



HAL
open science

SAT Solvers for Queries over Tree Automata with Constraints

Pierre-Cyrille Héam, Vincent Hugot, Olga Kouchnarenko

► **To cite this version:**

Pierre-Cyrille Héam, Vincent Hugot, Olga Kouchnarenko. SAT Solvers for Queries over Tree Automata with Constraints. CSTVA'10, 2nd Int. Workshop on Constraints in Software Testing Verification and Analysis - co-located with ICST'2010, 2010, France. hal-00563313

HAL Id: hal-00563313

<https://hal.science/hal-00563313v1>

Submitted on 4 Feb 2011

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

SAT Solvers for Queries over Tree Automata with Constraints

Pierre-Cyrille Héam and Vincent Hugot and Olga Kouchnarenko
INRIA/Cassis and University of Franche-Comté
Besançon, France

Email: pcheam@lifc.univ-fcomte.fr, vhugot@edu.univ-fcomte.fr, okouchnarenko@lifc.univ-fcomte.fr

Abstract—Tree automata turned out to be a very convenient framework for modeling and proving properties on infinite systems like communication protocols, Java programs and also in the context of XML programming. Unfortunately, these works are not always supported by efficient verification and validation tools.

This paper investigates the use of two SAT solvers — `MinisAT2` and `picoSAT` — to evaluate queries over tree automata with global equality and disequality constraints (TAGEDs for short). Unlike general tree automata, TAGEDs allow to express constraints useful for e.g., evaluating queries over XML documents, like 'in the document, two nodes do not have the same key'. These queries being based on the membership problem which is NP-complete for TAGEDs, we propose an efficient SAT encoding of the membership problem for TAGEDs and we show its correctness and soundness. The paper reports on the experimental results, and implementation details are given.

I. INTRODUCTION

Tree automata turned out to be a very convenient way for modeling and proving properties on infinite systems like communication protocols [1]–[3], multi-threaded Java byte code programs [4], [5], etc. Moreover, numerous recent works on analysis of structured XML-like documents and on validation of their transformations [6]–[9] exploit tree automata for their encoding.

For all above-mentioned applications, it is important to express constraints like 'in the term, two subterms do not have the same leaf'. For example, when evaluating queries over XML documents, it is important to express constraints like 'in the document, two nodes do not have the same key'. Unfortunately, tree automata are in general not expressive enough to express this kind of constraints. To this end some extensions of tree automata have been proposed, let us quote hedge tree automata [10], visibly tree automata with memory and constraints [11], rigid tree automata [12], tree automata with global equality and disequality constraints (TAGEDs for short) [13]. Although these recent works provide theoretical results promising w.r.t. practical applications, they are not supported by efficient verification and validation tools.

This paper investigates the use of two SAT solvers — `MinisAT2` and `picoSAT` — to evaluate queries over TAGEDs. Unlike general tree automata, TAGEDs allow expressing constraints useful e.g., for evaluating queries over XML documents. TAGEDs are also useful for security protocol verification [12], and for LTL model-checking of infinite states

systems [5]. When processing and analysing XML documents, queries exploit the membership problem for TAGEDs. This problem being NP-complete for TAGEDs, we propose an efficient SAT-based encoding of the membership problem for TAGEDs, and we show its correctness and soundness. The challenge is to evaluate queries over tree automata when considering both equality and disequality constraints. We present some experimental results showing the interest of such an approach.

Layout of the paper. The paper is organised as follows. After presenting a motivating example in Section II, preliminary notions on terms and tree-automata are given in Sect. III. Section IV presents the main contribution – the SAT-based encoding of the membership problem for TAGEDs, and states its correctness and soundness. Section V reports on experimental results showing the feasibility and the interest of the proposed approach, and gives implementation details. Finally, Section VI concludes and presents related work.

II. MOTIVATING EXAMPLE

This section illustrates on a toy example how TAGEDs can be used to analyse the structure of XML files.

Let us consider an XML document describing the members of a University. In this context, a university is viewed as a list of research teams. Each team is affiliated to a research laboratory and is composed of researchers, called members. To simplify the tree automaton encoding, we assume that in the university there are two teams, at least, and that there are two members in each team, at least. Therefore, the corresponding XML documents are to be conformed to the following DTD.

```
<!ELEMENT university (team,team+)>
<!ELEMENT team (member,member+,laboratory)>
<!ELEMENT laboratory (#PCDATA)>
<!ELEMENT member (#PCDATA)>
```

Such XML documents are classically encoded by terms: the binary symbol f_{team} encodes the list of teams, and the binary symbol g_{team} is a constructor for each team: Its first argument is the list of team members and its second argument is the laboratory managing the team. The list of members is encoded using the binary symbol f_{memb} . Finally, each letter of the alphabet is a unary symbol encoding itself. There is a unique constant symbol \perp . For instance, the following

XML file represents two teams of the Computer Science (CS) Laboratory: the first team has 3 members whose names are JS, JD and RKT, and the second team consists of two members, called WH and JFL.

For instance the following XML document

```
<university>
<team>
<member> JS </member>
<member> JD </member>
<member> RKT </member>
<laboratory> CS </laboratory>
</team>
<team>
<member> WH</member>
<member> JFL </member>
<laboratory> CS </laboratory>
</team>
</university>
```

is encoded by the term $fteam(t1, t2)$, where $t1 = gteam(h1, h2)$ and $t2 = gteam(h3, h4)$ with $h1 = fmemb(J(S(\perp)), fmemb(J(D(\perp)), R(K(T(\perp))))$, $h2 = C(S(\perp))$, $h3 = fmemb(W(H(\perp)), J(F(L(\perp)))$ and $h4 = C(S(\perp))$. Note that this term is depicted in Section VII.

Now, assuming that XML files result from a request providing all teams of a given laboratory, and that a researcher cannot be affiliated to two different teams, the specifier wants to check whether the given document satisfies the DTD and these two constraints. Such verifications can be done using an extended kind of tree automata, called TAGEDs and introduced in the next section. The TAGED corresponding to the example is provided in Section VII.

