AN EFFICIENT FINITE TREE AUTOMATA LIBRARY

Ondřej Lengál
Master Degree Programme (2), FIT BUT
E-mail: xlenga00@stud.fit.vutbr.cz

Supervised by: Tomáš Vojnar
E-mail: vojnar@fit.vutbr.cz

ABSTRACT

The structure of numerous computer systems resembles trees. Special techniques need to be employed when performing formal verification of such systems. Some of these techniques use finite tree automata and demand special manipulation. Current libraries for finite tree automata have limitations that make them inappropriate for some of these techniques. This paper proposes a design of a flexible library that allows efficient operations on nondeterministic finite tree automata.

1 INTRODUCTION

Trees play a crucial role in computer science. They recur in many of its fields, from the representation of programs in the form of abstract syntax trees, through the use for fast data retrieval in search trees, to tree topologies of computer networks. It is not surprising that trees are often a natural way to represent a model of many types of systems including safety-critical systems

Software errors in safety-critical systems may cause severe losses of money and, in the worst case, even human lives (the Ariane 5 failure is perhaps the best-known case of an expensive software failure). There are several means which help to avoid software bugs in such systems, one of them being verification based on formal mathematical methods, *formal verification*.

Formal verification of computer systems has gained in popularity in recent years. A popular approach to formal verification is *model checking*, a method based on checking whether a given system conforms to given specification by systematically searching its state space. In the real world, there exist systems with state spaces that are infinite, though they often have regular structure, e.g. queues or stacks. In order to handle such cases, *regular model checking* has been proposed [1]. Configurations of such systems can be represented as finite words over finite alphabet, transitions are represented as relations over words. Then finite (word) automata over the alphabet can be used to represent sets of configurations of the system and finite (word) transducers can be used to express the transition relation.

However, there are also systems that do not have a linear structure, which would enable natural encoding of their configuration into finite words. A special case of these are systems with tree-like structure, such as parameterized tree networks or heaps. For such cases, it is convenient to generalise the method to *regular tree model checking* [2], where finite tree automata,

a generalisation of finite automata to trees, and finite tree transducers are used. Nonetheless when used for reachability analysis, this basic approach may suffer from problems with infinite number of configurations, as the transducers may generate ever new configurations. Therefore various acceleration techniques that ensure finiteness of the method for the majority of real world problems have been proposed. These methods may need to perform sophisticated operations upon finite tree automata (transducers). In order to conduct the operations in verification of non-trivial systems in acceptable time, smart data structures and algorithms must be used. This work attempts to create an implementation of a tree automata library that exploits these optimizations.

2 PRELIMINARIES

A *ranked alphabet* is a couple $(\mathcal{F}, Arity)$ where $\mathcal{F}$ is a finite set of symbols and *Arity* is a mapping $Arity : \mathcal{F} \to \mathbb{N}$ ($\mathbb{N}$ denotes the set of non-negative integer numbers). $Arity(f)$, where $f \in \mathcal{F}$, is the *arity* of f. The set of symbols of arity p is denoted by $\mathcal{F}_p$. *Ground terms* $T(\mathcal{F})$ of an alphabet $\mathcal{F}$ are defined by the following: *(i)* $\mathcal{F}_0 \subseteq T(\mathcal{F})$ and *(ii)* if $p \geq 1$, $f \in \mathcal{F}_p$ and $t_1, \ldots, t_p \in T(\mathcal{F})$, then $f(t_1, \ldots, t_p) \in T(\mathcal{F})$. Ground terms form tree-like structures, therefore they are often used to represent trees.

A (bottom-up) *nondeterministic finite tree automaton* (NFTA) over $\mathcal{F}$ is a 4-tuple $\mathcal{A} = (Q, \mathcal{F}, Q_f, \Delta)$, where Q is a finite set of states ($Q \cap \mathcal{F} = \emptyset$), $Q_f \subseteq Q$ is a set of final states and Δ is a set of transition rules $f(q_1, \ldots, q_n) \to q$, where $n \in \mathbb{N}$, $f \in \mathcal{F}_n$ and $q, q_1, \ldots, q_n \in Q$. The *run* of $\mathcal{A}$ on t is obtained in this way: first, each leaf $a \in \mathcal{F}_0$ is assigned a state $q_a \in Q$ according to the transition rule $a \to q_a \in \Delta$, then for each internal node, if the subterms $t_1, \ldots, t_n$ of the node are labeled with states $q_1, \ldots, q_n$ and if rule $f(q_1, \ldots, q_n) \to q \in \Delta$, then the term $f(t_1, \ldots, t_n)$ can be labelled with q (this is a *move relation* written as $f(t_1, \ldots, t_n) \to_{\mathcal{A}} q$). $\to^*_{\mathcal{A}}$ is the reflexive and transitive closure of $\to_{\mathcal{A}}$. A ground term $t \in T(\mathcal{F})$ is *accepted* by a NFTA $\mathcal{A} = (Q, \mathcal{F}, Q_f, \Delta)$ if there exists $q \in Q_f$ such that $t \to^*_{\mathcal{A}} q$. The language accepted by the automaton $\mathcal{A}$ is the set of ground terms that are accepted, i.e. $\mathcal{L}(\mathcal{A}) = \{t \in T(\mathcal{F}) \mid t \to^*_{\mathcal{A}} q \wedge q \in Q_f\}$.

