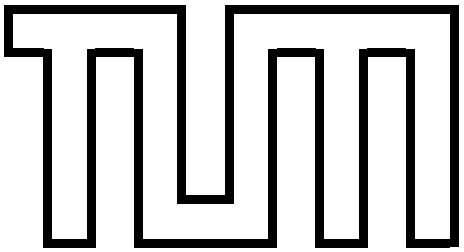


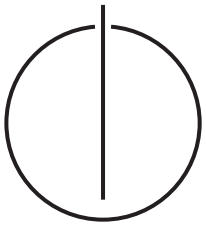
Fakultät für Informatik

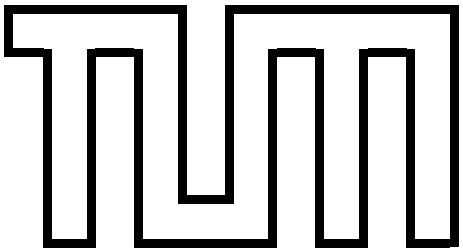
Technische Universität München

Bachelor's thesis in Informatics

Algorithms for refinement of modal process rewrite systems

Philipp Meyer





Fakultät für Informatik

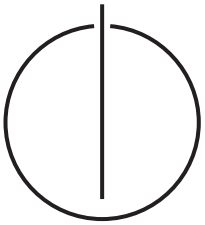
Technische Universität München

Bachelor's thesis in Informatics

Algorithms for refinement of modal process rewrite systems

Algorithmen zur Verfeinerung von modalen Prozessersetzungssystemen

Author:	Philipp Meyer
Supervisor:	Univ.-Prof. Dr. Dr. h.c. Javier Esparza
Advisor:	M. Sc. Jan Křetínský
Submission Date:	April 6, 2013



I assure the single handed composition of this bachelor's thesis only supported by declared resources.

Munich, April 6, 2013

Philipp Meyer

Contents

List of Figures

1 Introduction

Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

2 Theory

2.1 Basic definitions

Processes are terms combined with parallel or sequential composition.

The syntax here is taken from [May00] and [Esp01].

Definition 1 (Process)**.** The set of processes $\mathcal{P}$ over a set of constants $Const$ is given by

$$\frac{}{\varepsilon \in \mathcal{P}}\,(0) \qquad \frac{X \in Const}{X \in \mathcal{P}}\,(1) \qquad \frac{p \in \mathcal{P} \qquad q \in \mathcal{P}}{p \cdot q \in \mathcal{P}}\,(S) \qquad \frac{p \in \mathcal{P} \qquad q \in \mathcal{P}}{p \| q \in \mathcal{P}}\,(P)$$

Processes are considered modulo the usual structural congruence, i.e. the smallest congruence such that the operator $\cdot$ is associative, $\|$ is associative and commutative and ε is a unit for both $\cdot$ and $\|$.

From here on we will denote processes by lowercase letters $p, q, \ldots$ and single constants by uppercase letters $P, Q, \ldots$.

The class of processes that can be produced just with rule 0, 1 and S, i.e. contain no $\|$, is the class of *sequential processes* **S**. The class of processes that can be produced just with rule 0, 1 and P, i.e. contain no $\cdot$, is the class of *parallel processes* **P**.

The size of a process will be defined by the number of constants appearing in it.

Definition 2 (Size of a process)**.** The size $|p|$ of a process term p is defined by

$$\begin{aligned} |\varepsilon| &= 0 \\ |X| &= 1 \\ |p \cdot q| &= |p| + |q| \\ |p \| q| &= |p| + |q| \end{aligned}$$

2.2 Modal transition system

Modal process rewrite systems [BK12] are a modal extension of process rewrite system. There are two types of transitions, may and must transitions.

Modal transition system definition from [BK12]:

Definition 3 (Modal transition system)**.** A *modal transition system (MTS)* over an action alphabet Act is a triple $(\mathcal{P}, \dashrightarrow, \longrightarrow)$ where $\mathcal{P}$ is a set of processes $\longrightarrow \subseteq \dashrightarrow \subseteq \mathcal{P} \times Act \times \mathcal{P}$. An element $(p, a, q) \in \dashrightarrow$ is a *may transition*, also written as $p \overset{a}{\dashrightarrow} q$, and an element $(p, a, q) \in \longrightarrow$ is a *must transition*, also written as $p \overset{a}{\longrightarrow} q$.

2.3 Modal refinement

Modal refinement of one process of another. May transitions may be present in the refinement, while must transitions must be present. In the other directions, only may A must transition always has a corresponding may transition.

Out of simplicity, we regard only modal refinment for two processes from a single MTS. Modal refinement of processes from two different MTS can be reduced to this by taking the disjoint union of the MTS.

