objects in order to guide the matching processes during the tableau rules application.

In LoTREC, unifications are established only on new objects. For instance, at the step $T \xrightarrow{r_\Diamond} T_2$, $(world\ 2), (link\ 1\ 2)$ and $(containsformula\ 2\ (\Diamond p)$ are reported as new objects whereas $(world\ 1), (containsformula\ 1\ (\Diamond(\Diamond p))$ are not. So at step $T_2 \xrightarrow{r_\Diamond} T_3$, the substitution σ is no more considered.

Keeping track of these new objects can be done "by hands" by the programmer of the tableau method. For instance, for the logic K, rules can be defined in such a way that they are only applied on one active world (the current world). However, in a more complex logic (e.g. K plus universal modality etc.) the notion of active world is hard to define. Thus in LoTREC, we aim to make the notion of active worlds implicit for the programmer, so he or she has to give only high-level declaration of tableau rules.

In the next subsection, we describe the event-driven pattern detection which implements this automatic management of new objects in LoTREC. Next we prove that using this automatic management our rewriting system is still sound w.r.t a naive rewriting system.

1.6 LoTREC rewriting system

In LoTREC, new objects, called *events*, are managed with the paradigm of event-based programming [11]. When a new object is added to the premodel, it is dispatched to all the rules in form of a new occurring event. Indeed, in a naive rewriting system, the state of the machine is the current premodel whereas in LoTREC a state of the machine is the current premodel and a structure defining the launched events. Thus, the notions of rule applicability and rule application become slightly different. And that is why, we shall give the precise semantics for the LoTREC rewriting system.

In the sequel, R denotes a non empty finite set of rules.

Definition 8 (state of the LoTREC machine). *A state of the LoTREC machine is a couple (T, E) where $T \in \mathcal{PM}$ and $E : R \longrightarrow 2^{\mathcal{GT}}$.*

T is the current premodel and $E(r)$ (also denoted by E_r) is the set of launched events received by the rule r. Notice that an event is simply a ground term.

At the beginning of the tableau method, we start the process with an initial state (T, E_T) where T is the initial premodel[3] and E_T contains all terms appearing in T (as if all objects of T are declared as new events). Formally:

Definition 9 (initial events). *Given $T \subseteq \mathcal{GT}$ and a set of rules R, we define the map of initial events of a premodel T by the map $E_T : R \longrightarrow 2^{\mathcal{GT}}$ as for all $\rho \in R$, $E_T(\rho) = T$*

Now, we define the notion of rule applicability and application on a given state of the LoTREC machine.

Definition 10 (rule applicability in LoTREC by e, σ). *Let (T, E) be a state of LoTREC machine, $r = \frac{PC, NC}{\mathcal{A}} \in R$ and $\sigma \in \Sigma$, $e \in \mathcal{GT}$. We say that r has been awakened by the event e and is applicable in LoTREC on (T, E) with the substitution σ, denoted by $r \downarrow_{e,\sigma}^{LoTREC} (T, E)$, iff*

- *$e \in E_r$;*
- *$e \in \sigma(PC)$;*
- *and $r \downarrow_\sigma T$.*

Practically, the application of the rule r is driven by the event e. In fact, the event e can be viewed as a *pin* that guides our quest of finding a fruitful substitution σ, making the rule r applicable on T, by establishing the matching process *locally* around the new added object e ($e \in \sigma(PC)$).

Definition 11 (rule applicability in LoTREC). *We say that r is applicable in LoTREC on (T, E), denoted by $r \downarrow^{LoTREC} T$ iff there exists $e \in E_r$, $\sigma \in \Sigma$ such that $r \downarrow_{e,\sigma}^{LoTREC} T$.*

Definition 12 (transition of the LoTREC machine). *Let (T, E) and (T', E') be two states of the LoTREC machine and $r \in R$. We write $(T, E) \overset{r}{\twoheadrightarrow} (T', E')$ iff there exists $e \in E_r$, $\sigma \in \Sigma$, $A \in \mathcal{A}_r$ such that:*

- *$r \downarrow_{e,\sigma}^{LoTREC} T$ and $T' = T \cup \sigma(A)$;*
- *and for all $\rho \in R$, $E'_\rho = E_\rho \cup \sigma(A)$.*

Intuitively, we write $(T, E) \overset{r}{\twoheadrightarrow} (T', E')$ if and only if we can find a substitution σ holding one of the events stored in $E(r)$. In this case, we apply r on T by performing the add actions and we obtain the new premodel T'. Finally we send the new created objects to every rule and we obtain E'.

In order to avoid trying to apply r on the same pattern twice, we delete already exploited events. We describe this simplification process in the following:

Definition 13 (simplification transition of the LoTREC machine). *Let (T, E) and (T, E') be two states (with the same premodel). We write $(T, E) \rightsquigarrow (T, E')$ iff for all $\rho \in R$,*

$$E'_\rho = E_\rho \setminus \{e \in E_\rho \text{ such that for all } \sigma \in \Sigma,\ e \in \sigma(PC_\rho) \text{ implies } \rho \not\downarrow_\sigma T\}.$$

where $\rho \not\downarrow_\sigma T$ means that we have not $\rho \downarrow_\sigma T$.