3 DESIGN

Although there are several packages that provide support for finite tree automata, their implementations are often based on *explicit representation* of the automaton: they work with sets of states and sets of transitions. However, some approaches using *symbolic representation* appear to be very promising. One of these approaches used in MONA [3] (a tool that implements a MTBDD package for binary deterministic finite tree automata) uses representation of transition relation by a *multiple terminal binary decision diagram* (MTBDD) [4] for each pair of states $(q_1, q_2) \in Q^2$ of the automaton. Input symbol f of the automaton is encoded as a binary vector $(x_1, \ldots, x_n)$ which is mapped on respective Boolean variables of the MTBDD: $x_1, \ldots, x_n$. The leaves of the diagram are states $p \in Q$, such that $\Delta(f(q_1, q_2)) = p$.

A similar method was chosen to be used in the implemented library. However, due to the requirements placed on the library, several modifications need to be made. First, to deal with the problem that the library needs to be able to work with symbols of *arbitrary* arity, we propose a solution that instead of a pair of states uses a sequence of states of arbitrary (even 0) length (limited by the maximum arity of a symbol from $\mathcal{F}$) which correspond to *left-hand sides* of transitions in the transition table. Another requested feature is support for nondeterministic

automata, which is implemented by inserting a list of states instead of single states in the leaves of the MTBDD. An example of such mutliple-root MTBDD is in Figure 1. A zero-length sequence of states is denoted by `Empty` label and a *sink* non-accepting state of the automaton by `NULL` (if we add a transition from `NULL` to `NULL` for each symbols, then the automaton is complete).

The use of nondeterminism allows some optimizations. Union of two automata (i.e. construction of an automaton that accepts the language that is the union of languages of two automata) can be constructed very efficiently by taking roots for transitions of leaves and performing an *apply* operation on them such that given two MTBDDs, it concatenates both lists in respective leaves (accessible with the same symbol).

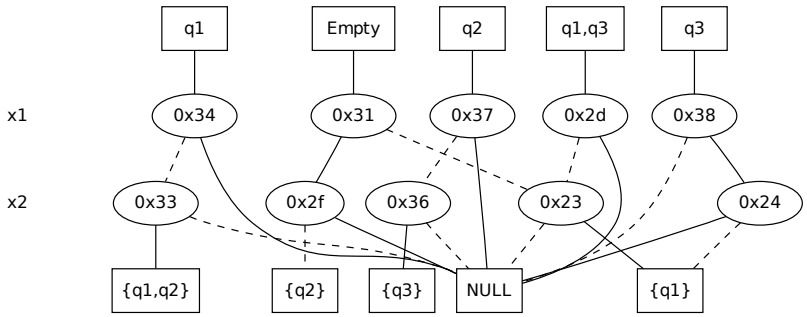

Figure 1: MTBDD representation of a transition relation

4 CONCLUSION

The paper proposes design of a flexible and efficient library that support work with nondeterministic finite tree automata that uses symbolis representation using MTBDDs for transition function.

ACKNOWLEDGEMENT

This work was partially supported by the BUT FIT grant FIT-S-10-1 and the research plan MSM0021630528.

REFERENCES

[1] Ahmed Bouajjani, Bengt Jonsson, Marcus Nilsson, and Tayssir Touili. Regular model checking. In *12th International Conference on Computer Aided Verification*, July 2000. Available at URL: http://www.liafa.jussieu.fr/~abou/rmc-cav00.ps.gz.

[2] Parosh Aziz Abdulla, Bengt Jonsson, Pritha Mahata, and Julien d'Orso. Regular tree model checking. In *14th International Conference on Computer Aided Verification*, 2002. Available at URL: http://user.it.uu.se/~parosh/publications/papers/trees.ps.

[3] MONA: Web pages of MONA. URL: http://www.brics.dk/mona/.

[4] Donald Ervin Knuth. *Combinatorial Algorithms*, volume 4 of *The Art of Computer Programming*, chapter Binary Decision Diagrams. A draft of this text is available at URL: www-cs-faculty.stanford.edu/~knuth/fasc1b.ps.gz.