III. PRELIMINARIES

Comprehensive surveys can be found in [14], [15] for tree automata and tree language theory, and in [16] for TAGEDs.

a) *Terms*: Let Σ be a finite set of symbols, associated with an arity function $ar : \Sigma \rightarrow \mathbb{N}$ and let $\mathcal{T}(\Sigma)$ denote the set of terms. A position p for a term t is a word over \mathbb{N} . The empty sequence ϵ denotes the top-most position. The set $Pos(t)$ of positions of a term t is inductively defined by $Pos(t) = \{\epsilon\}$ if $t \in \mathcal{X}$ and by $Pos(f(t_1, \dots, t_n)) = \{\epsilon\} \cup \{i.p \mid 1 \leq i \leq n \text{ and } p \in Pos(t_i)\}$ otherwise. If $p \in Pos(t)$, then $t|_p$ denotes the subterm of t at position p . We also denote by $t(p)$ the symbol occurring in t at position p .

b) *Tree automata*: Let Q be a finite set of symbols, of arity 0, called *states* such that $Q \cap \Sigma = \emptyset$. $\mathcal{T}(\Sigma \cup Q)$ is called the set of *configurations*. A *transition* is a rewrite rule $c \rightarrow q$, where $c \in \mathcal{T}(\Sigma \cup Q)$ is of the form $c = f(q_1, \dots, q_n)$, $f \in \Sigma$, $ar(f) = n$, and $q_1, \dots, q_n \in Q$.

A *bottom-up non-deterministic finite tree automaton* (tree automaton for short) over Σ is a tuple $\mathcal{A} = (\Sigma, Q, F, \Delta)$, $F \subseteq Q$ and Δ is a finite set of transitions. The rewriting relation on $\mathcal{T}(\Sigma \cup Q)$ induced by Δ of \mathcal{A} is denoted \rightarrow_{Δ} or $\rightarrow_{\mathcal{A}}$. The tree language $\{t \in \mathcal{T}(\Sigma) \mid t \rightarrow_{\mathcal{A}}^* q\}$ is denoted $L(\mathcal{A}, q)$ and called the *tree language recognised by \mathcal{A} in q* . The language recognised by \mathcal{A} , denoted $L(\mathcal{A})$, is the language $\bigcup_{q \in F} L(\mathcal{A}, q)$. A *run* of a tree automaton $\mathcal{A} = (\Sigma, Q, F, \Delta)$

on a term $t \in \mathcal{T}(\Sigma)$ is a function $\rho : Pos(t) \rightarrow Q$ such that $\rho(p) = q$ for all $p \in Pos(t)$, where $q \in Q$ and $t|_p = f(t_1, \dots, t_n)$, $ar(f) = n$, $f(\rho(p.1), \dots, \rho(p.n)) \rightarrow q \in \Delta$. A run is *successful* if $\rho(\epsilon) \in F$.

c) *Positive TAGEDs*: A TAGED [16] is a tuple $\mathcal{A} = (\Sigma, \Delta, Q, F, =_{\mathcal{A}}, \neq_{\mathcal{A}})$, where (Σ, Q, F, Δ) is a tree automaton over Σ , $=_{\mathcal{A}} \subseteq Q \times Q$ is a binary reflexive symmetric relation on a subset of Q and $\neq_{\mathcal{A}} \subseteq Q \times Q$ is a symmetric relation on Q ⁽¹⁾. The tree automaton (Q, F, Δ) is denoted $ta(\mathcal{A})$. A successful run of a TAGED $\mathcal{A} = (\Sigma, \Delta, Q, F, =_{\mathcal{A}}, \neq_{\mathcal{A}})$ on a term $t \in \mathcal{T}(\Sigma)$ is a successful run ρ of $ta(\mathcal{A})$ on t satisfying: for all positions $p_1, p_2 \in Pos(t)$, (1) if $(\rho(p_1), \rho(p_2)) \in =_{\mathcal{A}}$ then $t|_{p_1} = t|_{p_2}$, and (2) if $(\rho(p_1), \rho(p_2)) \in \neq_{\mathcal{A}}$ then $t|_{p_1} \neq t|_{p_2}$.

For TAGEDs, the membership problem is NP-complete [16]. Emptiness is known to be decidable for restrictive cases, whereas universality is undecidable [16, Proposition 5]. Following the respective definitions of runs, it is straightforward that for every positive TAGED \mathcal{A} , $L(\mathcal{A}) \subseteq L(ta(\mathcal{A}))$. Examples of TAGEDs and of a successful run are given in Section VII.

d) *Boolean formulas*: Let φ be a boolean formula over a set A of atomic propositions. An interpretation of variables is a function I from A into $\{\text{True}, \text{False}\}$. An interpretation I satisfies the formula φ , denoted $I \models \varphi$, if φ is true for the interpretation I of the variables.

IV. ENCODING

This section presents our propositional encoding of the membership problem, and we informally justify it step by step. We shall also illustrate our sub-formulae as we go along by instantiating them on a small example. For this purpose we will use the following TAGED \mathcal{A} and term t :

$$\mathcal{A} \stackrel{\text{def}}{=} (\Sigma = \{a, f\}, Q = \{q, \hat{q}, q_f\}, F = \{q_f\},$$

$$\Delta, \hat{q} \stackrel{\text{def}}{=} \hat{q}, \hat{q} \neq_{\mathcal{A}} q_f),$$

$$\text{where } \Delta \stackrel{\text{def}}{=} \{f(\hat{q}, \hat{q}) \rightarrow q_f, f(q, q) \rightarrow q, f(q, q) \rightarrow \hat{q},$$

$$a \rightarrow q, a \rightarrow \hat{q}\},$$

$$t \stackrel{\text{def}}{=} f_{\epsilon}^{t2} \begin{cases} f_1^{t1} \begin{cases} a_{11}^{t0} \\ a_{12}^{t0} \end{cases} \\ f_2^{t1} \begin{cases} a_{21}^{t0} \\ a_{22}^{t0} \end{cases} \end{cases}$$

This small TAGED accepts $\{f(t, t) \mid t \in \mathcal{T}(\mathcal{F})\}$, which is a classical non-regular language. Here $\neq_{\mathcal{A}}$ is redundant and used purely for illustrative purposes. In the term, subscripts are positions and superscripts are unique references to the structure of subterms. For instance $t1$ corresponds to $f(a, a)$, which appears at positions 1 and 2.

Let us enumerate the conditions which must be satisfied in order for our term t to be accepted by \mathcal{A} through a run ρ , and break them down in sub-conditions until we can encode them.

⁽¹⁾Notice that in [16], this relation is supposed to be irreflexive. In this paper, it is not required.

