Information Processing Letters 115 (2015) 418–424

Contents lists available at ScienceDirect

Information Processing Letters

www.elsevier.com/locate/ipl

Symbolic tree automata

Margus Veanes *, Nikolaj Bjørner

Microsoft Research, Redmond, WA, USA

ARTICLE INFO

Article history:
Received 23 August 2011
Received in revised form 31 October 2014
Accepted 15 November 2014
Available online 22 November 2014
Communicated by L. Viganò

Keywords:
Tree automata
Algorithms
Logic
Satisfiability modulo theories
Formal methods

ABSTRACT

We introduce symbolic tree automata as a generalization of finite tree automata with a parametric alphabet over any given background theory. We show that symbolic tree automata are closed under Boolean operations, and that the operations are effectively uniform in the given alphabet theory. This generalizes the corresponding classical properties known for finite tree automata.

1. Introduction

Finite word automata and finite tree automata provide a foundation for a wide range of applications in software engineering, from regular expressions to compiler technology and specification languages. Despite their immense practical use, explicit representations are not feasible in the presence of finite large alphabets. They require each transition to encode only a single element from the alphabet. For example, string characters in standard programming languages (such as the `char` type in C#) use 16-bit bit-vectors, an explicit representation would thus require an alphabet of size 2^{16}. Moreover, most common forms of finite automata do not support infinite alphabets.

A practical solution to the representation problem is *symbolic tree automata*. They are an extension of classical tree automata that addresses this problem by allowing transitions to be labeled with arbitrary formulas in a specified label theory. While the idea of allowing formulas is straightforward, typical extensions of finite tree automata often lead to either undecidability of the emptiness problem, such as tree automata with equality and disequality constraints [1], or many extensions lead to nonclosure under complement, such as the generalized tree set automata class [1], finite-memory tree automata [2] that generalize finite-memory automata [3] to trees, or unranked data tree automata [4]. We show that this is not the case for symbolic tree automata. The key distinction is that the extension here is with respect to *characters* rather than adding symbolic *states* or adding constraints over whole *subtrees*.

The symbolic extension is practically useful for exploiting efficient symbolic constraint solvers when performing basic automata-theoretic transformations: it enables a separation of concerns. The solver is used as a black box with a clearly defined interface that exposes the label theory as an effective Boolean algebra. The chosen label theory can be specific to a particular problem instance. For example, even when the alphabet is finite, e.g., 16-bit bit-vectors, it may be useful for efficiency reasons to use integer-linear arithmetic rather than bit-vector arithmetic when the solver is more efficient over integers and when only standard arithmetic operations (and no bit-level operations) are being used. Recent work [5,6] on symbolic

* Corresponding author.
E-mail addresses: margus@microsoft.com (M. Veanes), nbjorner@microsoft.com (N. Bjørner).
URLs: http://research.microsoft.com/~margus (M. Veanes), http://research.microsoft.com/~nbjorner (N. Bjørner).

http://dx.doi.org/10.1016/j.ipl.2014.11.005
0020-0190/

string recognizers and transducers takes advantage of this observation.

We here investigate the case of the more expressive class of symbolic *tree* automata. Even though a symbolic tree automaton is a finite object, a key point is that the number of interpretations for symbolic labels does not need to be finite. For example, as a consequence of our main result (Theorem 2) a label theory may itself be the theory of symbolic tree automata (over some basic label theory).

In order to use classical tree automata algorithms, it is possible to reduce a symbolic tree automaton A into a classical finite tree automaton whose alphabet is given by all of the satisfiable Boolean combinations of guards that occur in A. However, such a transformation is in general not practical because it introduces an exponential increase in the size of the automaton before the actual algorithm is applied. Moreover, when more than one automaton are involved, this has to be done up front for all predicates that occur in all the automata in order to define the common alphabet. A concrete example of such a blowup is given in [7, Example 2].