[3] Usually a single world with the input formula.

Example 8. We will give a remake of example 7. The process is:

$$(T, E_T) \overset{r_\diamond}{\twoheadrightarrow} (T_2, E'_2) \rightsquigarrow (T_2, E_2) \overset{r_\diamond}{\twoheadrightarrow} (T_3, E'_3) \rightsquigarrow (T_3, E_3)$$

where:

- T, T_2, T_3 are defined as before;
- E_T is such that $E_{Tr_\diamond} = T$;
- E'_2 is such that $E'_{2r_\diamond} = E_{Tr_\diamond} \cup \{(world\ 2), (link\ 1\ 2), (containsformula\ 2\ (\diamond p))\}$;
- E_2 is such that $E_{2r_\diamond} = E'_{2r_\diamond} \setminus \{(world\ 1), (containsformula\ 2\ (\diamond(\diamond p)))\}$, i.e. $E_{2r_\diamond} = \{(world\ 2), (link\ 1\ 2), (containsformula\ 2\ (\diamond p))\}$;
- E'_3 is such that $E'_{3r_\diamond} = E_{2r_\diamond} \cup \{(world\ 3), (link\ 2\ 3), (containsformula\ 3\ p)\}$;
- E_3 is such that $E_{3r_\diamond} = E'_{3r_\diamond} \setminus \{(world\ 2), (link\ 1\ 2), (containsformula\ 2\ (\diamond p))\}$, i.e. $E_{3r_\diamond} = \{(world\ 3), (link\ 2\ 3), (containsformula\ 3\ p)\}$.

Definition 14. *We write* $(T, E) \overset{r}{\twoheadrightarrow\!\!\rightsquigarrow} (T', E')$ *iff there exists* E'' *such that* $(T, E) \overset{r}{\twoheadrightarrow} (T', E'') \rightsquigarrow (T', E')$. *We write* $(T, E) \overset{r_1 \ldots r_n}{\twoheadrightarrow\!\!\rightsquigarrow} (T', E')$ *iff there exists* $((T_i, E_i))_{i \in \{2 \ldots n-1\}}$ *such that* $(T, E) \overset{r_1}{\twoheadrightarrow\!\!\rightsquigarrow} (T_2, E_2) \overset{r_2}{\twoheadrightarrow\!\!\rightsquigarrow} \ldots \overset{r_n}{\twoheadrightarrow\!\!\rightsquigarrow} (T', E')$.

1.7 Equivalence between usual rewriting system and LoTREC in terms of rules

We defined in subsection 1.4 the classical notion of rule application $T \xrightarrow{r} T'$. We also gave the semantics of rule application in our rewriting system LoTREC in subsection 1.6. In the following, we prove the equivalence of these two notions.

One direction is trivial: what is achieved in LoTREC can be achieved exactly in a classical naive rewriting system.

Proposition 2. *Let* (T, E) *and* (T', E') *be two states of the LoTREC machine and* $r \in R$. $(T, E) \overset{r}{\twoheadrightarrow} (T', E')$ *implies* $T \xrightarrow{r} T'$.

However proving the other direction is not that obvious since we are loosing some events during the simplification steps $\rightsquigarrow$. In what follows, we prove that those simplifications will not ban the applicable rules from being applied.

Theorem 1. *Let* R *be a set of rules. Let* $r_1 \ldots r_n$ *a sequence of rules of* R. *For all* $T_1, \ldots, T_n \in \mathcal{PM}$, *we have equivalence between:*

- *the classical rewriting*

$$T_1 \xrightarrow{r_1} T_2 \xrightarrow{r_2} \ldots \xrightarrow{r_{n-1}} T_n;$$

- *and, that under LoTREC, there exists* $(E_i)_{i \in \{2, \ldots, n\}}$ *such that:*

$$(T_1, E_{T_1}) \overset{r_1}{\twoheadrightarrow\!\!\rightsquigarrow} (T_2, E_2) \overset{r_2}{\twoheadrightarrow\!\!\rightsquigarrow} \ldots \overset{r_{n-1}}{\twoheadrightarrow\!\!\rightsquigarrow} (T_n, E_n).$$

Proof. $\boxed{\Uparrow}$ Straightforward with proposition 2.

$\boxed{\Downarrow}$ We prove this direction by induction on n. For $n = 1$, it is trivial. Suppose it for one $n \geq 1$. Let us prove it for $n + 1$.