1. The run ρ is a *successful* run for the underlying tree automaton $\mathcal{A}' = (\Sigma, \Delta, Q, F)$.

(a) The run ρ is a function mapping positions of t to states of \mathcal{A} .

i. $\rho \subseteq \text{Pos}(t) \times Q$

ii. $\forall \alpha \in \text{Pos}(t), p \neq q \in Q, (\alpha, p) \in \rho \implies (\alpha, q) \notin \rho$

iii. $\forall \alpha \in \text{Pos}(t), \exists q \in Q, (\alpha, q) \in \rho$

(b) The run ρ must be compatible with the transition rules of Δ .

(c) The run ρ must be *accepting*, ie. $\rho(\varepsilon) \in F$.

2. It must satisfy the global equality constraints in $\models_{\mathcal{A}}$.

3. It must satisfy the global disequality constraints in $\not\models_{\mathcal{A}}$.

Condition (1a)i guides the choice of the building blocks of our formula: they will be variables of the form, say, X_q^α , which will have the intuitive meaning that at a position $\alpha \in \text{Pos}(t)$, we end up in the state $q \in Q$. This corresponds to the statement “ ρ exists and $\rho(\alpha) = q$ ”. Let us now encode, using the above variables, the fact that ρ is a partial function (1a)ii, that is to say, given $\alpha \in \text{Pos}(t)$ and $p \neq q \in Q$, we cannot have X_p^α and X_q^α at the same time:

Definition IV-1 (Partial function constraint Θ_{\rightarrow}).

$$\Theta_{\rightarrow} \stackrel{\text{def}}{=} \bigwedge_{\substack{\alpha \in \text{Pos}(t) \\ q \in Q}} \left[X_q^\alpha \implies \bigwedge_{\substack{p \in Q \\ p \neq q}} \neg X_p^\alpha \right]$$

Applied to our minimalist example this yields $\{X_q^\varepsilon \Rightarrow [\neg X_q^\varepsilon \wedge \neg X_{qf}^\varepsilon]\} \wedge \{X_q^\varepsilon \Rightarrow [\neg X_q^\varepsilon \wedge \neg X_{qf}^\varepsilon]\} \wedge \dots \wedge \{X_{qf}^{22} \Rightarrow [\neg X_q^{22} \wedge \neg X_{qf}^{22}]\}$. We also need ρ to be compatible with the transition rules of \mathcal{A}' (1b). Let us translate the fact that a transition rule applies at a given position α by:

Definition IV-2 (Rule application constraint $\Psi^\alpha(r)$). We define, for any $\alpha \in \text{Pos}(t)$, and any transition rule $f(q_1, \dots, q_n) \rightarrow q \in \Delta$,

$$\Psi^\alpha(f(q_1, \dots, q_n) \rightarrow q) \stackrel{\text{def}}{=} X_q^\alpha \wedge \bigwedge_{k=1}^n X_{q_k}^{\alpha.k}$$

This is fairly straightforward: we are stating that the rule $f(q_1, \dots, q_n) \rightarrow q \in \Delta$ applies at position α . Therefore we have “ $\rho(\alpha) = q$ ” as a result of the application of the rule, and the k^{th} direct subterm is accepted by the state q_k , as the transition rule requires. Now, in order to express the notion of compatibility with the transition rules, we assert that, at each position in the term, a transition rule applies.

Definition IV-3. For any $f \in \Sigma$, we denote by $\Delta_f = \{f(\dots) \rightarrow \dots \in \Delta\}$ the set of transition rules which apply to f .

Definition IV-4 (Rules compatibility constraint $\Phi^\varepsilon(t)$).

$$\Phi^\varepsilon(t) = \bigwedge_{\alpha \in \text{Pos}(t)} \left[\bigvee_{r \in \Delta_{t(\alpha)}} \Psi^\alpha(r) \right].$$

For instance, on our small example this would be $([X_{qf}^\varepsilon \wedge X_q^1 \wedge X_q^2] \vee [X_q^\varepsilon \wedge X_q^1 \wedge X_q^2] \vee [X_q^\varepsilon \wedge X_q^1 \wedge X_q^2]) \wedge \dots \wedge (X_q^{22} \vee X_{qf}^{22})$. Note that if $\Phi^\varepsilon(t)$ satisfies (1b), then clearly ρ must be a total function (1a)iii, since at every position $\alpha \in \text{Pos}(t)$, we must be in some state q resulting from the application of some transition rule. Note also that if both Θ_{\rightarrow} and $\Phi^\varepsilon(t)$ are satisfied simultaneously, then exactly one rule applies at each position. The last thing we need to encode an accepting run for a tree automaton, is to specify that the run must end up in a final state at the root of the term (1c); this is directly translated into $\bigvee_{q \in F} X_q^\varepsilon$. Now we must add further restrictions to ensure compatibility with the global equality and disequality constraints (2 and 3). The variables we have already defined are not sufficient to translate statements of the form “such subtree does (or does not) evaluate to such state”; therefore we need to introduce new variables to link states and subterms by a relation. Let us use T_u^q to denote “the subterm u evaluates to q ”, for any $u \leq t$ and $q \in Q$. Of course, we need to “glue” these new variables to the old ones: if we are in a certain state q at a position α , then it follows that the subterm $t|_\alpha$ evaluates to q : this is straightforwardly translated into the next formula.

Definition IV-5 (Structural glue: Θ_{\Leftarrow}).

$$\Theta_{\Leftarrow} \stackrel{\text{def}}{=} \bigwedge_{\substack{\alpha \in \text{Pos}(t) \\ q \in Q}} \left[X_q^\alpha \implies T_{t|_\alpha}^q \right].$$

On our example, we have: $\{X_q^\varepsilon \Rightarrow T_2^q\} \wedge \{X_q^\varepsilon \Rightarrow T_2^q\} \wedge \{X_{qf}^\varepsilon \Rightarrow T_2^{qf}\} \wedge \dots \wedge \{X_{qf}^{22} \Rightarrow T_0^{qf}\}$, where the subscript “2” of T_2^q designates the subtree $f(f(a, a), f(a, a))$, as given in the definition of t . Now different kinds of variables being linked, let us encode the equality constraint. Supposing again that $\rho(\alpha) = q$, for the run to be compatible with the equality constraint, it must be such that no subterm different from $t|_\alpha$ can evaluate to p , where $p \neq_A q$. Note that \neq_A is reflexive by definition, so this includes q itself.

