Closure Properties and Decision Problems of Dag Automata

Siva Anantharaman
LIFO, Université d’Orléans (France), e-mail: siva@lifo.univ-orleans.fr

Paliath Narendran
University at Albany, SUNY (USA), e-mail: dran@cs.albany.edu

Michael Rusinowitch
LORIA - Nancy (France), e-mail: rus@loria.fr

Abstract

Tree automata are widely used in various contexts. They are closed under boolean operations and their emptiness problem 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 deterministic; 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.

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. They have also been employed in solving unification problems over theories extending ACUI (AC with Unit element plus Idempotence), see for instance [4] and [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 2-sided distributive over a basic ACUI symbol) is decidable with a DEXPTIME lower bound and 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 non-deterministic 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 2-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 non-deterministic 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
determination. 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 out-
degree 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 non-empty set of states, F ⊆ 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, ..., q_n) → q, where f ∈ Σ is of arity
(rank) n, and the q_1, ..., q_n, q are in Q.

Note that the dag automata are defined in a bottom-up style. A run r of a DA A = (Σ, Q, F, Δ)
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), ..., r(u_k)) → r(u)
must be a transition in Δ, where u_1, ..., u_k are the successor-nodes of u given in order. A run r is
accepting on t if and only if r(t) ∈ 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 Figure 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 in short) over a ranked alphabet Σ is a quintuple
(Σ, Q, F, E, Δ), where Q is a finite non-empty set of states, F ⊆ 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, ..., q_k) → q, where k is the rank of f, l is a label from E, and q_1, ..., q_k, q are in Q.
A run $r$ of an LDA $(\Sigma, Q, F, E, \Delta)$ on an lt-dag $t$ with label $E$ is a mapping from the nodes of $t$ to $Q$ that respects the labels and the transition relation $\Delta$ in the following sense:

- for every node $u$ on $t$, if the symbol at $u$ is $f$ of arity $k$, and the label of $t$ at $u$ is $l$, then transitions are possible via rules in $\Delta$ of the form $f(r(u_1), \ldots, r(u_k)) \rightarrow r(u)$, where $u_1, \ldots, u_k$ are the successor nodes of $u$ on $t$ given in order.

The above condition says, in intuitive terms, that the label on an LDA-transition must be the one at the node reached. A run $r$ is said to be accepting on $t$ iff $r(t) \in F$, i.e., it maps the root node to an accepting state. An lt-dag $t$ is said to be accepted by an LDA iff there is an accepting run on $t$. The language of an LDA is the set of all lt-dags that it accepts.

We shall also be using the notion of deterministic DAs and LDAs in the sequel. A dag automaton is said to be deterministic iff any two distinct transition rules have distinct left-hand-sides. A labeled dag automaton is deterministic iff no two distinct transition rules have the same left-hand-side and the same label.

**Remark 1.** If $A$ is a deterministic DA and $L$ its language, then the set of terms represented by the daggs of $L$ is a regular tree language; indeed, if an automaton is bottom-up deterministic, then there is no difference whether it runs on a tree or on the dag representing this tree.

**Lemma 1** The emptiness of any given LDA is decidable in non-deterministic polynomial time.

**Proof:** This is a consequence of the $NP$-completeness of the emptiness of the language for any given DA ([6]). Here are the details. Given the LDA $A$, construct an associated unlabeled DA denoted $A'$, as follows: the states of $A'$ are the pairs of form $(q, L_q)$, denoted as $\hat{q}$, where $q$ is a state of $A$ and $L_q$ is the set of all labels of the transitions of $A$ which have $q$ as target; the (unlabeled) transitions of $A'$ are of the form $f(\hat{q_1}, \ldots, \hat{q_k}) \rightarrow \hat{q}$, whenever $f(q_1, \ldots, q_k) \rightarrow q$ is a (labeled) transition on the given LDA $A$; the accepting states of $A'$ are the $\hat{q}$'s corresponding to the accepting states $q$ on $A$. Note that $A'$ is constructed from $A$ in linear time: its number of states is the same as for $A$ (since $L_q$ is completely determined by $q$ on $A$), and its number of transitions is at most that of $A$.