Suppose $T_1 \xrightarrow{r_1} T_2 \xrightarrow{r_2} \ldots \xrightarrow{r_{n-1}} T_n \xrightarrow{r_n} T_{n+1}$. By induction there exists $(E_i)_{i\in\{2,\ldots,n\}}$ and $(E'_i)_{i\in\{2,\ldots,n\}}$ such that

$$(T_1, E_{T_1}) \overset{r_1}{\twoheadrightarrow} (T_2, E'_2) \rightsquigarrow (T_2, E_2) \overset{r_2}{\twoheadrightarrow} (T_3, E'_3) \rightsquigarrow \ldots \overset{r_{n-1}}{\twoheadrightarrow} (T_n, E'_n) \rightsquigarrow (T_n, E_n).$$

We have to prove there exists E_{n+1} and E'_{n+1} such that:

$$(T_n, E_n) \overset{r_n}{\twoheadrightarrow} (T_{n+1}, E'_{n+1}) \rightsquigarrow (T_{n+1}, E_{n+1}). \qquad (*)$$

As $T_n \xrightarrow{r_n} T_{n+1}$, there exists $\sigma \in \Sigma$ such that $r_n \downarrow_\sigma T_n$.

From now on, we will note $r = r_n$, $PC = PC_{r_n}$,$NC = NC_{r_n}$. Here we just need to prove that we can find $e \in \mathcal{GT}$ such that $r \downarrow_{e,\sigma}^{\text{LoTREC}} T_n$. First we will define $e \in \sigma(PC)$ and secondly we will prove that $e \in E_n(r)$.

As $r \downarrow_\sigma T_n$, we have $\sigma(PC) \subseteq T_n$. Consider the set $I = \{i \geq 1 \mid \sigma(PC) \subseteq T_i\}$. It is a non empty (because $n \in I$) set of positive integers. So we can consider the minimum $i_0 = min(I)$.

First let us define $e \in \sigma(PC)$:

- if $i_0 = 1$, as $\sigma(PC) \neq \emptyset$ by definition 4, we simply take $e \in \sigma(PC)$;
- if $i_0 > 1$, as $T_{i_0-1} \subsetneq \sigma(PC)$ by definition of i_0, let $e \in \sigma(PC) \setminus T_{i_0-1}$.

Secondly our aim is to prove $e \in E_n(r)$. We prove it by proving that for all $k \in \{i_0, \ldots, n\}$, $e \in E_k(r)$ by induction on k.

- Initial case: we have $e \in E_{i_0}(r)$. Indeed, if $i_0 = 1$, $E_1 = E_{T_1}$ so $e \in \sigma(PC) \subseteq T_1 = E_1(r)$. If $i_0 > 1$, we have: there exists $\sigma_{i_0-1} \in \Sigma$ such that $r_{i_0-1} \downarrow_{\sigma_{i_0-1}} T_{i_0-1}$. There exists $A \in \mathcal{A}_{r_{i_0-1}}$ such that $T_{i_0} = T_{i_0-1} \cup \sigma_{i_0-1}(A)$. As $e \in \sigma(PC) \setminus T_{i_0-1}$, we have $e \in \sigma_{i_0-1}(A)$. By definition of $\overset{r_{i_0-1}}{\twoheadrightarrow}$, we have $E'_{i_0}(r) = E_{i_0-1}(r) \cup \sigma_{i_0-1}(A)$. So, $e \in E'_{i_0}(r)$. Then we have $e \in \sigma(PC)$ and $r \downarrow_\sigma T_{i_0}$. Indeed:
  - $\sigma(PC) \subseteq T_{i_0}$;
  - $\sigma(NC) \not\subseteq T_{i_0}$; (because $\sigma(NC) \not\subseteq T_n$ and $T_{i_0} \subseteq T_n$)
  - for all $A \in \mathcal{A}$ such that $\sigma(A) \not\subseteq T_{i_0}$. (because for all $A \in \mathcal{A}$ such that $\sigma(A) \not\subseteq T_n$ and $T_{i_0} \subseteq T_n$)

  So $e \in E_{i_0}(r)$.
- Now, let us prove the induction case: for all $k \in \{i_0, n-1\}$, $e \in E_k(r)$ implies $e \in E_{k+1}(r)$. If $e \in E_k(r)$, as $E_k(r) \subseteq E'_{k+1}(r)$ we have $e \in E'_{k+1}(r)$. Then for the same reason as in the initial case we have $e \in \sigma(PC)$ and $r \downarrow_\sigma T_{k+1}$. Hence $e \in E_{k+1}(r)$.

Finally, we have $e \in E_n(r)$ and $e \in \sigma(PC)$ and $r \downarrow_\sigma T_n$: hence $r \downarrow^{\text{LoTREC}} (T_n, E_n)$! So (*) holds and this concludes the proof of theorem 1.