Definition 4 (Refinement)**.** Let $(\mathcal{P}, \dashrightarrow, \longrightarrow)$ be an MTS and $p, q \in \mathcal{P}$ be processes. We say that p *refines* q, written $p \leq_m q$, if there is a relation $\mathcal{R} \subseteq \mathcal{P} \times \mathcal{P}$ such that $(p, q) \in \mathcal{R}$ and for every $(p, q) \in \mathcal{R}$ and every $a \in Act$:

1. If $p \overset{a}{\dashrightarrow} p'$ then there is a transition $q \overset{a}{\dashrightarrow} q'$ s.t. $(p', q') \in \mathcal{R}$.
2. If $q \overset{a}{\longrightarrow} q'$ then there is a transition $p \overset{a}{\longrightarrow} p'$ s.t. $(p', q') \in \mathcal{R}$.

Modal refinement can also be seen as a refinement game from a pair of processes (p, q) where each side plays an attacking transition and the other a defending transition to reach a new state.

Thus if from the state (p, q) there is a transition $p \overset{a}{\dashrightarrow} p'$ or $q \overset{a}{\dashrightarrow} q'$, we will call this an *attacking transition* and a transition $q \overset{a}{\dashrightarrow} q'$ or $p \overset{a}{\longrightarrow} p'$ from that state matching the type and action of the attacking transition a *defending transition.*

Then $p \leq_m q$ holds if there no winning strategy from (p, q), i.e. a sequence of attacking transitions such that for every choice of defending transition we will reach a state

(p', q') from which there is an attacking transition but no defending transition.

2.4 Modal process rewrite system

Definition 5 (Modal process rewrite system)**.** A *process rewrite system (PRS)* over an action alphabet Act is a finite relation $\Delta \subseteq \mathcal{P} \setminus \{\varepsilon\} \times Act \times \mathcal{P}$. Elements of Δ are called *rewrite rules.* A *modal process rewrite system (mPRS)* is a tuple $(\Delta_{\text{may}}, \Delta_{\text{must}})$ where $\Delta_{\text{may}}, \Delta_{\text{must}}$ are process rewrite systems such that $\Delta_{\text{may}} \subseteq \Delta_{\text{must}}$.

An mPRS $(\Delta_{\text{may}}, \Delta_{\text{must}})$ induces an MTS $(\mathcal{P}, \dashrightarrow, \longrightarrow)$ as follows:

$$\frac{(p, a, p') \in \Delta_{\text{may}}}{p \overset{a}{\dashrightarrow} p'}\ (1) \qquad \frac{(p, a, p') \in \Delta_{\text{must}}}{p \overset{a}{\longrightarrow} p'}\ (2)$$

$$\frac{p \overset{a}{\dashrightarrow} p'}{p \cdot q \overset{a}{\dashrightarrow} p \cdot q}\ (3) \qquad \frac{p \overset{a}{\longrightarrow} p'}{p \cdot q \overset{a}{\longrightarrow} p' \cdot q}\ (4) \qquad \frac{p \overset{a}{\dashrightarrow} p'}{p \| q \overset{a}{\dashrightarrow} p \| q}\ (5) \qquad \frac{p \overset{a}{\longrightarrow} p'}{p \| q \overset{a}{\longrightarrow} p' \| q}\ (6)$$

2.5 Attack tree

Definition 6 (Attack tree)**.** An *attack tree* over a set of processes $\mathcal{P}$ is a rooted tree where each node has two kinds of children. It is given by a triple $((p, q), O, C)$, representing the tree with the root node labeled by $(p, q) \in \mathcal{P}^2$, the set of open edges O leading to states $(p', q') \in \mathcal{P}^2$ and the set of closed edges C leading to the attack trees that are children of the root node.

For an attack tree $T = ((p, q), O, C)$, we will use the short notations $T_r = (p, q)$ for the root, $T_O = O$ for the set of states open edges lead to and $T_C = C$ for the set of subtrees closed edges lead to.

The set of attack trees $\mathcal{T}$ constructable from an MTS $(\mathcal{P}, \dashrightarrow, \longrightarrow)$ are defined induc-

tively by:

$$\frac{p, q \in \mathcal{P}, p \overset{a}{\dashrightarrow} p'}{((p, q), \{(p', q') \mid q \overset{a}{\dashrightarrow} q'\}, \emptyset) \in \mathcal{T}} (1)$$

$$\frac{p, q \in \mathcal{P}, q \overset{a}{\longrightarrow} q'}{((p, q), \{(p', q') \mid p \overset{a}{\longrightarrow} q'\}, \emptyset) \in \mathcal{T}} (2)$$

$$\frac{T \in \mathcal{T} \qquad R \in \mathcal{T} \qquad R_r \in T_O}{(T_r, T_O \setminus \{R_r\}, T_C \cup \{R\}) \in \mathcal{T}} (3)$$

Rules 1 and 2 specify an initial tree for an attacking rule with the possible defensive states while rule 3 replaces an open edge to a state with a tree with that state as its root.

As we can see from the construction rules, every tree has a corresponding attacking transition from the root node, while for each defending transition applicable from that state and attacking transition there is an edge to either an open state or a subtree. Therefore we can identify nodes with attacking transitions and edges with defending transitions.

The sot of all *subtrees* of T, including T itself, are given recursively by $subtree(T) = T \cup \bigcup_{T' \in T_C} subtree(T')$.

The set of all *open states* of T are the states that have an open edge to it, that is $open(T) = \bigcup_{T' \in subtree(T)} T'_O$.