Definition IV-6 (Compatibility with \neq_A : Θ_{\neq_A}).

$$\Theta_{\neq_A} \stackrel{\text{def}}{=} \bigwedge_{\substack{\alpha \in \text{Pos}(t) \\ q \in Q}} \left[X_q^\alpha \implies \bigwedge_{\substack{p \in Q \\ p \neq_A q}} \bigwedge_{\substack{u \leq t \\ u \neq t|_\alpha}} \neg T_u^p \right]$$

For instance: $\{X_q^\varepsilon \Rightarrow [\neg T_1^q \wedge \neg T_0^q]\} \wedge \{X_q^{11} \Rightarrow [\neg T_2^q \wedge \neg T_1^q]\} \wedge \dots \wedge \{X_{qf}^{22} \Rightarrow [\neg T_2^q \wedge \neg T_1^q]\}$. There remains to encode the compatibility with the disequality constraint. Let us deal with the case where either \neq_A is assumed to be irreflexive (as in [16]), or the states involved are different. Suppose that we are at position α , and that $\rho(\alpha) = q$; then we cannot have any subterm $t|_\alpha$ evaluate to any p , when $p \neq_A q$.

Definition IV-7 (Compatibility with \neq_A ($p \neq q$): Θ_{\neq_A}).

$$\Theta_{\neq_A} \stackrel{\text{def}}{=} \bigwedge_{\substack{\alpha \in \text{Pos}(t) \\ q \in Q}} \left[X_q^\alpha \implies \bigwedge_{\substack{p \in Q \\ p \neq_A q \\ p \neq q}} \neg T_{t|_\alpha}^p \right]$$

For instance: $\{X_q^\varepsilon \Rightarrow \neg T_2^{qf}\} \wedge \{X_{qf}^\varepsilon \Rightarrow \neg T_2^q\} \wedge \dots \wedge \{X_{qf}^{22} \Rightarrow \neg T_0^q\}$. However, for the needs of our test examples, we chose to alter the definition of \neq_A by removing its irreflexivity. The idea is to be able to write statements such as $p \neq_A p$, with the meaning that no two *distinct* subtrees which evaluate to p may be structurally identical. Formally, ρ satisfies \neq_A iff $\forall \alpha, \beta \in \text{Pos}(t), (\alpha \neq \beta \wedge \rho(\alpha) \neq_A \rho(\beta)) \implies t|_\alpha \neq t|_\beta$. This cannot be done solely in Θ_{\neq_A} , because the formula will not differentiate between two distinct subterms and the same subterm, taken twice, which is why the case where $q \neq_A q$ must be dealt with separately. Indeed, as we do not yet have any means for linking subterms with positions, a new kind of variables is needed, of the form S_u^α , which encodes the statement “the subterm u is rooted in α ”. The above property is then encoded using this variable, as follows:

Definition IV-8 (Compatibility with \neq_A (non-irreflexive; $q \neq_A q$): Ω_{\neq_A}).

$$\Omega_{\neq_A} \stackrel{\text{def}}{=} \bigwedge_{\alpha \in \text{Pos}(t)} S_{t|_\alpha}^\alpha \wedge \bigwedge_{\substack{\alpha \neq \beta \in \text{Pos}(t) \\ q \neq_A q}} [X_q^\alpha \wedge X_q^\beta \implies \neg S_{t|_\beta}^\alpha]$$

We can now state our main result:

Definition IV-9 (SAT encoding of TAGED membership problem $\Delta_A(t)$). Let $\mathcal{A} = (\Sigma, \Delta, Q, F, \Rightarrow_A, \neq_A)$ be a TAGED and $t \in \mathcal{T}(\Sigma)$; then we define

$$\Delta_A(t) \stackrel{\text{def}}{=} \Theta_{\rightarrow} \wedge \Phi^\varepsilon(t) \wedge \bigvee_{q \in F} X_q^\varepsilon \wedge \Theta_{\Rightarrow_A} \wedge \Theta_{\neq_A} \wedge \Omega_{\neq_A}.$$

Theorem IV-1 (TAGED membership, correctness and soundness). *There exists a successful run ρ of the TAGED \mathcal{A} on a term t iff $\Delta_A(t)$ is satisfiable. Moreover, if $I \models \Delta_A(t)$, then for any $\alpha \in \text{Pos}(t)$ we have $\rho(\alpha) = q \iff I \models X_q^\alpha$.*

The above encoding has been simplified, implemented and tested. This is the matter of the next section.

V. IMPLEMENTATION AND EXPERIMENTATIONS

In the first part of this section we will quickly go over some ways in which the formula can be lightened through simple observations, before discussing some of our experimentations in the second part.

The above SAT encoding, though sizeable, remains polynomial in the size of our input automaton \mathcal{A} and the term t : the size of $\Delta_A(t)$ (as number of literals) is a $O(|t|^2 |Q|^2)$. In practice however, this can often be trimmed down considerably. Let ρ be a successful run of the underlying tree automaton \mathcal{A} on t , and consider for instance the structural glue: $\Theta_{\Leftarrow} = \bigwedge_{\alpha \in N_t, q \in Q} [X_q^\alpha \implies T_{t|_\alpha}^q]$. The formula considers all possible couples (α, q) , but in general this is unnecessary because not all states are obtainable at any given position. In order to ever have X_q^α , that is to say, $\rho(\alpha) = q$, there must be some transition rule of the form $t(\alpha)(\dots) \rightarrow q$ in Δ , at least. Thus we let $\sigma(\alpha)$ be the set of *possibly obtainable states at position α* : $\sigma(\alpha) \stackrel{\text{def}}{=} \{q \in Q / \exists t(\alpha)(\dots) \rightarrow q \in \Delta\}$ and, given a position α , we only need to deal with $q \in \sigma(\alpha)$. Another observation which can be made *a priori* is that the only occurrences