The following definition allows us to state propositions 3 and 4 as direct reformulation of theorem 1 that we use in the sequel:

Definition 15 (E is rich for T). *Let (T, E) be a state of the LoTREC machine. We say that E is rich for T iff there exists a sequence u of rules in R, T_0 such that $(T_0, E_{T_0}) \overset{u}{\twoheadrightarrow\!\!\rightsquigarrow} (T, E)$. (if the sequence $u = \epsilon$ it is equivalent to the condition $E = E_T$)*

"E is rich for T" means that E contains enough events to preserve rules applicability on T.

Proposition 3. *Let (T, E) a state of the LoTREC machine. If E is rich for T then $r \downarrow T$ iff $r \downarrow^{LoTREC} (T, E)$.*

Proposition 4. *Let (T, E) a state of LoTREC machine and $T' \in \mathcal{PM}$. If E is rich for T, we have $T \xrightarrow{r} T'$ iff there exists E' such that $(T, E) \overset{r}{\twoheadrightarrow} (T', E')$ (and of course E' is rich for T')*

2 Strategies

Defining strategies over rules is the traditional way to declare complex scenarios of rule applications: sequences and repetitive applications of rules. This is done in LoTREC by a simple and high-level declarative language.

In this section, we define a syntax of this language. Then we give two semantics of strategies:

- We extend the notions of applicability and applications in a naive rewriting system (definitions 6 and 7);
- We extend the notions of applicability and applications in LoTREC (definition 11 and 12).

Finally, we extend the result of theorem 1, i.e. we show that the two semantics are equivalent.

2.1 Syntax

The syntax is basically inspired from the theory of regular expressions.

Definition 16 (strategy). *Let R be a set of rules. The set of all strategies S over R is defined as the smallest set such that:*

- $R \subseteq S$;
- *if* $s_1, s_2 \in S$, *then* $s_1 \mathbf{:} s_2 \in S$;
- *if* $s_1, s_2 \in S$, *then* $s_1 \mid^{or}_{else} s_2 \in S$;
- *if* $s \in S$, $s^{\sun} \in S$.

Example 9. If $r_\diamond$ and $r_\wedge$ are two rules, $(r_\diamond \mathbf{:} r_\wedge)^{\sun} \mid^{or}_{else} r_\wedge$ is a strategy.

In order to avoid semantical ambiguity, we have introduced the new operators $\mathbf{:}$ and $^{\sun}$ instead of using the classical ; and * operators of regular expressions.

2.2 Standard semantics

A strategy consisting of one single rule r has the same definition of applicability and application. The sequence operator **:** is used to create an order between two strategies application: s_1 **:** s_2 means that we try to apply s_1 first and then we try to apply s_2. The strategy $s_1 \mid^{or}_{else} s_2$ means that we try to apply s_1 and if it is not possible we try to apply s_2. The strategy $s^{\Leftrightarrow}$ means that we apply s as long it is possible. These semantics are detailed in the following definitions.

Definition 17 (strategy applicability). *We define $s \downarrow_{str} T$ by induction:*

- *if $r \in R$, $r \downarrow_{str} T$ iff $r \downarrow T$;*
- *if $s_1, s_2 \in S$, s_1 **:** $s_2 \downarrow_{str} T$ iff $s_1 \mid^{or}_{else} s_2 \downarrow_{str} T$ iff $s_1 \downarrow_{str} T$ or $s_2 \downarrow_{str} T$;*
- *if $s \in S$, $s^{\Leftrightarrow} \downarrow_{str} T$ iff $s \downarrow_{str} T$.*

Notice that **:** and $\mid^{or}_{else}$ have the same notion of applicability but not the semantics.

Definition 18 (rewriting transition with strategy). *Given $T, T' \in \mathcal{PM}$, $s \in S$ we define $T \xrightarrow{\overline{s}} T'$ (the premodel T can be rewritten in T' by applying the the strategy s) by induction:*

1. *If $r \in R$, $T \xrightarrow{\overline{r}} T'$ iff $T \xrightarrow{r} T'$;*
2. *If $s_1, s_2 \in S$, $T \xrightarrow{\overline{s_1 \,:\, s_2}} T'$ iff:*
   (a) *there exists T'' such that $T \xrightarrow{\overline{s_1}} T''$ and $T'' \xrightarrow{\overline{s_2}} T'$;*
   (b) *or else $T \xrightarrow{\overline{s_1}} T'$ and $s_2 \not\downarrow_{str} T'$;*
   (c) *or else $s_1 \not\downarrow_{str} T$ and $T \xrightarrow{\overline{s_2}} T'$.*
3. *If $s_1, s_2 \in S$, $T \xrightarrow{\overline{s_1 \mid^{or}_{else} s_2}} T'$ iff:*
   (a) *we have $T \xrightarrow{\overline{s_1}} T'$;*
   (b) *or $s_1 \not\downarrow_{str} T$ and $T \xrightarrow{\overline{s_2}} T'$.*
4. *If $s \in S$, $T \xrightarrow{\overline{s^{\Leftrightarrow}}} T'$ iff there exists $n \in \mathbb{N}$, $T_1, T_2, \ldots T_n \in \mathcal{PM}$, such that $T \xrightarrow{\overline{s}} T_1 \xrightarrow{\overline{s}} T_2 \xrightarrow{\overline{s}} \ldots \xrightarrow{\overline{s}} T_n \xrightarrow{\overline{s}} T'$ and $s \not\downarrow_{str} T'$.*