We say that a tree is *closed* if it has no open states, that is $closed(T) \quad open(T) = \emptyset$.

Lemma 1 (Tree composition)**.** *If there are attack trees T and R with $R_r \in open(T)$, then there is an attack tree S with $S_r = T_r$ and $open(S) = open(T) \setminus \{R_r\} \cup open(R)$. and $s \in T_O$*

Proof. For every subtree $T' \in subtree(T)$ with $R_r \in T'_O$, we can construct $S' = (T'_r, T'_O \setminus \{R_r\}, T'_C \cup \{R\}$ with $open(S') = T'_O \setminus \{R_r\} \cup open(R)$. Th tree S' can be added as a subtree whenever T' could be, be used in the construction rules as T. $open(S) = (\bigcup_{T' \in subtree(T) \| R_r \in T'_O} S'_O) \cup (\bigcup_{T' \in subtree(T) \| R_r \notin T'_O} T'_O) = (\bigcup_{T' \in subtree(T) \| R_r \in T'_O} T'_O \setminus \{R_r\} \cup open(R). \cup (\bigcup_{T' \in subtree(T) \| R_r \notin T'_O} T'_O \setminus \{R_r\}) = (\bigcup_{T' \in subtree(T)} T'_O) \setminus \{R_r\} \cup open(R). = open(T)\{R_r\} \cup open(R).$ □

Proof. We prove the proposition by induction on the number of child trees with an open edge to R_r, that is $n = |\{T' \in T_C \mid R_r \in open(T')\}|$:

1. $n = 0$: Then $R_r \in T_O$ and $T_r \notin open(T')$ for $T' \in T_C$ and we can construct $S = ((p, q), T_O \setminus \{R_r\}, T_C \cup \{R\})$ with $open(S) = open(T) \setminus \{R_r\} \cup open(R)$

2. $n \geq 1$: Then there is $T' \in T_C$ such that $R_r \in open(T')$. By induction hypothesis there is S' with $S'_r = T'_r$ and $open(S') = open(T') \setminus \{R_r\} \cup open(R)$

   As T' was added to T_C some point in the construction of T, we can substitute T' with S' and obtain T'' with $T''_r = T_r$, $T''_O = T_O$ and $T''_C = T_C \setminus \{T'\} \cup \{S'\}$. We have $open(T'') = T_O \cup (\bigcup_{R' \in T_C \setminus \{T'\}} open(R')) \cup open(S')$.

   If $R_r \neq open(T'')$, then $open(T'') = open(T) \setminus \{R_r\} \cup open(R)$ and we are done. Otherwise T'' has less children with an open edge to R_r, therefore we can apply the induction hypothesis on it and obtain S with $S_r = T_r$ and $open(S) = open(T'') \setminus \{R_r\} \cup open(R) = open(T) \setminus \{R_r\} \cup open(R)$.

□

Theorem 1 (Attack tree refinement)**.** *For an MTS* $(\mathcal{P}, \dashrightarrow, \longrightarrow)$ *and processes* $p, q \in \mathcal{P}$*:*

$$(p \leq_m q) \iff \neg \exists T \in \mathcal{T} : T_r = (p, q) \wedge closed(T)$$

Proof. $\Longrightarrow$: Assume $p \leq_m q$. Then there is a refinement relation $\mathcal{R}$. To show that for $(p, q) \in \mathcal{R}$ there is no closed tree from (p, q), we show the contraposition for any T, that is if T is closed, then $T_r \notin R$.

Recall that for any T there is an attacking transition from T_r and the edges correspond to the fitting defending transition. Further if T is closed, we have $T_O = \emptyset$ and every $T' \in T_C$ is also closed.

Now we show the proposition by induction over the number of subtrees $|subtrees(T)|$:

1. $|subtrees(T)| = 0$: Then there is an attacking transition and as $T_C = \emptyset$ there is no defending transition, therefore $(p, q) \notin \mathcal{R}$.

2. $|subtrees(T)| \geq 1$: Then there is an attacking rule and for every defending transition leading to (p', q'), there is an edge to a closed tree T' with $T'_r = (p', q')$. T' is a proper subtree of T and has less subtrees itself, so by induction hypothesis we have $(p', q') \notin \mathcal{R}$ and therefore $(p, q) \notin \mathcal{R}$.

$\Longleftarrow$: Assume that there is no closed attack tree T with $T_r = (p, q)$. To show $p \leq_m q$, we show that $\mathcal{R} := \{(p', q') \mid \neg\exists T : T_r = (p', q') \wedge closed(T)\}$ is a valid refinement relation with $(p, q) \in \mathcal{R}$.