of negations of the form $\neg T_u^q$ occur in Θ_{\Rightarrow_A} and Θ_{\neq_A} , when q is in the domain of either \neq_A or \Rightarrow_A . It follows that literals of the form T_u^q can only alter the satisfiability of $\Delta_A(t)$ when q is in $\text{dom}(\neq_A) \cup \text{dom}(\Rightarrow_A)$. Thus we can reduce the formula to $\Theta_{\Leftarrow} = \bigwedge_{\alpha \in N_t, q \in \sigma(\alpha) \cap (\text{dom}(\neq_A) \cup \text{dom}(\Rightarrow_A))} [X_q^\alpha \implies T_{t|_\alpha}^q]$. The same observations can be made in Θ_{\neq_A} , Ω_{\neq_A} and Θ_{\Rightarrow_A} . In the case of Θ_{\Rightarrow_A} , we can also argue that in the subformula $\bigwedge_{u \leq t, u \neq t|_\alpha} \neg T_u^p$ it is unnecessary to write $\neg T_u^p$ when we know that the subtree u cannot possibly evaluate to the state p . This is clearly the case if the root symbol $u(\varepsilon)$ is not used in any transition rule leading to p . Thus we let $\tau(q) \stackrel{\text{def}}{=} \{f \in \Sigma \mid \exists f(\dots) \rightarrow q \in \Delta\}$ be the set of symbols which a subterm may be rooted in, given that it evaluates to the state q , and we lighten the above subformula into $\bigwedge_{u \leq t, u \neq t|_\alpha, u(\varepsilon) \in \tau(p)} \neg T_u^p$. Lastly, in the revised formula Ω_{\neq_A} , it is clear that the variables $S_{t|_\alpha}^\alpha$ serve no purpose whatsoever when the subtree in α cannot evaluate to a state q such that $q \neq_A q$. Thus we let $\mu(q) \stackrel{\text{def}}{=} \{\alpha \in \text{Pos}(t) \mid t(\alpha) \in \tau(q)\}$ be the set of positions at which the subtree may evaluate to the state q , and reduce the first part of the subformula to $\bigwedge_{\alpha \in \bigcup_{q \neq_A q} \mu(q)} S_{t|_\alpha}^\alpha$. In its second part, we arbitrarily order positions and re-group couples of implications with the same premises: $\bigwedge_{\alpha < \beta \in \mu(q), q \neq_A q} [X_q^\alpha \wedge X_q^\beta \implies \neg S_{t|_\beta}^\alpha \wedge \neg S_{t|_\alpha}^\beta]$. Note that reducing Θ_{\rightarrow} is much more problematic, but it is possible to simply do away with this part of the formula altogether if one replaces $\bigvee_{q \in F} X_q^\varepsilon$ by $\bigwedge_{q \in F} \neg X_q^\varepsilon$, provided that the term is accepted by the underlying tree automaton. This can be checked separately by other, less expensive means, since the membership problem for tree automata is polynomial. Of course in that case the second result of theorem IV-1 does not apply anymore. While computationally inexpensive, these simplifications can yield significant savings on TAGEDs with low density and where few states are involved in the global constraints, which are fairly reasonable assumptions in the context of XML documents processing. Note that one could find more drastic simplifications by examining the tree automaton more closely; for instance one could remove, at each position, any state which cannot appear in a successful run. Simplifications of this kind would certainly yield better results on sizeable and complex TAGEDs, but it is not certain that the overhead of implementing and computing them would be compensated by the SAT solving performance gains. For our tests we implemented the static simplifications described above, which divided the size of the generated formula by 36 in the case of our Laboratory example automaton.

In order to test our encoding, we have been developing a tool which takes as input a TAGED (in a syntax close to that of Timbuk [17]) and a term, and generates the corresponding formula $\Delta_A(t)$. However, most modern SAT solvers take input in the DIMACS CNF format, and naive conversion to Conjunctive Normal Form could lead to an explosion of the size of the formula. In order to avoid running into this problem we used an existing tool to handle linear-size conversion to CNF and generation of DIMACS

CNF files: the BAT⁽²⁾ [18], which implements an efficient CNF conversion algorithm [19]. Experiments were run on an 2.53GHz Intel Core2 Duo machine with 2Gb of RAM running Linux. Figure 1 shows the respective running times of the two SAT solvers `picoSAT` and `MiniSAT2` on an implementation of our Laboratory example. Accepted trees of varying sizes have been generated with random members names of random length. In the figure the size of the generated trees is given in terms of the number of teams in the university; the size in terms of the number of nodes is proportional to these data. The test shows that while both solvers perform very well on

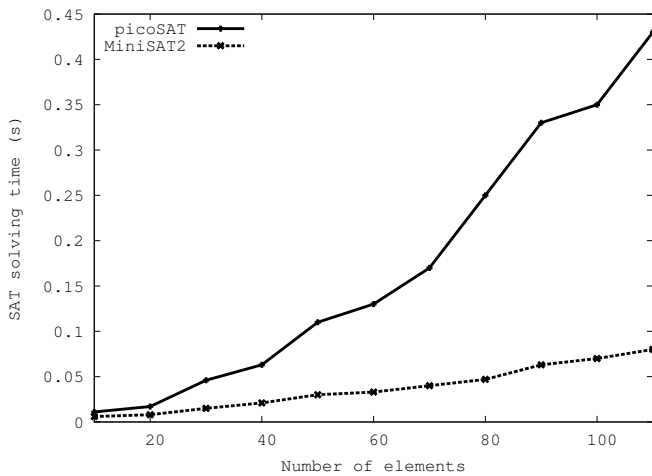


Fig. 1: CNF solving time, Laboratory example

this query, `MiniSAT2` tends to outperform `picoSAT` as the terms grow, which suggests that the heuristic used for SAT solving may significantly impact the overall efficiency of our queries. Figure 2 shows the same experiment, this time with

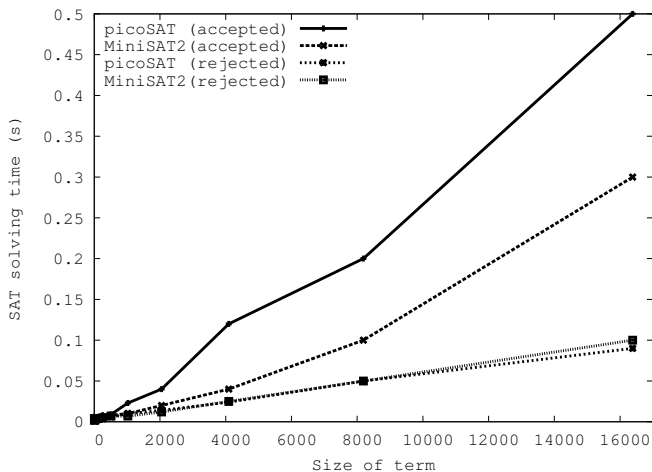


Fig. 2: CNF solving time, $\{ f(t,t) / t \in T(\mathcal{F}) \}$

the small TAGED accepting $\{ f(t,t) / t \in T(\mathcal{F}) \}$ (introduced at the beginning of section IV), and for both accepted and

⁽²⁾Bit-level Analysis Tool, version 0.2

rejected terms. The size of the terms designates the number of nodes of the tree. Both solvers display similar performances for this experiment, with `MiniSAT2` being about twice as fast as `picoSAT` on accepted terms. On rejected terms however both solvers show roughly the same performances, and take less time than on accepted terms, by a factor of 3 (`picoSAT`) and 5 (`MiniSAT2`) on large terms.

