Available online at www.sciencedirect.com
SCIENCE DIRECT®
Information Processing Letters 94 (2005) 231–240
Information Processing Letters
www.elsevier.com/locate/ipl

Closure properties and decision problems of dag automata

Siva Anantharaman [a,*], Paliath Narendran [b], Michael Rusinowitch [c]

[a] *LIFO, Université d'Orléans, France*
[b] *University at Albany, SUNY, USA*
[c] *LORIA – Nancy, France*

Received 27 July 2004; received in revised form 5 November 2004

Available online 16 March 2005

Communicated by D. Basin

Abstract

Tree automata are widely used in various contexts. They are closed under boolean operations and their emptiness problem is decidable in polynomial time. Dag automata are natural extensions of tree automata, operating on dags instead of on trees; they can also be used for solving problems. Our purpose in this paper is to show that algebraically they behave differently: the class of dag automata is not closed under complementation, dag automata are not determinizable, their membership problem is NP-complete, the universality problem is undecidable, and the emptiness problem is NP-complete even for deterministic labeled dag automata.

Keywords: Tree automata; Determinism; Complementation; Universality problem; Emptiness problem; Formal languages

1. Introduction

The expressive power of tree automata has proved to be very useful in several contexts, such as rewriting (e.g., [8]), the analysis of XML documents (e.g., [13]), and formal program or protocol verification techniques based on set constraints [1,10]. They have also been employed in solving unification problems over theories extending ACUI (AC with Unit element plus Idempotence), see for instance [4,2]. Dag automata were first introduced as extensions of tree automata in [6]; in brief, a dag automaton is a bottom-up tree automaton which runs on dags, not on trees. A labeled dag automaton is a dag automaton where the transitions are labeled; it runs on dags with labeled nodes; the runs have then to use transitions whose labels tally with those at the nodes reached. It was shown in [2] that unification modulo ACUID (the theory obtained by adjoining a binary operator assumed two-sided distributive over a basic ACUI symbol) is decidable with a DEXPTIME lower bound and

* Corresponding author.
E-mail addresses: siva@lifo.univ-orleans.fr (S. Anantharaman), dran@cs.albany.edu (P. Narendran), rusi@loria.fr (M. Rusinowitch).

0020-0190/$ – see front matter
doi:10.1016/j.ipl.2005.02.004

a NEXPTIME upper bound complexity; this was done by formulating the problem as one of emptiness of a deterministic labeled dag automaton (LDA) that can be constructed naturally from the given unification problem, in exponential time.

Thus, if emptiness of *deterministic* LDAs could be shown to be decidable in polynomial time, one could have deduced that ACUID-unification is DEXPTIME-complete. But we shall be showing below that deciding emptiness is NP-complete for deterministic LDAs. We also establish that:

(i) the class of dag automata is not stable under complementation,
(ii) the membership problem is NP-hard for nondeterministic dag automata, and
(iii) universality is undecidable for dag automata.

The results on emptiness and membership are obtained via reduction from boolean satisfiability, while that on universality is obtained via reduction from the Minsky two-counter machine problem. These results illustrate how different the algebraic behavior of dag automata can be from that of tree automata. Observe, in this connection, that for nondeterministic tree automata, the uniform membership problem is decidable in polynomial time, and universality is known to be EXPTIME-complete, cf. TATA ([7], Section 1.7, respectively, Theorems 10 and 14).

Dag automata were studied in detail in [6]; the problem of their emptiness was shown there to be NP-complete, and their membership problem was shown to be in NP. The stability under complementation of the class of dag automata was raised as an open problem, closely linked with that of their determinization. We are thankful to the anonymous referees for having pointed out that the proof of our Theorem 1 (Section 3) settles these questions negatively.

2. Dag automata with or without labels

We first recall the notions of term-dags and of dag automata as developed in [6]. A *term-dag* over a ranked alphabet Σ is a rooted dag where each node has a symbol from Σ such that:

(i) the outdegree of the node is the same as the rank of the symbol,
(ii) edges going out of a node are ordered, and
(iii) no two distinct subgraphs are isomorphic.

Every node represents a unique term in a term-dag, so we often treat "node" and "term" as synonymous on a term-dag.

Definition 1. A term-dag automaton (or dag automaton, DA, for short) over a ranked alphabet Σ is a tuple (Σ, Q, F, Δ), where Q is a finite nonempty set of states, $F \subseteq Q$ is the set of final (or accepting) states, and Δ is a set of transition rules of the form: $f(q_1, q_2, \ldots, q_n) \to q$, where $f \in \Sigma$ is of arity (rank) n, and the $q_1, \ldots, q_n, q$ are in Q.

Note that the dag automata are defined in a bottom-up style. A *run* r of a DA $\mathbf{A} = (\Sigma, Q, F, \Delta)$ on a term-dag t is a mapping from the set of nodes of t to the set of states Q that respects the transition relation Δ; i.e., for every node u, if the symbol at u is f of arity k, then $f(r(u_1), \ldots, r(u_k)) \to r(u)$ must be a transition in Δ, where $u_1, \ldots, u_k$ are the successor-nodes of u given in order. A run r is *accepting* on t if and only if $r(t) \in F$, i.e., it maps the root node to an accepting state. A term-dag t is accepted by a DA iff there is an accepting run on t. The language of a DA is the set of all term-dags that it accepts. It has been proved in [6], that deciding the emptiness of a DA *is* in NP.

A *labeled term-dag*, or *lt*-dag, for short, is a term-dag equipped additionally with a mapping from the nodes of the dag to a given set of labels E. The motivation for adding labels is that, in the case where the labels are boolean, i.e., when $E = \{0, 1\}$, a labeled term-dag can be used to specify finite sets of terms. For instance, the labeled term-dag in Fig. 1 represents the set $\{a, g(g(a, a), b)\}$ of terms. More generally, if the labels are boolean vectors of length m, then each labeled dag corresponds to an m-tuple of finite sets of terms.

Definition 2. A *labeled dag automaton* (or LDA, for short) over a ranked alphabet Σ is a quintuple $(\Sigma, Q, F, E, \Delta)$, where Q is a finite nonempty set of states, $F \subseteq Q$ is the set of final (or accepting) states, E is a finite set of *labels*, and the transition relation Δ consists of *labeled* rewrite rules of the form $f(q_1, \ldots,$