For any attacking transition and $(p, q) \in \mathcal{R}$, by inference rule 1 or 2 there exists an attacking tree T with $T_r = (p, q)$. From all such T with $T_r = (p, q)$, choose one where T_O is minimal with regard to the inclusion order. There exists $(p', q') \in T_O$ with $(p', q') \in \mathcal{R}$, because otherwise there would be a closed attack tree T' with $T'_r = (p', q')$ and by inference rule 3 we would get $T'' = (T_r, T_O \setminus \{p', q'\}, T_C \cup \{T'\})$ with $T''_O = T_O \setminus \{(p', q')\} \subsetneq T_O$ in contradiction to the minimality of O. So for the attacking transition from (p, q) there is a defending transition to (p', q') with $(p', q') \in \mathcal{R}$. $\square$

2.6 Visibly pushdown automaton

Definition 7 (Visibly pushdown automaton)**.** A PRS Δ over the action alphabet Act is a *visibly pushdown automaton (vPDA)* if there is a partition $Act = Act_r \uplus Act_i \uplus Act_c$ such that every rule $(p, a, p') \in \Delta$ has the form:

$$p = P \cdot S \qquad \text{and} \qquad p' = \begin{cases} Q & \text{if } a \in Act_r \quad \text{(return rule)} \\ Q \cdot T & \text{if } a \in Act_i \quad \text{(internal rule)} \\ Q \cdot T \cdot R & \text{if } a \in Act_c \quad \text{(call rule)} \end{cases}$$

A *modal visibly pushdown automaton (mvPDA)* is then an mPRS $(\Delta_{\text{may}}, \Delta_{\text{must}})$ such that Δ_{may} and Δ_{must} are vPDA.

For mvPDA we will define a relation similiar which allows us to represent a parts of the attack tree for the corresponding MTS. These are attack rules.

Definition 8 (Attack rule)**.** An *attack rule* is a tuple $((p, q), S)$ with $p, q \in \mathcal{P}$ and $S \subseteq \mathcal{P}$. It is written as $(p, q) \longrightarrow_a S$

For an mvPDA $(\Delta_{\text{may}}, \Delta_{\text{must}})$, the attack rules obtainable from the rewrite rules are given

by:

$$\frac{(p,a,p') \in \Delta_{\text{may}}}{(p,q) \longrightarrow_a \{(p',q') \mid (q,a,q') \in \Delta_{\text{may}}\}} \ (1)$$

$$\frac{(q,a,q') \in \Delta_{\text{must}}}{(p,q) \longrightarrow_a \{(p',q') \mid (p,a,p') \in \Delta_{\text{must}}\}} \ (2)$$

$$\frac{(p,q) \longrightarrow_a \{(p',q')\} \uplus S \qquad (p',q') \longrightarrow_a S' \qquad \forall (p'',q'') \in S' : |p''| = 1}{(p,q) \longrightarrow_a S \cup S'} \ (3)$$

$$\frac{(p,q) \longrightarrow_a \{(p' \cdot P, q' \cdot Q)\} \uplus S \qquad (p',q') \longrightarrow_a S' \qquad \forall (p'',q'') \in S' : |p''| = 1}{(p,q) \longrightarrow_a S \cup \{(p'' \cdot P, q'' \cdot Q) \mid (p'',q'') \in S'\}} \ (4)$$

Due to the constraints on the rewrite rules of an mvPDA and the construction of the attack rules, we can see that for any rule $(p,q) \longrightarrow_a S$, it holds that $|p| = |q| = 2$ and for any $(p',q') \in S$ that $1 \leq |p'| = |q'| \leq 3$.

When the rules 3 and 4 always combine a rule $(p,q) \longrightarrow_a S \uplus \{(p',q')\}$ and a rule $(p',q') \longrightarrow_a S'$ on right, it always holds that $|p'| = 2$ or $|p'| = 3$ and for all $(p'',q'') \in S'$ $|p''| = 1$. We will call a rule $p \longrightarrow_a S$ a *right-hand side* rule if $\forall (p',q') \in S : |p'| = 1$ and otherwise a *left-hand side* rule. This partitions the set of rules into two classes.

Lemma 2. *Given an MTS generated by a mvPDA, for $|p| \geq 2$, $|q| \geq 2$, p, q sequential and any $s, t \in \mathcal{P}$:*

$$p \xrightarrow{a} p' \iff p \cdot s \xrightarrow{a} p' \cdot s \qquad \textit{and} \qquad q \xrightarrow{a} q' \iff q \cdot t \xrightarrow{a} q' \cdot t$$

Proof. $\Longrightarrow$: Follows directly from the induction rules of an MTS from an mPRS.

$\Longleftarrow$: In the inference chain for $p \cdot s \xrightarrow{a} p' \cdot s$, there is a $(r,a,r') \in \Delta_{\text{may}}$ which was used to obtain that rule with $p \cdot s = r \cdot s'$ and $p' \cdot s = r' \cdot s'$. As $|r| \leq |p|$, $|r'| \leq |p'$ and p, p', r, r' are all sequential, there is s'' with $p = r \cdot s''$ and $p' = r \cdot s''$. Then we can infer the transition $r \cdot s'' \xrightarrow{a} r' \cdot s'' = p \xrightarrow{a} p'$. As $|r| \leq |p \cdot s|$ and both p and r are sequential, we have $r = p$ The same holds for $q \cdot t \longrightarrow q' \cdot t$. □

Corollary 1. *Given an MTS generated by a mvPDA, for $|p| \geq 2$, $|q| \geq 2$, p, q sequential and any $s, t \in \mathcal{P}$:*

There is an attack tree T with $T_r = (p,q)$ exactly if there is an R with $R_r = (p \cdot s, q \cdot t)$ and $open(R) = \{(p' \cdot s, q' \cdot t) \mid (p',q') \in open(T)\}$.