We claim that the language of the LDA $A$ is non-empty if and only if the unlabeled DA $A'$ accepts some term-dag. The ‘only if’ part of the assertion is trivial; so consider any term-dag $t'$ accepted by $A'$, and choose some accepting run $r$ of $A'$ on $t'$; then transform the term-dag $t'$ into an lt-dag (named $t$) by labeling any given node $u$ on $t'$ as follows: if $r(u) = \hat{q}$ and the transition used by the run $r$ to reach $u$ is $f(\hat{q_1}, \ldots, \hat{q_k}) \rightarrow \hat{q}$, then pick any label $l$ such that $f(q_1, \ldots, q_k) \rightarrow q$ is a transition on $A$. It is obvious that there is then an accepting run of the LDA $A$ on the lt-dag $t$ thus constructed. □

**Remark 2.** Note that, even if the LDA $A$ is deterministic, the associated DA $A'$ constructed as above will in general be non-deterministic.

**3. Algebraic Properties of DAs and LDAs**

**3.1. Complementation and Determinization**

One can prove, by standard arguments, that the class of all DAs (resp. LDAs) is stable under union and intersection; cf. e.g. [6]. The question of stability under complementation of the class of DAs, was left open in [6]. We give a negative answer here to this question.

**Theorem 1** The class of dag automata is not stable under complement.

**Proof:** Consider the infinite set $M$ of term-dags defined recursively over the signature $\{a^{(0)}, g^{(2)}\}$, as follows (the superscripts are the arities):

\begin{itemize}
  \item[i)] $a \in M$
  \item[ii)] if $t \in M$ then $g(t, t) \in M$
  \item[iii)] nothing else is in $M$
\end{itemize}

We first show that there is no DA that accepts precisely the daggs of $M$. The proof is by contradiction. Suppose there is such an automaton with its number of states $|Q| = k$. Consider then a term-dag $t$ in $M$ with at least $k + 2$ nodes. Then in any accepting run $r$ of the DA on the dag $t$, there must be $2$ distinct nodes $s_1$ and $s_2$ on $t$, neither of them the root node of $t$, such that $r(s_1) = r(s_2) = p$ for some $p \in Q$. Since neither $s_1$ nor $s_2$ is the root node of $t$, there must be a node $u$ on the dag $t$ corresponding to $g(s_2, s_2)$. We can then construct
(see Figure 2) an accepting run \( r' \) for the term-dag which is the same as before except for the first edge out of \( u \) which goes to the node \( s_1 \) instead of going to \( s_2 \). In other words, the term at node \( u \) in the new dag is \( g(s_1, s_2) \). But this term-dag clearly should not be accepted.

![Diagram](image)

Fig. 2. Transformation used in the proof of Th. 1

We show next that the complement \( M' \) of \( M \) (with respect to the set of all ground terms generated by \( \{a^{(0)}, g^{(2)}\} \), is accepted by a DA. We begin with the observation that \( M' \) is the set of all ground terms containing at least one subterm not in \( M \); more precisely, for any ground term \( t \), we have \( t \in M' \) iff \( t \) contains a subterm of the form \( g(t_1, t_2) \) with \( t_1 \neq t_2 \). Our claim is that the following DA accepts precisely the dags of terms in \( M' \):

\[
\begin{align*}
a & \rightarrow q_0 & g(q_0, q_1) & \rightarrow q_a \\
a & \rightarrow q_1 & g(q_1, q_0) & \rightarrow q_a \\
g(q_0, q_0) & \rightarrow q_0 & g(q_0, \_ ) & \rightarrow q_a \\
g(q_0, q_0) & \rightarrow q_1 & g(\_, q_0) & \rightarrow q_a \\
g(q_1, q_1) & \rightarrow q_0 & & \\
g(q_1, q_1) & \rightarrow q_1 & &
\end{align*}
\]

(where \( q_a \) is the only accepting state and \( \_ \) stands for any state). The claim is proved as follows.