It would have been interesting to increase the size of our terms until both solvers timed out, but we were unfortunately limited by the software we used. Our own tool is not optimised for speed, and CNF conversion with BAT took about 4.5 times as much time as formula generation. Moreover, BAT fails with a stack overflow when the input formula becomes too large. Despite these practical setbacks, the results remain fairly encouraging, as the current bottleneck lies on the least computationally expensive parts of the process: both the generation of the formula and the conversion to CNF are quadratic in the worst case. On the other hand, SAT solving proves quite efficient, even on fairly large formulae: the order of magnitude of the largest tested formulae is of approximately 70'000 variables, 120'000 clauses and 250'000 literals (in CNF), for a solving time well under one second.

VI. CONCLUSION AND RELATED WORK

This paper proposes the encoding of the membership problem for a class of extended tree automata, called TAGEDs, in a SAT formula. Therefore, the paper shows that using SAT-solvers allows to successfully handle this NP-complete decision problem useful for practical applications in security protocol verification, Java byte code program analysis, and in XML document processing. The paper also proposes several heuristics to reduce the size of generated formulas, and reports on the experimental results for some of them. The work continues on the implementation of proposed heuristics. We also intend to go further by exploiting the proposed SAT-based approach for larger XML documents.

Related Work. Using SAT-solvers for verification purposes was introduced in [20]: practical experience shows that many bugs in programs occur on small length executions. The idea is then to prove the correctness of a system for bounded executions. In this context, verification problems are frequently NP-complete and can be solved using SAT-solvers. For instance, this approach was successfully used in [21] for hardware verification, in [22] for program analysis, in [23], [24] for security protocol verification, in [25] for LTL model-checking, etc.

Tree automata were intensively studied in the literature, notably for program verification, where they provide abstraction-based approximations of program configurations. In this direction, several classes of extended automata were defined in order to have finer approximations. In particular, it appears that comparing subtrees is a crucial modelling issue that leads to the definition of several classes of tree automata with constraints. In [26], [27], the authors studied a class of automata that allows comparing subterms during runs. Unfortunately, in this framework, the emptiness problem is

undecidable. However, several subclasses with the decidable emptiness problem were pointed out (see [14] for more detail). In order to verify security protocols, a class of tree automata with memory was introduced in [28]. For a similar class of applications, [12] introduces the class of *rigid automata*, which is a subclass of the TAGEDclass [16]. In order to model data base applications, tree automata with Presburger constraints were used in [29]. In this direction, the recent work [30] investigates unranked tree automata with equality and disequality constraints. Several works have also been done on tree automata with equality modulo an equational theory: [31] focuses on associative-commutative theories, while [32] tackles more general cases.

In [8], tree automata and the rewriting theory have been used for verifying XML updates. Basically, in the context of XML programming, types can be viewed as hedge tree automata [10]. Then, given a set of update operations modelled by rewrite rules, rewriting over languages recognised by those automata is used to ensure that XML document types are preserved along any sequence of updates. Like in our approach, the results exploit the decidability of the membership problem and of the emptiness problem for the considered classes of tree automata.