Definition 9 (Partition of an attack tree)**.** A partition P of an attack tree T is given by a set of subtrees $P \subseteq subtree(T)$ with $T \in P$.

For $R_1, R_2 \in P$, we define a partial ordering $R_1 \leq R_2 \iff R_1 \in subtree(R_2)$ and consequently $R_1 < R_2 \iff R_1 \leq R_2 \wedge R_1 \neq R_2$. We define the partition successors of $R \in P$ given P as $succ_P(R) = \{R' \in P \mid R' < R \wedge \neg\exists R'' : R' < R'' \wedge R'' < R\}$.

Definition 10 (Part represented by an attack rule)**.** A subtree $R \in P$ in a partition is said to be *represented* by an attack rule $(p, q) \longrightarrow_a S$ if there exist $s, t \in \mathcal{P}$ such that $T_r = (p \cdot s, q \cdot t)$ and $\{R'_r \mid R' \in succ_P(R)\} = \{(p' \cdot s, q' \cdot t) \mid (p', q') \in S\}$

Theorem 2. *For an mvPDA $(\Delta_{may}, \Delta_{must})$ with its induced MTS $(\mathcal{P}, \dashrightarrow, \longrightarrow)$, it holds that for any $P, S, Q, R \in Const$:*

$$\exists T : T_r = (P \cdot S, Q \cdot R) \wedge closed(T) \iff (P \cdot S, Q \cdot T) \longrightarrow_a \emptyset$$

Proof. $\Longrightarrow$: Assume T to be closed tree with $T_r = (P \cdot S, Q \cdot T)$.

We show that if there is a n rules $\{a_1, ..., a_n\}$ such that there is a partition $\{R_1, ..., R_n\}$ of T with each R_i being represented by a_i, then there is a rule representing T.

We show that by induction on n:

1. $n = 1$: Then $R_1 = T$ and a_1 represents T.

2. $n > 1$: Let R_1 be the subtree with $R_r = (P \cdot S, Q \cdot T)$

   As $n > 1$, there is $R' \in succ_P(R)$ with $R'_r = (p', q')$. Then for $a_1 = (P \cdot S, Q \cdot T) \longrightarrow_a S$ we have by representation $R'_r \in S$.

   We have $|p'| = |q'| \geq 2$, as otherwise there would be no rule applicable from that state and therefore R' would not exist. So a_1 is a left-hand side rule.

   For every subtree $R \in P$ with $succ_P(R) = \emptyset$, we have $a_i = (p, q) \to \emptyset$, so that is a right-hand side rule. Every path in T eventually leads to such a subtree.

   Then by following the successors of the subtrees from R_1, we will eventually come to a subtree R succeded by a subtree S such that b_i is a left-hand side rule and b_j is a right-hand side rule.

   The partition $P' = P \setminus \{S\}$ then is again a partition of T where $succ_{P'}(R) = succ_P(R) \setminus \{S\} \cup succ_P(S)$ and other successors are unchanged. We now show that we can construct a rule representing R_0:

Let $b_i = (p,q) \longrightarrow_a S$ and $b_j = (p',q') \longrightarrow_a S'$. By the representation of b_i for R and $S_r \in \{R'_r \| R' \in succ_P(R)\}$, there is $s,t \in \mathcal{P}$ and $(p'',q'') \in S$ with $S_r = (p'' \cdot s, q'' \cdot t)$. By the representation of b_j for S, there is $s',t' \in \mathcal{P}$ with $S_r = (p' \cdot s', q' \cdot t')$.

Then $(p'' \cdot s, q'' \cdot t) = (p' \cdot s', q' \cdot t')$. As $2 \leq |p''| = |q''| \leq 3$ and $|p'| = |q'| = 2$ either $s = s'$ and $t = t'$ or $P \cdot s = s'$ and $Q \cdot t = t'$ for some $P, Q \in Const$.

In the first case, $(p',q') = (p'',q'')$, and we can apply rule 3 to obtain $(p,q) \longrightarrow_a S\{(p'',q'')\} \cup S'$. With $\{(p' \cdot s, q' \cdot t) \mid (p',q') \in S \setminus \{(p'',q'')\} \cup S'\} = succ_{P'}(R)$, it represents R in P'.

In the second case, $(p' \cdot P, q') = (p'',q'')$, and we can apply rule 4 to obtain $(p,q) \longrightarrow_a S \cup \{(p'' \cdot P, q'' \cdot Q) \mid (p'',q'') \in S'\}$ With $\{(p' \cdot s, q' \cdot t) \mid (p',q') \in S \setminus \{(p'',q'')\}\} \cup \{(p'' \cdot P \cdot s, q'' \cdot Q \cdot t) \mid (p'',q'') \in S'\} = succ_{P'}(P_i)$, it represents R in P'.

Then as P' is a partition for T having a rule representing each part with $n-1$ element, we can apply the induction hypothesis and obtain a rule representing T.