2. Definition of symbolic tree automata

We introduce an extension of tree automata with an effective encoding of labels by predicates that denote *sets* of labels, rather than individual labels. We assume a countable *background universe* $\mathscr{B}$. A *predicate* φ over $\mathscr{B}$ is a finite representation of a subset $[\![\varphi]\!]^{\mathscr{B}}$ of $\mathscr{B}$; we write $[\![\varphi]\!]$ when $\mathscr{B}$ is clear from the context. We assume given an effectively enumerable set of predicates Σ such that, for each element $a \in \mathscr{B}$ there is $\hat{a} \in \Sigma$ such that $[\![\hat{a}]\!] = \{a\}$, $\top, \bot \in \Sigma$ such that $[\![\top]\!] = \mathscr{B}$ and $[\![\bot]\!] = \emptyset$, and Σ is effectively closed under Boolean operations: for all $\varphi, \psi \in \Sigma$, we have $\varphi \wedge \psi \in \Sigma$, $\varphi \vee \psi \in \Sigma$, $\neg\varphi \in \Sigma$, where $[\![\varphi \wedge \psi]\!] = [\![\varphi]\!] \cap [\![\psi]\!]$, $[\![\varphi \vee \psi]\!] = [\![\varphi]\!] \cup [\![\psi]\!]$, and $[\![\neg\varphi]\!] = \mathscr{B} \setminus [\![\varphi]\!]$. We write $\varphi \equiv \psi$ for $[\![\varphi]\!] = [\![\psi]\!]$. We say that $(\Sigma, [\![\cdot]\!]^{\mathscr{B}})$ (or Σ, when $[\![\cdot]\!]^{\mathscr{B}}$ is clear from the context) is an *effective Boolean algebra over* $\mathscr{B}$. We say that Σ is *decidable* if the problem of deciding $\varphi \equiv \bot$ for $\varphi \in \Sigma$ is decidable.

Example 1. An example of a decidable effective Boolean algebra is $(LA(x), [\![\cdot]\!]^{\mathbb{Z}})$ where $[\![\cdot]\!]^{\mathbb{Z}}$ is the standard interpretation of integer arithmetic, and $LA(x)$ is an effectively enumerable set of all quantifier free integer-linear arithmetic formulas, with one fixed free variable x, e.g., $[\![0 < x \wedge x + 1 < 3]\!] = [\![0 < x]\!] \cap [\![x + 1 < 3]\!] = \{1\}$. ☒

In this paper we focus on *binary* trees. This will keep the notational overhead at a minimum, while the results can be generalized to non-binary trees through standard encoding techniques. $\mathscr{T}(\mathscr{B})$ is the smallest set such that the *empty tree* $\epsilon \in \mathscr{T}(\mathscr{B})$ and if $a \in \mathscr{B}$ and $t_1, t_2 \in \mathscr{T}(\mathscr{B})$ then $t = \langle a, t_1, t_2\rangle \in \mathscr{T}(\mathscr{B})$, where a is the *label* of t, denoted $label(t)$, t_1 is the *left subtree* of t, denoted $left(t)$, and t_2 is the *right subtree* of t, denoted $right(t)$.

For example $\langle 1, \langle -2, \epsilon, \langle 3, \epsilon, \epsilon\rangle\rangle, \langle 4, \epsilon, \epsilon\rangle\rangle \in \mathscr{T}(\mathbb{Z})$.

Definition 1. A *symbolic tree automaton (STA)* A is a tuple $(\Sigma, Q, Q^{\mathrm{L}}, Q^{\mathrm{R}}, \Delta)$ where Σ is an effective Boolean algebra called the *label theory* of A, Q is a finite set of *states*, $Q^{\mathrm{L}} \subseteq Q$ is a set of *leaves*, $Q^{\mathrm{R}} \subseteq Q$ is a set of *roots*, and $\Delta \subseteq Q \times \Sigma \times Q \times Q$ is a finite set of *transitions*.