A straightforward induction shows that the following holds:

Proposition 5. *Let $T \in \mathcal{PM}$. If there exists $T' \in \mathcal{PM}$ such that $T \xrightarrow{\overline{s}} T'$ then it implies $s \downarrow_{str} T$.*

Be careful: the other direction is false. We can have at the same time $s \downarrow_{str} T$ and no $T' \in \mathcal{PM}$ such that $T \xrightarrow{\overline{s}} T'$. This is the case when a strategy does not terminate. Thus with our settings, proving that a strategy terminates amounts to proving that there exists such a T'.

Example 10. Consider the rule $r_{ser} = \frac{\{(world\ W)\},\emptyset}{\{\{(world\ (r_{ser}\ W)),(link\ W(r_{ser}\ W))\}\}}$ corresponding to the seriality property of a Kripke frame and let $T = \{(world\ 1)\}$. We have $r_{ser}{}^{\overline{\Leftrightarrow}} \downarrow_{str} T$ because $r_{ser} \downarrow_{str} T$. But there is no T' such that $T \xrightarrow{\overline{r_{ser}{}^{\overline{\Leftrightarrow}}}} T'$ because the execution will always continue to add a new successor to the last created world.

Our strategy language is similar to TWB language inspired by Angel [13]. The only difference is that *skip* and *fail* tactics of Angel are embedded in LoTREC rules. In order to clarify the semantics of our strategy language we would compare it with programs of the logic PDL with test [12]. In fact, we can express a strategy into the PDL action language by the following translation tr:

- $tr(r) = r$ (rules are atomic actions);
- $tr(s_1 \mathbf{:} s_2) = (tr(s_1); (?[tr(s_2)]\bot \mid tr(s_2)) \mid (?([tr(s_1)]\bot); tr(s_2))$;
- $tr(s_1 \mid^{or}_{else} s_2) = tr(s_1) \mid (?([tr(s_1)]\bot); tr(s_2))$;
- $tr(s^{\overline{\Leftrightarrow}}) = tr(s)^*; ?[tr(s)]\bot$.

The reader can verify that $(s_1 \mid^{or}_{else} s_2)^{\overline{\Leftrightarrow}}$ and $(s_1^{\overline{\Leftrightarrow}} \mathbf{:} s_2)^{\overline{\Leftrightarrow}}$ have the same semantics.

2.3 Semantics in LoTREC

Similarly, we give the corresponding semantics of strategies in LoTREC.

Definition 19 (strategy applicability in LoTREC). *We define* $s \downarrow^{LoTREC}_{str} (T, E)$ *by induction:*

- *if* $r \in R$, $r \downarrow^{LoTREC}_{str} (T, E)$ *iff* $r \downarrow^{LoTREC} (T, E)$;
- *if* $s_1, s_2 \in S$, $s_1 \mathbf{:} s_2 \downarrow^{LoTREC}_{str} (T, E)$ *iff* $s_1 \mid^{or}_{else} s_2 \downarrow^{LoTREC}_{str} (T, E)$ *iff* $s_1 \downarrow^{LoTREC}_{str} (T, E)$ *or* $s_2 \downarrow^{LoTREC}_{str} (T, E)$;
- *if* $s \in R$, $s^{\overline{\Leftrightarrow}} \downarrow^{LoTREC}_{str} T$ *iff* $s \downarrow^{LoTREC}_{str} (T, E)$.

Definition 20 (LoTREC rewriting transition with strategy). *Given* (T, E), $(T', E') \in \mathcal{PM}$, $s \in S$, *we define* $(T, E) \overset{\overline{s}}{\twoheadrightarrow\!\!\rightsquigarrow} (T', E')$ *(the state* (T, E) *can be rewritten in* (T', E') *by applying the the strategy* s*) by induction:*