Now we need to show there is a partition represented by attack rules. If we initially take $P = subtrees(T)$, for each $R \in P$ we have: There is an attacking transition from R_r which induced R. As $succ_P(R) = R_C$, for each $R' \in succ_P(R)$ there is a fitting defending transition to R'_r and vice-versa. If $p \cdot s \overset{a}{\dashrightarrow} p' \cdot s$ was induced by $(p,a,p') \in \Delta_{\text{may}}$ and each $q \cdot t \overset{a}{\dashrightarrow} q' \cdot t$ was induced by $(q,a,q') \in \Delta_{\text{may}}$, then $(p,q) \longrightarrow_a \{(p',q') \mid (q,a,q')\}$ represents R

Finally for a rule $(p,q) \longrightarrow_a S$ representing the closed tree T with $T_r = (P \cdot S, Q \cdot T)$, necessarily $(p,q) = (P \cdot S, Q \cdot T)$ and $S = \emptyset$.

$\Longleftarrow$: We show that if $(p,q) \longrightarrow_a S$, then there is a tree T with $T_r = (p,q)$ such that $open(T) = S$ by induction on the construction of $(p,q) \longrightarrow_a S$:

1. It was constructed by rule 1 from $(p,a,p') \in \Delta_{\text{may}}$. Then there is an attacking transition $p \overset{a}{\dashrightarrow} p'$ and for every $(q,a,q') \in \Delta_{\text{may}}$ there is an induced defending transition $q \overset{a}{\dashrightarrow} q'$. Then $S = \{(p',q') | q \overset{a}{\dashrightarrow} q'\}$ and by attack tree inference rule 1 there is $T = ((p,q), S, \emptyset)$ with $open(T) = S$.

2. It was constructed by rule 2 from $(q,a,q') \in \Delta_{\text{must}}$. Then there is an attacking transition $q \overset{a}{\longrightarrow} q'$ and for every $(p,a,p') \in \Delta_{\text{may}}$ there is an induced defending transition $p \overset{a}{\longrightarrow} p'$. Then $S = \{(p',q') | p \overset{a}{\dashrightarrow} p'\}$ and by attack tree inference rule 2 there is $T = ((p,q), S, \emptyset)$ with $open(T) = S$.

3. It was constructed by rule 3 from $(p, q) \longrightarrow_a \{(p' \cdot P, q' \cdot Q)\} \uplus S''$ and $(p', q') \longrightarrow_a S'$ with $S = S'' \cup S'''$ and $S''' = \{(p'' \cdot P, q'' \cdot Q) \mid (p'', q'') \in S'\}$. Then by induction hypothesis there is a tree T' with $T'_r = (p', q')$ and $open(T') = S'$ and a tree T'' with $T''_r = (p, q)$ and $open(T'') = S'' \uplus \{(p' \cdot P, q' \cdot Q)\}$. By applying lemma 1 on T' there is a tree T''' with $T'''_r = (p' \cdot P, q' \cdot Q)$, $open(T''') = O''' \uplus \{(p' \cdot P, q' \cdot Q)\}$ and $O''' = \{(p'' \cdot P, q'' \cdot Q) \mid (p'', q'') \in S'\} = S'''$. By applying lemma 1 on T'' and T''' there is a tree T with $T_r = (p, q)$ and $open(T) = S'' \cup S''' = S$.

4. It was constructed by rule 4 from $(p, q) \longrightarrow_a S'' \uplus \{(p', q')\}$ and $(p', q') \longrightarrow_a S'$ with $S = S'' \cup S'$. Then by induction hypothesis there is a tree T' with $T'_r = (p', q')$ and $open(T') = S'$ and a tree T'' with $T''_r = (p, q)$ and $open(T'') = S'' \uplus \{(p', q')\}$. By applying lemma 1 on T' and T'' there is a tree T with $T_r = (p, q)$ with $open(T) = S'' \cup S' = S$.

Therefore if $(P \cdot S, Q \cdot T) \longrightarrow_a \emptyset$, then there is a tree T with $T_r = (P \cdot S, Q \cdot T)$ and $open(T) = \emptyset$. □

3 The algorithm

3.1 Description

Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

3.2 Implementation

Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur

Figure 3.1: Algorithm for calculating the basic attack rules on mvPDAs

```
1: function AttackRules(mvPDA = (Δ_may, Δ_must))
2:     rules ← ∅
3:     for P, Q, S, T ∈ Const, a ∈ Act, type ∈ {may, must} do
4:         if type = may then
5:             lhs ← (P · S, Q · T)          ▷ Attack from left-hand side for may rules
6:         else
7:             lhs ← (Q · T, P · S)          ▷ Attack from right-hand side for must rules
8:         end if
9:         for (P · S, a, p′) ∈ Δ_type do
10:            rhs ← ∅
11:            for (Q · T, a, q′) ∈ Δ_type do
12:                if type = may then
13:                    newRhs ← (p′, q′)
14:                else
15:                    newRhs ← (q′, p′)
16:                end if
17:                rhs ← rhs ∪ {newRhs}
18:            end for
19:            rules ← rules ∪ {(lhs, rhs)}
20:        end for
21:    end for
22:    return rules
23: end function
```

sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

3.3 Soundness and completeness

The set of attack rules over a finite set of constants is also finite. As the algorithm never adds a rule twice to its set of rules, it will terminate.