REFERENCES

- [1] Y. Boichut, P.-C. Héam, and O. Kouchnarenko, "Approximation-based tree regular model-checking," *Nordic Journal of Computing*, vol. 14, pp. 194–219, 2008.
- [2] Y. Boichut, R. Courbis, P.-C. Héam, and O. Kouchnarenko, "Finer is better: Abstraction refinement for rewriting approximations," in *Rewriting Techniques and Application, RTA'08*, ser. Lecture Notes in Computer Science, vol. 5117. Springer, 2008, pp. 48–62.
- [3] Y. Boichut, P.-C. Héam, and O. Kouchnarenko, "Tree automata for detecting attacks on protocols with algebraic cryptographic primitives," *ENTCS*, vol. 239, pp. 57–72, 2009, infinity 2006, 2007, 2008 Best papers.
- [4] Y. Boichut, T. Genet, T. P. Jensen, and L. L. Roux, "Rewriting approximations for fast prototyping of static analyzers," in *RTA*, ser. Lecture Notes in Computer Science, F. Baader, Ed., vol. 4533. Springer, 2007, pp. 48–62.
- [5] R. Courbis, P.-C. Héam, and O. Kouchnarenko, "Taged approximations for temporal properties model-checking," in *CIAA*, ser. Lecture Notes in Computer Science, S. Maneth, Ed., vol. 5642. Springer, 2009, pp. 135–144.
- [6] T. Schwentick, "Automata for xml—a survey," *J. Comput. Syst. Sci.*, vol. 73, no. 3, pp. 289–315, 2007.
- [7] S. Abiteboul, L. Segoufin, and V. Vianu, "Modeling and verifying active xml artifacts," *IEEE Data Eng. Bull.*, vol. 32, no. 3, pp. 10–15, 2009.
- [8] F. Jacquemard and M. Rusinowitch, "Rewrite based verification of xml updates," *CoRR*, vol. abs/0907.5125, 2009.
- [9] M. Bojanczyk, A. Muscholl, T. Schwentick, and L. Segoufin, "Two-variable logic on data trees and xml reasoning," *J. ACM*, vol. 56, no. 3, 2009.
- [10] M. Murata, "Hedge automata: a formal model for XML schemata," 1999.
- [11] H. Comon-Lundh, F. Jacquemard, and N. Perrin, "Visibly tree automata with memory and constraints," *Logical Methods in Computer Science*, vol. 4, no. 2, 2008.
- [12] F. Jacquemard, F. Klay, and C. Vacher, "Rigid tree automata," in *LATA*, ser. Lecture Notes in Computer Science, A. H. Dediu, A.-M. Ionescu, and C. Martín-Vide, Eds., vol. 5457. Springer, 2009, pp. 446–457.
- [13] E. Filiot, J.-M. Talbot, and S. Tison, "Tree automata with global constraints," in *Developments in Language Theory*, 2008, pp. 314–326.
- [14] H. Comon, M. Dauchet, R. Gilleron, F. Jacquemard, D. Lugiez, S. Tison, and M. Tommasi, *Tree Automata Techniques and Applications*, 2002, available at <http://www.grappa.univ-lille3.fr/tata/>.
- [15] R. Gilleron and S. Tison, "Regular tree languages and rewrite systems," *Fundamenta Informatica*, vol. 24, no. 1/2, pp. 157–174, 1995.
- [16] E. Filiot, J.-M. Talbot, and S. Tison, "Tree automata with global constraints," in *Developments in Language Theory*, 2008, pp. 314–326.
- [17] G. Feuillade, T. Genet, and V. V. T. Tong, "Reachability analysis over term rewriting systems," *J. Autom. Reasoning*, vol. 33, no. 3-4, pp. 341–383, 2004.
- [18] P. Manolios, S. Srinivasan, and D. Vroon, "BAT: The bit-level analysis tool," *Lecture Notes in Computer Science*, vol. 4590, p. 303, 2007.
- [19] B. Manolios and D. Vroon, "Faster SAT Solving with Better CNF Generation," *Design, Automation and Test in Europe, DATE*, 2009.
- [20] E. M. Clarke, A. Biere, R. Raimi, and Y. Zhu, "Bounded model checking using satisfiability solving," *Formal Methods in System Design*, vol. 19, no. 1, pp. 7–34, 2001.
- [21] A. Gupta, M. K. Ganai, and C. Wang, "Sat-based verification methods and applications in hardware verification," in *SFM*, ser. Lecture Notes in Computer Science, M. Bernardo and A. Cimatti, Eds., vol. 3965. Springer, 2006, pp. 108–143.
- [22] F. Ivancic, Z. Yang, M. K. Ganai, A. Gupta, and P. Ashar, "Efficient sat-based bounded model checking for software verification," *Theor. Comput. Sci.*, vol. 404, no. 3, pp. 256–274, 2008.
- [23] A. Armando and L. Compagna, "An optimized intruder model for sat-based model-checking of security protocols," in *Electronic Notes in Theoretical Computer Science*, A. Armando and L. Viganò, Eds., vol. 125. Elsevier Science Publishers, March 2005, pp. 91–108, presented to the IJCAR04 Workshop ARSPA, available at <http://www.avispa-project.org>.
- [24] A. Armando, D. A. Basin, Y. Boichut, Y. Chevalier, L. Compagna, J. Cuéllar, P. H. Drielsma, P.-C. Héam, O. Kouchnarenko, J. Mantovani, S. Mödersheim, D. von Oheimb, M. Rusinowitch, J. Santiago, M. Turuani, L. Viganò, and L. Vigneron, "The avispa tool for the automated validation of internet security protocols and applications," in *CAV*, ser. Lecture Notes in Computer Science, K. Etessami and S. K. Rajamani, Eds., vol. 3576. Springer, 2005, pp. 281–285.
- [25] R. Armoni, S. Egorov, R. Fraer, D. Korchemny, and M. Y. Vardi, "Efficient ltl compilation for sat-based model checking," in *ICCAD*. IEEE Computer Society, 2005, pp. 877–884.
- [26] B. Bogaert and S. Tison, "Equality and disequality constraints on direct subterms in tree automata," in *STACS*, 1992, pp. 161–171.
- [27] M. Dauchet, A.-C. Caron, and J.-L. Coquidé, "Automata for reduction properties solving," *J. Symb. Comput.*, vol. 20, no. 2, pp. 215–233, 1995.
- [28] H. Comon and V. Cortier, "Tree automata with one memory set constraints and cryptographic protocols," *Theoretical Computer Science (TCS'05)*, vol. 331, 2005.
- [29] H. Seidl, T. Schwentick, A. Muscholl, and P. Habermehl, "Counting in trees for free," in *ICALP*, 2004, pp. 1136–1149.
- [30] W. Karianto and C. Löding, "Unranked tree automata with sibling equalities and disequalities," in *ICALP*, 2007, pp. 875–887.
- [31] H. Ohsaki and T. Takai, "ACTAS: A system design for associative and commutative tree automata theory," *Electronic Notes in Theoretical Computer Science*, vol. 124, no. 1, pp. 97–111, 2005. [Online]. Available: <http://dx.doi.org/10.1016/j.entcs.2004.07.017>
- [32] F. Jacquemard, M. Rusinowitch, and L. Vigneron, "Tree automata with equality constraints modulo equational theories," in *IJCAR*, 2006, pp. 557–571.

VII. APPENDIX

A. The TAGED \mathcal{A}_{exa}

We present the TAGED used for the example in Section II.

$\mathcal{A}_{\text{exa}} = (\Sigma, \Delta, Q, F, \equiv_A, \neq_A)$ where

- ◊ $\Sigma = \{fteam, fmemb, gteam, A, B, C, \dots, Z, \perp\}$;
 $fteam$, $fmemb$ and $gteam$ are binary symbols,
 A, \dots, Z are unary symbols and \perp is a constant;
- ◊ $Q = \{qChar, qName, qNames, qLab, qTeam, qTeams\}$;
- ◊ $F = \{qTeams\}$;
- ◊ Δ is the following sets of rules. First, all strings can be reduced to state $qChar$.

- $\perp \rightarrow qChar$,
- $A(qChar) \rightarrow qChar$
- $B(qChar) \rightarrow qChar$
- \vdots
- $Z(qChar) \rightarrow qChar$

Second, let $qLab$ be the state for the laboratory identifier.

So, there are the rules below:

- $A(qChar) \rightarrow qLab$
- $B(qChar) \rightarrow qLab$
- \vdots
- $Z(qChar) \rightarrow qLab$

Third, each name of team members reduces to $qName$.

So, one has the following rules:

- $A(qChar) \rightarrow qName$
- $B(qChar) \rightarrow qName$
- \vdots
- $Z(qChar) \rightarrow qName$

Now one can encode the list of team members using the state $qNames$ and the rules

- $fmemb(qName, qName) \rightarrow qNames$
- $fmemb(qName, qNames) \rightarrow qNames$

To finish, one has to encode the list of teams by the following rules:

- $gteam(qNames, qLab) \rightarrow qTeam$
- $fteam(qTeam, qTeam) \rightarrow qTeams$
- $fteam(qTeam, qTeams) \rightarrow qTeams$

At this stage, a term can be reduced to $qTeams$ iff it encodes an XML document satisfying the DTD.