1. *If* $r \in R$, $(T, E) \overset{\overline{r}}{\twoheadrightarrow\!\!\rightsquigarrow} (T', E')$ *iff* $(T, E) \overset{r}{\twoheadrightarrow\!\!\rightsquigarrow} (T', E')$;
2. *If* $s_1, s_2 \in S$, $(T, E) \overset{\overline{s_1 \mathbf{:} s_2}}{\twoheadrightarrow\!\!\rightsquigarrow} (T', E')$ *iff:*
   (a) *there exists* (T'', E'') *such that* $(T, E) \overset{\overline{s_1}}{\twoheadrightarrow\!\!\rightsquigarrow} (T'', E'')$ *and* $(T'', E'') \overset{\overline{s_2}}{\twoheadrightarrow\!\!\rightsquigarrow} (T', E')$;
   (b) *or else* $(T, E) \overset{\overline{s_1}}{\twoheadrightarrow\!\!\rightsquigarrow} (T', E')$ *(and* $s_2 \not\downarrow^{LoTREC}_{str} (T', E')$*)*;
   (c) *or* $s_1 \not\downarrow^{LoTREC}_{str} (T, E)$ *and* $(T, E) \overset{\overline{s_2}}{\twoheadrightarrow\!\!\rightsquigarrow} (T', E')$.
3. *If* $s_1, s_2 \in S$, $(T, E) \overset{\overline{s_1 \mid^{or}_{else} s_2}}{\twoheadrightarrow\!\!\rightsquigarrow} (T', E')$ *iff:*
   (a) *we have* $(T, E) \xrightarrow{\overline{s_1}} (T', E')$;

(b) or $s_1 \not\downarrow_{str}^{LoTREC} (T, E)$ *and* $(T, E) \xrightarrow{\overline{s_2}} (T', E')$.

4. *If* $s \in S$, $(T, E) \overset{\overline{s^{\Leftrightarrow}}}{\twoheadrightarrow\rightsquigarrow} (T', E')$ *iff there exists* $n \in \mathbb{N}$, $(T_1, E_1), (T_2, E_2), \ldots (T_n, E_n)$, *such that* $(T, E) \overset{\overline{s}}{\twoheadrightarrow\rightsquigarrow} (T_1, E_1) \overset{\overline{s}}{\twoheadrightarrow\rightsquigarrow} (T_2, E_2) \overset{\overline{s}}{\twoheadrightarrow\rightsquigarrow} \ldots \overset{\overline{s}}{\twoheadrightarrow\rightsquigarrow} (T_n, E_n) \overset{\overline{s}}{\twoheadrightarrow\rightsquigarrow} (T', E')$ *and* $s \downarrow_{str}^{LoTREC} (T', E')$.

2.4 Equivalence between usual rewriting system and LoTREC in terms of strategies

Proposition 6. *If* $T \in \mathcal{PM}$ *and* E *is rich for* T *then* $s \downarrow_{str} T$ *iff* $s \downarrow_{str}^{LoTREC} (T, E)$.

Proof. We extend proposition 3 by induction on s.

The following theorem is exactly what is stated in theorem 1 and proposition 4 extended to the strategies case.

Theorem 2 (Equivalence Theorem). *Let* (T, E) *and* (T', E') *be two states of the LoTREC machine such that* E *is rich for* T. $T \xrightarrow{\overline{s}} T'$ *iff there exists* E' *such that* $(T, E) \overset{\overline{s}}{\twoheadrightarrow\rightsquigarrow} (T', E')$.

Proof. We show by induction on s the two following properties:

1. If E is rich for T, $T \xrightarrow{\overline{s}} T'$ iff there exists E' such that E' is rich for T' and $(T, E) \overset{\overline{s}}{\twoheadrightarrow\rightsquigarrow} (T', E')$;
2. there exists E' such that $(T, E) \overset{\overline{s}}{\twoheadrightarrow\rightsquigarrow} (T', E')$ implies $T \xrightarrow{\overline{s}} T'$.

This proof is tedious and we give just an extract of it. We give the most difficult points.

1. in case of a rule r: If $T \xrightarrow{\overline{r}} T'$, by definition 18 we have $T \xrightarrow{r} T'$. Proposition 4 gives that there exists E' rich for T' such that $(T, E) \overset{r}{\twoheadrightarrow\rightsquigarrow} (T', E')$. By definition 20, $(T, E) \overset{\overline{r}}{\twoheadrightarrow\rightsquigarrow} (T', E')$.

1. in case of a sequence:

Suppose $T \xrightarrow{\overline{s_1 : s_2}} T'$. We have either (2a), (2b) or (2c) of Definition 18. Let us consider (2a): there exists T'' such that $T \xrightarrow{\overline{s_1}} T''$ and $T'' \xrightarrow{\overline{s_2}} T'$. By induction applied on $T \xrightarrow{\overline{s_1}} T''$, we can say that: there exists (T'', E'') such that $(T, E) \overset{\overline{s_1}}{\twoheadrightarrow\rightsquigarrow} (T'', E'')$ and E'' is rich for T''. We can use induction on $T'' \xrightarrow{\overline{s_2}} T'$ and that there exists E' such that $(T'', E'') \overset{\overline{s_2}}{\twoheadrightarrow\rightsquigarrow} (T', E')$ and E' is rich for T'. Thus we obtain: $(T, E) \overset{\overline{s_1 : s_2}}{\twoheadrightarrow\rightsquigarrow} (T', E')$. Cases (2b) and (2c) are left to the reader.