Observe that the above dag automaton does not accept any term in \( M \); indeed, for all terms in \( M \) with the symbol \( g \) as root, the two strict subterms of maximal height must be the same by definition, so must be represented by the same node with only one corresponding state in any run: either \( q_0 \) or \( q_1 \), but not both; so the transitions \( g(q_0, q_1) \rightarrow q_a \) and \( g(q_1, q_0) \rightarrow q_a \) are never applicable; therefore the state \( q_a \) is never reached. On the other hand, if \( t \) is a term in \( M' \), observe that one of the transitions \( g(q_0, q_1) \rightarrow q_a \) or \( g(q_1, q_0) \rightarrow q_a \) is applicable at the subterm \( t' \) of minimal height \( t \) which is not in \( M \); this is because the two strict subterms of maximal height in \( t' \) are different, so there is a run of the DA assigning to one of them the state \( q_0 \) and \( q_1 \) to the other; the four right-hand-side rules above can then lead to a successful run of the automaton on the term-dag of \( t \). □

**Remark 3.** Along with that of stability under complement for the class of DAs, the following two questions were also raised in [6]:

1. Are DAs determinizable?
2. Does there exist a set \( T \) of term-dags recognized by a DA such that the set of all terms represented by the dags in \( T \) is not a regular tree language?

Our above construction also settles these two questions: indeed the set \( M \) defined above, as well as its complement \( M' \), are both non-regular tree languages (this is easily checked via the same pigeon-hole principle argument as above); but we just saw that the set of terms-dags of \( M' \) is recognized by the DA constructed above; we thus get a positive answer to question (2). Question (1) gets a negative answer therefrom: the DA recognizing \( M' \) cannot be determined. This is so because a deterministic DA can only recognize regular tree languages (Remark 1).

### 3.2. The Emptiness and Membership Problems

Deciding emptiness of (general, non-deterministic) dag automata has been shown to be \( \text{NP} \)-complete in [6], where it was also observed that the membership problem (i.e., checking if \( t \) is accepted by \( A \) for an arbitrarily given term-dag \( t \) and DA \( A \)) is decidable in non-deterministic linear time. Since a non-deterministic LDA can be translated into a non-deterministic DA, the above two conclusions hold also for the same problems on LDAs.

The situation is different, however, when we consider deterministic DAs or LDAs. We observed above, at the end of the previous sub-section, that deterministic DAs behave exactly like (bottom-up) deterministic tree automata, so deciding their emptiness can be done in polynomial time. It turns out however that deciding emptiness is \( \text{NP} \)-hard
for deterministic LDAs; which, therefore, do not behave like deterministic labeled tree automata.

**Theorem 2** The emptiness problem is NP-hard for deterministic LDAs.

**Proof:** The proof is by reduction from boolean satisfiability (somewhat similar to the proof of NP-hardness given in [6]). Let $B$ be an arbitrarily chosen boolean formula over a given set of boolean variables $\{x_1, \ldots, x_n\}$, and the usual boolean connectives $\{\land, \lor, \neg\}$.

Let $t_B$ be a term-dag for $B$, and $m$ its number of distinct nodes. The idea is then to construct an LDA $A$ with $2m$ states, such that $A$ accepts exactly the term-dag $t_B$ labeled suitably with boolean values 0, 1 if and only if $B$ is satisfiable. The construction goes as follows.