- ◊ It remains to encode that all teams are affiliated to the same laboratory, and that all members are pairwise different. The first constraint is encoded by the \equiv_A relation:

$$\equiv_A = \{(qLab, qLab)\}.$$

The second constraint is encoded by the \neq_A relation:

$$\neq_A = \{(qName, qName)\}.$$

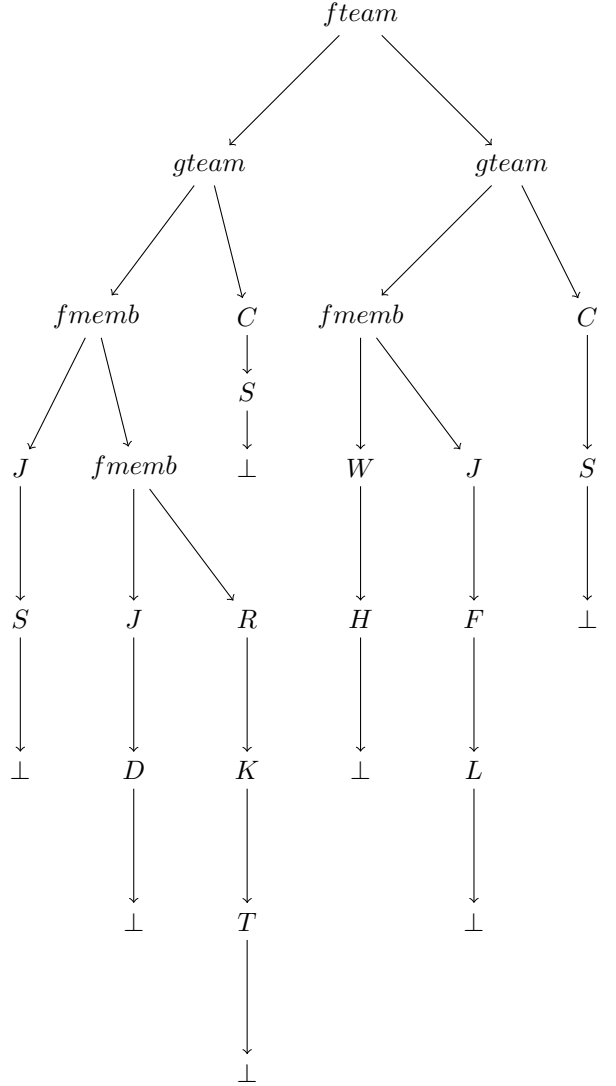
Notice that the TAGED used in Section V is slightly different.

B. Encoding an XML Document

We consider the following XML document depicted in Section II.

```
<university>
  <team>
    <member> JS </member>
    <member> JD </member>
    <member> RKT </member>
    <laboratory> CS </laboratory>
  </team>
  <team>
    <member> WH</member>
    <member> JFL </member>
    <laboratory> CS </laboratory>
  </team>
</university>
```

It is encoded by the following term.



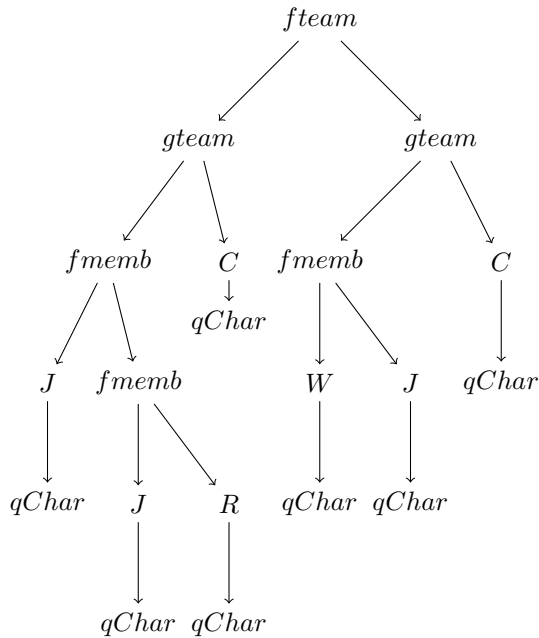
C. A Successful Run

We present a successful run of \mathcal{A}_{exa} on the term above.

Using transitions

- $\perp \rightarrow qChar,$
- $A(qChar) \rightarrow qChar$
- $B(qChar) \rightarrow qChar$
- \vdots
- $Z(qChar) \rightarrow qChar$

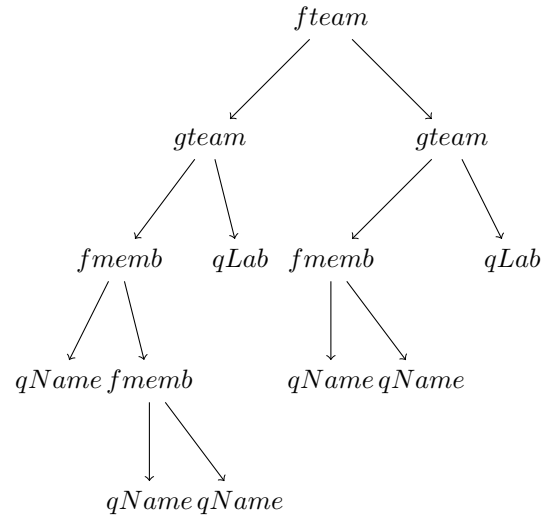
the term can be rewritten into



Now using

- $A(qChar) \rightarrow qName$
- $B(qChar) \rightarrow qName$
- \vdots
- $Z(qChar) \rightarrow qName$
- $A(qChar) \rightarrow qLab$
- $B(qChar) \rightarrow qLab$
- \vdots
- $Z(qChar) \rightarrow qLab$

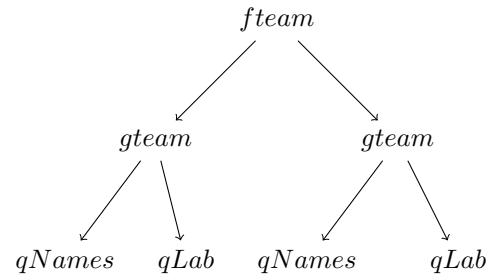
one can obtain



Using the rules

- $fmemb(qName, qName) \rightarrow qNames$
- $fmemb(qName, qNames) \rightarrow qNames$

one obtains



Using the rules

- $gteam(qNames, qLab) \rightarrow qTeam$
- $fteam(qTeam, qTeam) \rightarrow qTeams$

allows us to reduce the term to $qTeams$, which is the final state.

It remains to verify that the constraints imposed by \equiv_A and \neq_A are satisfied. In the considered run, all subterms reducing to $qName$ are distinct; so, the constraint induced by \neq_A is satisfied. Now, all the subterms reducing to $qLab$ in this run are equal to $C(S(\perp))$: the constraint induced by \equiv_A is satisfied, too.