Figure 3.2: Algorithm for combining attack rules

```
1: function Combine(lhsRule = (lhs, lhsRhsSet), rhsRule = (rhsLhs, rhsSet))
2:     rules ← ∅
3:     if ∀rhs ∈ rhsSet : size(rhs) ≤ 1 then
4:         for lhsRhs ∈ lhsRhsSet : lhsRhs = rhsLhs · p do
5:             newRhs ← (lhsRhsSet \ lhsRhs) ∪ {rhs · p | rhs ∈ rhsSet}
6:             rules ← rules ∪ {(lhs, newRhs)}
7:         end for
8:     end if
9:     return rules
10: end function
```

Figure 3.3: Refinement algorithm for mvPDAs

```
1: function VPDARefinement(P · S, Q · T, mvPDA)          ▷ P · S ≤_m Q · T
2:     initial ← [P · S, Q · T]
3:     rules ← AttackRules(mvPDA)
4:     while ∃lhsRule, rhsRule ∈ rules : Combine(lhsRule, rhsRule) ∉ rules do
5:         rules ← rules ∪ Combine(lhsRule, rhsRule)
6:     end while
7:     return (initial, ∅) ∈ rules
8: end function
```

With termination, from theorem and theorem it follows that it returns **true** if and only if $P \cdot S \leq_m Q \cdot T$

Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

3.4 Runtime

Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur

sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

3.5 Optimizations

Worklist algorithm Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

Hash map lookup Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

Keeping only minimial rules Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

Heuristic for combining rules Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

Reachable state exploration Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad

minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

Early stopping Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

3.6 Input and output

Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

3.6.1 Input grammar

Whitespace is needed between keywords, but otherwise ignored.

Common definitions

⟨*letter*⟩ ::= `a` | ... | `z` | `A` | ... | `Z` ⟨*digit*⟩ ::= `0` | ... | `9` ⟨*id*⟩ ::= ⟨*letter*⟩(⟨*letter*⟩ | ⟨*digit*⟩)*

Process definition

⟨*process*⟩ ::= ⟨*empty*⟩ | ⟨*constant*⟩ | ⟨*parallel*⟩ | ⟨*sequential*⟩ | `(` ⟨*process*⟩ `)` ⟨*empty*⟩ ::= `_` ⟨*constant*⟩ ::= ⟨*id*⟩ ⟨*parallel*⟩ ::= ⟨*process*⟩ `.` ⟨*process*⟩ ⟨*sequential*⟩ ::= ⟨*process*⟩ `|` ⟨*process*⟩

Rule definition

⟨*rule*⟩ ::= ⟨*process*⟩ ⟨*action*⟩ ⟨*rule_type*⟩ ⟨*process*⟩ ⟨*action*⟩ ::= ⟨*id*⟩ ⟨*rule_type*⟩ :: = ⟨*may_rule*⟩
| ⟨*must_rule*⟩ ⟨*may_rule*⟩ :: = ? ⟨*must_rule*⟩ :: = !

mPRS definition

⟨*mprs*⟩ ::= `mprs` ⟨*id*⟩ `[` ⟨*refinement*⟩ ⟨*rules*⟩ `]` ⟨*refinement*⟩ ::= ⟨*process*⟩ `<=` ⟨*process*⟩
⟨*rules*⟩ ::= ⟨*rule*⟩*

Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

3.6.2 Calling the programm

3.6.3 Output of the programm

Error codes Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

3.7 Performance evaluation

Refining processes Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

Non-refining processes Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

Random PRS Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

3.8 Example

Figure 3.4 and 3.5 define two mvPDA. The corresponding may transitions for the must transitions are implied. The problem is to decide whether $p \cdot S \leq_m q \cdot S$.

$$
\begin{aligned}
P \cdot S &\xrightarrow{\text{coin}} P \cdot M \cdot S \\
P \cdot M &\xrightarrow{\text{coin}} P \cdot M \cdot M \\
P \cdot M &\xrightarrow{\text{tea}} T \\
P \cdot M &\xrightarrow{\text{coffee}} c \\
T \cdot M &\xrightarrow{\text{tea}} T \\
T \cdot S &\xrightarrow{\text{coin}} P \cdot M \cdot S \\
c \cdot M &\xrightarrow{\text{coffee}} c \\
c \cdot S &\xrightarrow{\text{coin}} P \cdot M \cdot S
\end{aligned}
$$

Figure 3.4: mvPDA for process $P \cdot S$

$$
\begin{aligned}
Q \cdot S &\overset{\text{coin}}{\dashrightarrow} Q \cdot T \cdot S \\
Q \cdot S &\overset{\text{coin}}{\dashrightarrow} Q \cdot C \cdot S \\
Q \cdot T &\overset{\text{coin}}{\dashrightarrow} Q \cdot T \cdot T \\
Q \cdot C &\overset{\text{coin}}{\dashrightarrow} Q \cdot C \cdot C \\
Q \cdot T &\xrightarrow{\text{tea}} Q \\
Q \cdot T &\overset{\text{coffee}}{\dashrightarrow} Q \\
Q \cdot C &\overset{\text{tea}}{\dashrightarrow} Q \\
Q \cdot C &\xrightarrow{\text{coffee}} Q
\end{aligned}
$$