This result allows to reason on strategies without being involved in optimisation details. A strategy for LoTREC is complete (resp. sound, resp. terminating) if and only if it is complete (resp. sound, resp. terminating) w.r.t. naive rewriting.

3 Experimental results

In this section, we give two kind of experimental results. First, we compare LoTREC with other existing provers. Secondly, we focus on the optimisation presented in this paper.

3.1 Comparison with other provers

LWB authors presented in [5] a benchmark for the logics K, KT and S4. For each logic they give about twenty classes of formulas characterized by different types of difficulty (non-deterministic choices, branching factor etc.). There are 21 numbered formulas in each class. The highest number corresponds to the most complex formula in a class. The authors propose to compare theorem provers by running them on the same machine over these sets of formulas. By fixing 100 seconds as a ceil for the run time, provers handling formulas with higher numbers are the better in terms of performance.

TWB authors published in [1] the comparison of TWB and LWB on the basis of the LWB benchmark. By using the same benchmark in the same conditions we give in Figure 1 the results of the comparison of LoTREC to LWB and TWB theorem provers.

How to read the charts: in the first chart above the class name "S4_branch_n", we notice that LWB can treat the 15^{th} formula within 100 seconds, TWB can handle the 14^{th} whereas LoTREC can maximum deal with the 3^{rd}. In the same chart, but considering the class "S4_S5_p", both LoTREC and LWB can treat the hardest formula, the 21^{th}, whereas TWB can only treat the 16^{th}. Hence LoTREC is generally less efficient than other theorem provers even if it does better in some classes of formulas.

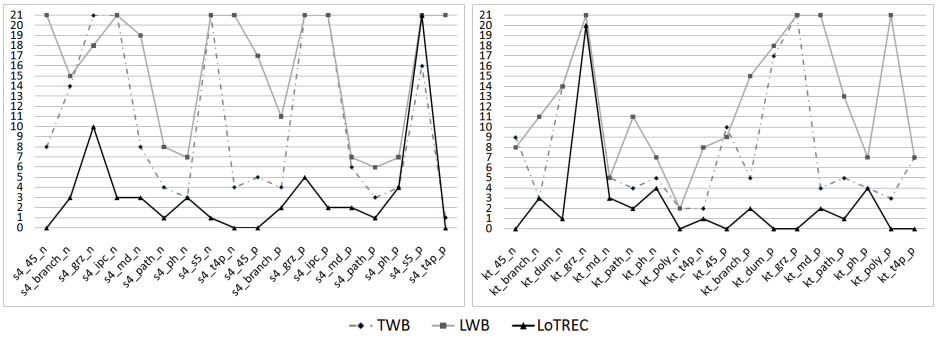

Fig. 1. Experimental results: Benchmark with other provers

3.2 Evaluation of the rewriting optimization

Now we are going to evaluate the optimization of the pattern matching process in the rewriting system of LoTREC. We propose to run LoTREC with and without the optimisation on the benchmark suite of LWB presented above and by considering the hardest formulas that LoTREC can treat within 100 seconds. Then we count for each test the number of tentatives of pattern matching established in LoTREC.

When LoTREC runs using the event-driven mechanism, this number is equal to the sum of the number of relevant events treated by the rules during all the strategy iterations. Whereas when running without events, this number is equal to the sum of the objects (nodes, formulas...) found in the premodels at each strategy iteration.

The results of these experiments are illustrated in Figure 2. In the first chart, treating the hardest formula of the class "S4_grz_n" (the 10^{th}) leads to about 116 000 tentatives of pattern matching in optimised LoTREC, and to about 7 445 000 tentatives in the naive version of LoTREC, that is to say 64 times more of unfruitful and time consuming matching processes.

We notice that generally the difference is more important when the formula that LoTREC handle is harder.

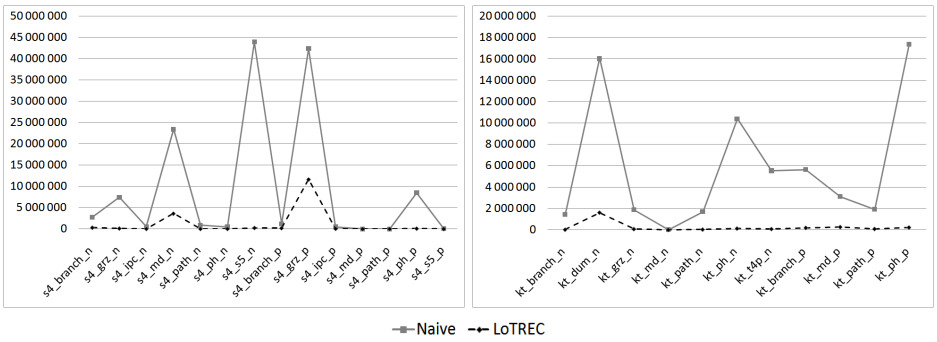

Fig. 2. Experimental results: LoTREC with and without optimisation

4 Conclusion

It was shown in [7] that proving by tableau method can be done using graph rewriting. But the semantics of rewriting was purely functional and not optimised which made it suitable for proving properties like termination, soundness and completeness. However there were a gap between this semantics and the semantics of the implementation. Filling this gap was our main aim here. We have presented the LoTREC generic tableau prover as a term rewriting system.

