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,okouchnarenko}@lifc.univ-fcomte.fr, vhugot@edu.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-based 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 this recent work provides 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 section 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.

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

Such XML documents are traditionally encoded by terms: the binary symbol $\textit{fteam}$ encodes the list of $\textit{teams}$, and the binary symbol $\textit{gteam}$ is a constructor for each $\textit{team}$. Its first argument is the list of team $\textit{members}$ and its second argument is the $\textit{laboratory}$ managing the team. The list of members is encoded using the binary symbol $\textit{fmember}$. Finally, each letter of the alphabet is a unary symbol encoding itself. There is a unique constant symbol $\bot$. For instance, the following XML file represents two $\textit{teams}$ 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

```xml
<team>
  <member> JS </member>
  <member> JD </member>
  <member> RKT </member>
</team>
```
1. The run $\rho$ is a successful run for the underlying tree automaton $A' = (\Sigma, \Delta, Q, F)$.
   
   (a) The run $\rho$ is a function mapping positions of $t$ to states of $A$.
   
   i. $\rho \subseteq Pos(t) \times Q$
   
   ii. $\forall \alpha \in Pos(t), p \neq q \in Q, (\alpha, p) \in \rho \implies (\alpha, q) \notin \rho$
   
   iii. $\forall \alpha \in Pos(t), \exists q \in Q, (\alpha, q) \in \rho$
   
   (b) The run $\rho$ must be compatible with the transition rules of $\Delta$.
   
   (c) The run $\rho$ must be accepting, i.e. $\rho(\epsilon) \in F$.
   
2. It must satisfy the global equality constraints in $\equiv_A$.
   
3. It must satisfy the global disequality constraints in $\neq_A$.

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

**Definition 1** (Partial function constraint $\Theta_{\not=}$.)

$$
\Theta_{\not=} \overset{\text{def}}{=} \bigwedge_{\alpha \in Pos(t) \atop q \in Q} \left[ X^\alpha_q \implies \bigwedge_{p \in Q \atop p \neq q} \neg X^\alpha_p \right]
$$

Applied to our minimalist example this yields $\{X^f_q \implies [-X_f^e \land \neg X^e_q] \lor [X^e_q \implies [-X^e_f \land \neg X^f_q]] \lor \cdots \lor [X^{22}_q \implies [-X^{22}_f \land \neg X^f_q]]\}$. We also need $\rho$ to be compatible with the transition rules of $A'$ (1(b)). Let us translate the fact that a transition rule applies at a given position $\alpha$ by:

**Definition 2** (Rule application constraint $\Psi^\alpha(r)$). We define, for any $\alpha \in Pos(t)$, and any transition function $\rho(q_1, \ldots, q_n) \rightarrow q \in \Delta$, $\Psi^\alpha(f(q_1, \ldots, q_n) \rightarrow q) \overset{\text{def}}{=} X^\alpha_q \land \bigwedge_{k=1}^n X^\alpha_{q_k}$.

This is fairly straightforward: we are stating that the rule $f(q_1, \ldots, q_n) \rightarrow q \in \Delta$ applies at position $\alpha$. 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 3.** For any $f \in \Sigma$, we denote by $\Delta_f = \{ f(\ldots) \rightarrow \cdots \in \Delta \}$ the set of transition rules which apply to $f$.

**Definition 4** (Rules compatibility constraint $\Phi^e(t)$). $\Phi^e(t) \overset{\text{def}}{=} \bigwedge_{\alpha \in Pos(t) \atop q \in Q} \left[ \bigvee_{r \in \Delta_f(\alpha)} \Psi^\alpha(r) \right]$.

For instance, on our small example this would be $\{[X^e_f \land X^e_q] \lor [X^e_q \land X^e_f] \lor [X^f_q \land X^f_q] \land \cdots \lor [X^{22}_q \land X^{22}_q]\}$.