Figure 3.5: mvPDA for process $Q \cdot S$

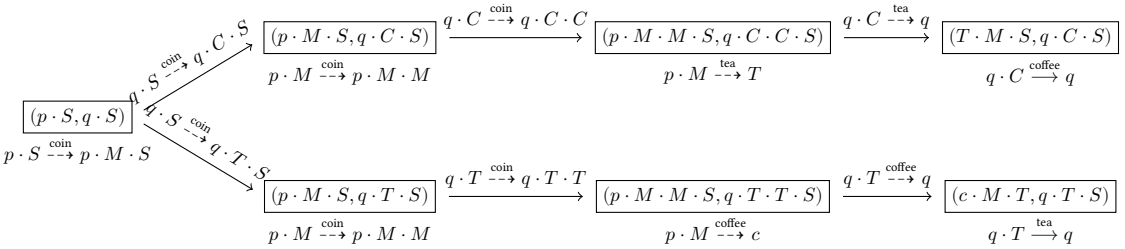


Figure 3.6: Tree for winning strategy

Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur

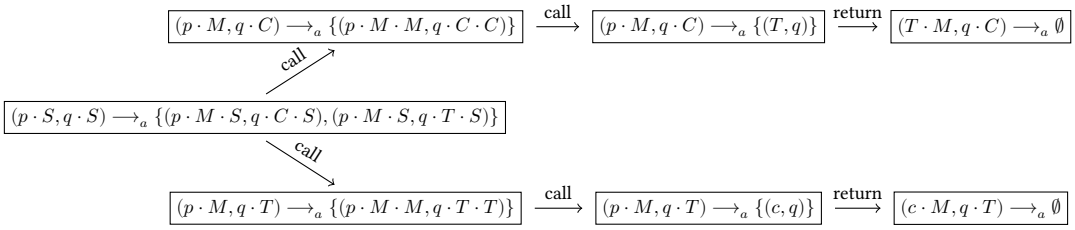


Figure 3.7: Tree for winning strategy with attack rules

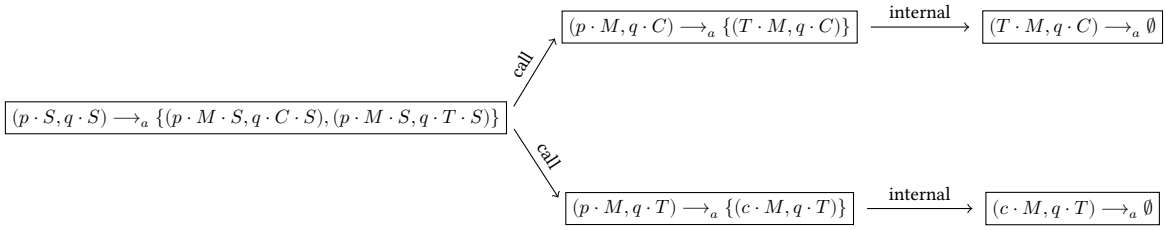


Figure 3.8: Merged tree for winning strategy with attack rules after one step

sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

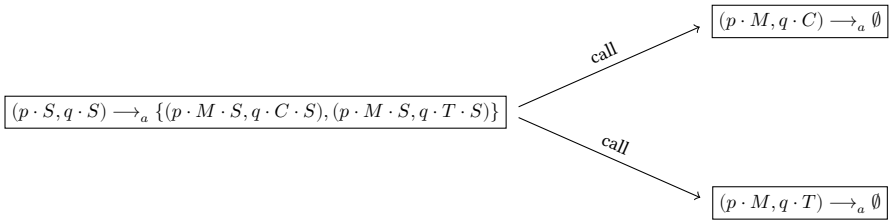


Figure 3.9: Merged tree for winning strategy with attack rules after two steps

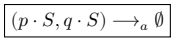


Figure 3.10: Final merged tree for winning strategy

4 Conclusion

4.1 Results of this work

Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

4.2 Further extensions

Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur.

Bibliography

[BK12] Nikola Benes and Jan Kretínský, *Modal process rewrite systems*, ICTAC (Abhik Roychoudhury and Meenakshi D'Souza, eds.), Lecture Notes in Computer Science, vol. 7521, Springer, 2012, pp. 120--135.

[BKLS09] Nikola Benes, Jan Kretínský, Kim Guldstrand Larsen, and Jirí Srba, *On determinism in modal transition systems*, Theor. Comput. Sci **410** (2009), no. 41, 4026--4043.

[Esp01] Javier Esparza, *Grammars as processes*, Lecture Notes in Computer Science **2300** (2001), 277--298.

[May00] Richard Mayr, *Process rewrite systems*, Inf. Comput **156** (2000), no. 1-2, 264--286.