We use A as a subscript to identify a component, unless A is clear from the context. We write $\mathbf{STA}(\Sigma)$ for an effectively enumerable set of all STAs over Σ. Let $A = (\Sigma, Q, Q^{\mathrm{L}}, Q^{\mathrm{R}}, \Delta) \in \mathbf{STA}(\Sigma)$ be fixed. Given a transition $\rho = (p, \varphi, q_1, q_2) \in \Delta$, let $lhs(\rho)$, $\gamma(\rho)$, and $rhs(\rho)$, denote, respectively, the *left-hand-side* p, the *guard* φ, and the *right-hand-side* (q_1, q_2) of ρ. We use $\bar{q}$ as an abbreviation for (q_1, q_2).

Definition 2. The *language of* A *for* $q \in Q$, denoted by $\mathscr{L}(A, q)$, is the smallest subset of $\mathscr{T}(\mathscr{B})$ such that: if $q \in Q^{\mathrm{L}}$ then $\epsilon \in \mathscr{L}(A, q)$; if $(q, \varphi, q_1, q_2) \in \Delta$, $a \in [\![\varphi]\!]$, and, for $i \in \{1, 2\}$, $t_i \in \mathscr{L}(A, q_i)$, then $\langle a, t_1, t_2\rangle \in \mathscr{L}(A, q)$. The *language of* A is $\mathscr{L}(A) \stackrel{\text{def}}{=} \bigcup_{q \in Q^{\mathrm{R}}} \mathscr{L}(A, q)$.

Two STAs A and B are *equivalent*, denoted $A \equiv B$, when $\mathscr{L}(A) = \mathscr{L}(B)$.

Let $\bot_{\mathbf{STA}(\Sigma)} \stackrel{\text{def}}{=} (\Sigma, \emptyset, \emptyset, \emptyset, \emptyset)$. Thus $\mathscr{L}(\bot_{\mathbf{STA}(\Sigma)}) = \emptyset$. The following example illustrates a representation of valid Unicode character sequences as an STA that uses UTF16 encoding of surrogate pairs.[1]

The particular feature of the representation is that the trees preserve the length of the original Unicode strings as the length of the rightmost branch. The leftmost branch from any node in the tree is either the node itself when the node is not a surrogate, or a surrogate pair otherwise, and encodes thus a single Unicode symbol.

Example 2. Let BV16 stand for quantifier free 16-bit bit-vector arithmetic; BV16 is isomorphic to quantifier free integer linear arithmetic modulo 2^{16}. We use a single fixed free variable x in predicates φ over BV16, thus $[\![\varphi]\!]$ is the set of all values a such that $\varphi[x/a]$ is true. Let

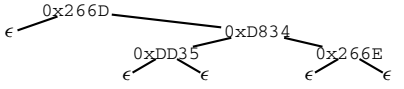
$$HighSurr \stackrel{\text{def}}{=} \texttt{0xD800} \le x \le \texttt{0xDBFF},$$

$$LowSurr \stackrel{\text{def}}{=} \texttt{0xDC00} \le x \le \texttt{0xDFFF},$$

Let $A = (\text{BV16}, \{q_{ok}, q_{ls}, q_\epsilon\}, \{q_{ok}, q_\epsilon\}, \{q_{ok}\}, \Delta)$, where

$$\Delta = \left\{ \begin{array}{l} (q_{ok}, \neg LowSurr \wedge \neg HighSurr, q_\epsilon, q_{ok}), \\ (q_{ok}, HighSurr, q_{ls}, q_{ok}), \\ (q_{ls}, LowSurr, q_\epsilon, q_\epsilon) \end{array} \right\}$$

For example, the tree



is in $\mathscr{L}(A)$ and encodes the Unicode string "♭𝆑" of musical symbols, where 𝆑 is the symbol "cut time"

[1] Complete Unicode alphabet has over one million characters, UTF16 encoding is used to encode the alphabet with 16-bit bit-vectors, where surrogate pairs are used for encoding characters in the upper Unicode range.