Note that if $\Phi^e(t)$ satisfies (1b), then clearly $\rho$ must be a total function (1(aiii)), since at every position $\alpha \in Pos(t)$, we must be in some state $q$ resulting from the application of some transition rule. Note also that if both $\Theta_{\not=}$ and $\Phi^e(t)$ 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 (1e); this is directly translated into $\bigvee_{\alpha \in F \atop q \in Q} X^\alpha_q$. 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 $T^\alpha_q$ to denote “the subterm $u$ evaluates to $q$”, for any $u \subseteq t$ 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 $\alpha$, then it follows that the subterm $t|\alpha$ evaluates to $q$; this is straightforwardly translated into the next formula.

**Definition 5** (Structural glue: $\Theta_{\equiv}$). $\Theta_{\equiv} \overset{\text{def}}{=} \bigwedge_{\alpha \in Pos(t) \atop q \in Q} \left[ X^\alpha_q \implies T^\alpha_q \right]$.

On our example, we have: $\{X^e_f \implies T^e_q \} \land \{X^e_q \implies T^e_q \} \land \{X^{22}_q \implies T^{22}_q \} \land \cdots \land \{X^{22}_q \implies T^{22}_q \} \land \cdots \land \{X^{22}_q \implies T^{22}_q \} \land \cdots \land \{X^{22}_q \implies T^{22}_q \}$.

For instance: $\{X^e_f \implies [\neg T^e_2 \land \neg T^e_2] \} \land \{X^e_q \implies [\neg T^e_2 \land \neg T^e_2] \} \land \cdots \land \{X^{22}_q \implies [\neg T^{22}_2 \land \neg T^{22}_2] \} \land \cdots \land \{X^{22}_q \implies [\neg T^{22}_2 \land \neg T^{22}_2] \}$.

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 [13]), or the states involved are different. Suppose that we are at position $\alpha$, and that $\rho(\alpha) = q$; then we cannot have any subterm $t|\alpha$ evaluate to any $p$, when $p \neq_A q$.

**Definition 6** (Compatibility with $\equiv_A$; $\Theta_{\equiv_A}$). $\Theta_{\equiv_A} \overset{\text{def}}{=} \bigwedge_{\alpha \in Pos(t) \atop q \in Q} \left[ X^\alpha_q \implies \bigwedge_{p \in Q \atop p \neq q} \neg T^p_{t|\alpha} \right]$.

For instance: $\{X^e_f \implies [\neg T^e_2 \land \neg T^e_2] \} \land \{X^e_q \implies [\neg T^e_2 \land \neg T^e_2] \} \land \cdots \land \{X^{22}_q \implies [\neg T^{22}_2 \land \neg T^{22}_2] \}$.

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, $\rho$ satisfies $\neq_A$ iff $\forall \alpha, \beta \in \text{Pos}(t), (\alpha \neq \beta \land \rho(\alpha) \neq_A \rho(\beta)) \implies t_{\alpha} \neq t_{\beta}$.

This cannot be done solely in $\Theta_{\neq_A}$, because the formula will not differentiate between two distinct subtrees 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 $S_{\alpha}^{\rho}$, which encodes the statement “the subterm $u$ is rooted in $\alpha$”. The above property is then encoded using this variable, as follows:

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

$$\Omega_{\neq_A} \equiv \bigwedge_{\alpha \in \text{Pos}(t)} S_{\alpha}^{\rho} \land \bigwedge_{\alpha \neq \beta \in \text{Pos}(t)} q \neq_A q \implies \neg S_{\alpha}^{\rho} \land \neg S_{\beta}^{\rho}$$

We can now state our main result:

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

$$\Delta_A(t) \equiv \Theta_\neg \land \Phi^e(t) \land \bigvee_{q \in F} X_q^e \land \Theta_{\neq_A} \land \Theta_{\neq_A} \land \Omega_{\neq_A}.$$  

**Theorem 1** (TAGED membership, correctness and soundness). There exists a successful run $\rho$ of the TAGED $A$ 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 $A$ 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 $\rho$ be a successful run of the underlying tree automaton $A$ on $t$, and consider for instance the structural glue: $\Theta_\neg = \bigwedge_{\alpha \in \text{Pos}(t), q \in Q} X_q^\alpha \implies T_{\alpha q}^d$. The formula considers all possible couples $(\alpha, q)$, but in general this is unnecessary because not all states are obtainable at any given position. In order to ever have $X_q^\alpha$, that is to say, $\rho(\alpha) = q$, there must be some transition rule of the form $t(\alpha)(\ldots) \rightarrow q \in \Delta$, at least. Thus we let $\sigma(q)$ be the set of possibly obtainable states at position $q$: $\sigma(q) \equiv \{ q \in Q/ \exists \alpha(q)(\ldots) \rightarrow q \in \Delta \}$, and, given a position $\alpha$, we only need to deal with $q \in \sigma(q)$.