Corresponding to each node we have two states which stand for that sub-formula getting the corresponding truth-value. For ease of exposition we represent the states in the form $q(s_0)$ or $q(s_1)$ where $s$ is a subterm of $t_B$. The labels are 0 and 1. The transition rules on $A$ are of the following form:

$x_i \xrightarrow{0} q(x_i, 0)$

$x_i \xrightarrow{1} q(x_i, 1)$,

and, for $h \in \{\land, \lor\}$, transition rules of the form:

$h(q(s_1, b_1), q(s_2, b_2)) \xrightarrow{h(b_1, b_2)} q(h(s_1, s_2), h(b_1, b_2))$

where $b_1, b_2$ are boolean values and $h(s_1, s_2)$ is a subterm of $t_B$; for the connective $\neg$, the transitions will be of the form: $\neg(q(s, a)) \xrightarrow{\neg b} q(\neg(s, \neg a))$. The only accepting state is $q(t_B, 1)$.

It follows directly from these definitions that the $\ell$-dags accepted by the LDA $A$ are exactly those obtained by labeling the nodes of $t_B$ with 0 or 1, in such a way that the formula $B$ is satisfiable; i.e., the language of $A$ is non-empty iff $B$ is satisfiable.

Finally, note that the LDA $A$ is deterministic: for a given left-hand-side node and a label, at most one transition can be fired; for instance, if the node $x_i$ on $t_B$ is given the boolean label 1, then the only legal labeled transition is $x_i \xrightarrow{1} q(x_i, 1)$.

**Proof:** The same construction as above works, except that we replace now the transition labels 0 and 1 with a single ‘don’t-care’ boolean label $x$ and adopt the boolean conventions: $x = x \lor 0 = x \land 1$, $x \land 0 = 0 = x \land \neg x$, $\neg \neg x = x$, $x \land x = x \lor x$, $x \lor 1 = 1 = x \lor \neg x$. Then, with the above notation, the given boolean formula $B$ is satisfiable iff there is an accepting run of the LDA (thus modified) on the term-dag $t_B$. Note however that the modified labels render the LDA non-deterministic, and the LDA thus modified may accept several other term-dags. □

**Remarks 4.** i) Theorem 2 above is not in conflict with the observation made in Remark 2, namely, that deterministic DAs behave like deterministic bottom-up tree automata (for which emptiness is decidable in polynomial time). The reason is that a deterministic LDA can be ‘translated’ in general only to a non-deterministic DA, cf. Remark 1.

ii) This also brings into evidence that deterministic LDAs do not behave like deterministic labeled tree automata; indeed, on a tree the same subterm can be at two different nodes with different labels. Thus, a deterministic labeled tree automaton can be easily constructed to accept the boolean formula $a \land \neg a$ as a suitably labeled tree.

iii) The above results, combined with the upper bounds obtained in [6], imply that the uniform membership problem for (general, non-deterministic) LDAs is NP-complete; and the same holds also for the emptiness problem on deterministic LDAs. Note however, that this does not follow directly from the results of [6], established for the non-deterministic case.

4. Universality of DAs is Undecidable

For the sake of being complete in this study, we consider now the universality problem for DAs. This problem is formulated as follows:

**Input:** A dag automaton $A$

**Question:** Does $A$ accept all inputs?

**Theorem 4** The universality problem for DAs is undecidable.

As an immediate consequence, we deduce that:

**Corollary 1** It is undecidable if two arbitrarily given DAs are equivalent. □
The proof of the theorem, to be given below, is via reduction from the halting problem for a deterministic 2-counter machine \((q_0, F, Q, \Delta)\) where \(Q\) is a finite set of states, \(F \subseteq Q\) is the set of final states, \(q_0\) is the initial state and \(\Delta\) is a transition relation. Each transition of the machine must correspond to a ‘correct’ instruction, e.g., as described in [12] or in [3]; the only difference here is that a computation of the machine is accepted if it leads from the ‘initial’ state \(q_0\), with initial counter values both 0, to an accepting state in \(F\). We shall actually design a non-deterministic dag automaton \(A\) accepting precisely all the non-accepting or incorrect computations of the deterministic 2-counter machine.