We gave a semantics for tableau rules and their application that we may have in a naive rewriting system. We also gave a semantics for rules application in LoTREC rewriting system which has a global and automatic optimization based on events. Then we proved that this optimisation is sound and complete. Finally we gave a semantics to strategies both in naive system and in LoTREC and we proved their equivalence which in turn ensures that general properties of naive strategies propagates to optimised ones.

5 Perspectives

One perspective would be to handle the non-deterministic choice with backtrack while keeping in memory only one premodel. In addition, using an external optimized SAT solver would improve drastically the whole rewriting process by avoiding some unfruitful non-deterministic branchings.

In the semantics presented in this paper, we only addressed "add" actions. But in fact, in LoTREC we can also perform "delete" actions (as needed for model transformation like for Public Announcement Logic [4]). For instance, we can unmark a node or a formula. Those "delete" actions are treated in the same way as "add" actions. That is to say they also launch "delete" events that will guide the pattern matching during the rule rewriting process. In this paper, those "delete" actions have not been taken into account. Thus we plan to integrate them in forthcoming extensions of this formal semantics.

References

1. Pietro Abate. *The Tableau Workbench: a framework for building automated tableau-based theorem provers.* PhD thesis, Australian National University, 2007.
2. Pietro Abate and Rajeev Goré. The tableaux work bench. In *TABLEAUX 2003, LNCS*, volume 2796, pages 230–236. Springer, 2003.
3. Franz Baader and Tobias Nipkow. *Term rewriting and all that.* Cambridge University Press, New York, NY, USA, 1998.
4. Philippe Balbiani, Hans Van Ditmarsch, Andreas Herzig, and Tiago De Lima. Tableaux for public announcement logic. *Journal of Logic and Computation Advance Access*, 2008.
5. Peter Balsiger, Alain Heuerding, and Stefan Schwendimann. A benchmark method for the propositional modal logics k, kt, s4. Technical report, Journal of Automated Reasoning, 1996.
6. David Bressolin, Valentin Goranko, Angelo Montanari, and Pietro Sala. Tableau-based decision procedures for the logics of subinterval structures over dense orderings. *Journal of Logic and Computation Advance Access (to appear).*
7. Marcos A. Castilho, Luis Fariñas del Cerro, Olivier Gasquet, and Andreas Herzig. Modal tableaux with propagation rules and structural rules. *Fundam. Inf.*, 32(3-4):281–297, 1997.
8. Luis Fariñas del Cerro, David Fauthoux, Olivier Gasquet, Andreas Herzig, Dominique Longin, and Fabio Massacci. Lotrec: The generic tableau prover for modal and description logics. In *IJCAR '01: Proceedings of the First International Joint Conference on Automated Reasoning*, volume 2083, pages 453–458. Springer-Verlag, 2001.

9. Thorsten Fischer, Jörg Niere, Lars Torunski, and Albert Zündorf. Story diagrams: A new graph rewrite language based on the unified modeling language and java. In *TAGT'98: Selected papers from the 6th International Workshop on Theory and Application of Graph Transformations*, pages 296–309. Springer-Verlag, 2000.
10. Olivier Gasquet, Andreas Herzig, Dominique Longin, and Mohamed Saade. LoTREC: Logical Tableaux Research Engineering Companion. In Bernhard Beckert, editor, *International Conference on Automated Reasoning with Analytic Tableaux and Related Methods (TABLEAUX 2005), Koblenz, Germany*, pages 318–322. Springer-Verlag, 2005.
11. Graham Hamilton. JavaBeans: Api specification. Technical report, Sun Microsystems, 1997.
12. David Harel, Dexter Kozen, and Jerzy Tiuryn. *Dynamic Logic*. MA:MIT PRESS, 2000.
13. A. P. Martin, Paul H. B. Gardiner, and Jim Woodcock. A tactic calculus-abridged version. *Formal Aspects of Computing Journal*, 8(4):479–489, 1996.
14. Bilal Said and Olivier Gasquet. Efficient graph rewriting system using local event-driven pattern matching. In *International Workshop on Graph Computation Models 2008 - GCM 08, Leicester*, 2008.
15. Gergely Varró and Dániel Varró. Graph transformation with incremental updates. *Electronic Notes in Theoretical Computer Science*, 2004. Proceedings of the Workshop on Graph Transformation and Visual Modelling Techniques (GT-VMT 2004).
16. Albert Zündorf. Graph pattern matching in progres. In *Selected papers from the 5th International Workshop on Graph Gramars and Their Application to Computer Science*, pages 454–468. Springer-Verlag, 1996.