Another observation which can be made a priori is that the only occurrences of negations of the form $\neg T_{\alpha q}^d$ occur in $\Theta_{\neq_A}$ and $\Theta_{\neq_A}$, when $q$ is in the domain of either $\neq_A$ or $\neq_A$. It follows that literals of the form $T_{\alpha q}^d$ can only alter the satisfiability of $\Delta_A(t)$ when $q$ is in $\text{dom}(\neq_A) \cup \text{dom}(\neq_A)$. Thus we can reduce the formula to

$$\Theta_\neg = \bigwedge_{\alpha \in \text{Pos}(t), q \in Q(\Delta(\neq_A))} X_q^\alpha \implies T_{\alpha q}^d.$$  

The same observations can be made in $\Theta_{\neq_A}$ and $\Theta_{\neq_A}$.

In the case of $\Theta_{\neq_A}$, we can also argue that in the subformula $\bigwedge_{u \in A, u \neq_A \neq_A} \neg T_{\alpha q}^d$ it is unnecessary to write $\neg T_{\alpha q}^d$ when we know that the subtree $u$ cannot possibly evaluate to the state $p$. This is clearly the case if the root symbol $u(e)$ is not used in any transition rule leading to $p$. Thus we have

$$\tau(q) \equiv \{ f \in \Sigma/ \exists f(\ldots) \rightarrow q \in \Delta \}$$

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 $\bigwedge_{u \in A, u \neq_A \neq_A} \neg T_{\alpha q}^d$. Lastly, in the revised formula $\Omega_{\neq_A}$, it is clear that the variables $S_{\alpha}^{\rho}$ serve no purpose whatsoever when the subtree in $\alpha$ cannot evaluate to a state $q$ such that $q \neq_A q$. Thus we let $\mu(q) \equiv \{ \alpha \in \text{Pos}(t) / t(\alpha)(q) \}$ be the set of positions at which the subtree may evaluate to the state $q$, and reduce the first part of the subformula to $\bigwedge_{\alpha \in U(q), u \in \mu(q)} X_q^\alpha \land S_{\alpha}^{\rho} \implies \neg -S_{\alpha}^{\rho} \land -S_{\alpha}^{\rho}$. Note that reducing $\Theta_{\neq_A}$ is much more problematic, but it is possible to simply do away with this part of the formula altogether if one replaces $\bigwedge_{\alpha \in U(q), u \in \mu(q)} X_q^\alpha \land S_{\alpha}^{\rho}$ with the term $\neg X_q^\alpha$ provided that the term is accepted by the underlying automaton $t$. 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 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 [16]) 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 (using De Morgan’s laws, distributivity and removal of double negations) 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) [17], which implements an efficient CNF conversion

\[\text{(2)Bit-level Analysis Tool, version 0.2}\]
algorithm [18]. 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 member 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 size of the term designates the number of nodes 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 formula is of approximately 70'000 variables, 120'000 clauses and 250'000 literals (in CNF), for a solving time well under one second.

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 formula 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 [19]: 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 [20] for hardware verification, in [21] for program analysis, in [22], [23] for security protocol verification, in [24] 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 [25], [26], 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 [27]. For a similar class of applications, [12] introduces the class of rigid automata, which is a subclass of the TAGED class [13]. In order to model database applications, tree automata with Presburger constraints were used in [28]. In this direction, the recent work [29] investigates unranked tree automata with equality and disequality constraints. Several works have also been done on tree automata with equality modulo an equational theory: [30] focuses on associative-commutative theories, while [31] 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