The non-universality of \(A\) is then equivalent to the existence of an accepting correct computation of the 2-counter machine. The automaton \(A\) is designed by encoding the non-accepting or incorrect computations of the machine in terms of an appropriate set of ground rewrite rules, which will define the transition rules of \(A\). For this encoding, we first consider the alphabet \(Q \cup \{s, 0\}\) where every symbol of the set of states of the machine, \(Q = \{q_0, \ldots, q_n\}\), will be seen as a ternary symbol, \(s\) is a new unary symbol and 0 is a constant. The integer value \(n\) for a counter will be represented as \(s^n(0)\).

A machine configuration can be represented as a triple \(\langle q, c_1, c_2 \rangle\), where \(q\) is the current state and \(c_1, c_2\) are the current counter values. The transitions of the machine are of the following two types:

- type (i): \(\langle q', x, y \rangle \rightarrow \langle q, s(x), y \rangle\),
- type (ii): \(\langle q', 0, y \rangle \rightarrow \langle q, 0, y \rangle\),

or \(\langle q', s(x), y \rangle \rightarrow \langle q''', x, y \rangle\);

the former increments the first counter, the latter tests for zero the first counter. (We also have similar transitions for the second counter.) The counter machine is deterministic, by definition, iff any two distinct transition rules have distinct left-hand-side triples \(\langle q', x, y \rangle\).

We shall be encoding the machine configurations as term-dags. The initial configuration of the machine – i.e., the triple \(\langle q_0, 0, 0 \rangle\) – will be encoded as the dag representation of the term \(t_0 = q_0(0, 0, \bot)\), where \(\bot\) is the empty dag; a correct computation of length \(n\) corresponding to a sequence of machine instructions will be encoded as the dag representation of the term \(q_{t_n}(s^{t_n}(0), s^{t_n-1}(0), \ldots, s^0(0), t_{n-1})\) where \(q_{t_n}(s^{t_n}(0), s^{t_n-1}(0), \ldots, s^0(0), t_{n-1})\) encodes a correct computation of length \(n - 1\) and the transition from configuration \(\langle q_{t_n-1}, s^{t_n-1}(0), s^{t_n-2}(0) \rangle\) to \(\langle q_{t_n}, s^t(0), s^{t-1}(0) \rangle\) is possible with the given counter machine. (Note: When a machine state \(q\) is seen as a ternary function symbol, its first two arguments stand for the two respective counter values of the machine.)

**Proof of Theorem 4:**

The desired dag automaton \(A\) will be defined as the union of four auxiliary automata \(A_1, A_2, A_3, A_4\) that we define below. (From Proposition 6 in [6], we know that the class of DAs is stable under union.) All these automata will be defined over the ranked alphabet \(\Sigma = Q \cup \{s, 0, \bot\}\).

The auxiliary automata \(A_1, A_2, A_3, A_4\) are designed as follows:

1. \(A_1\) accepts the dags that are ill-formed, i.e., dags that do not encode a sequence of machine configurations.
2. \(A_2\) accepts all the dags that correspond to runs not starting at the initial state \(\langle q_0, 0, 0 \rangle\).
3. \(A_3\) accepts all the dags that correspond to runs that do not end in a final state \(\langle q, x, y \rangle\) for some \(q \in F\) and \(x, y \in \{s^n(0) | n \geq 0\}\).
4. \(A_4\) accepts all the dags that violate the transition relation of the 2-counter machine at some sub-dag \(q(x, y, q'(x', y', z'))\).

The states of the four auxiliary automata will be chosen from the following sets of symbols:

\[
\{\sigma_q | q \in Q\} \cup \{Q_0, Q_1, Q_2, 0, error, ?\} \\
\cup \{\tau_{(q', i, j, q)} | i \in \{1, 2\}, \ j \in \{0, 1, 2, \text{zero}, \neq \text{zero}\}, q' \in Q\};
\]

and their accepting states will be chosen from:

\[
\{\text{error}, ?\} \cup \{\sigma_q | q \notin F\} \cup \{Q_0\}
\]

where \(F\) is the set of accepting states of the deterministic 2-counter machine.

Before defining the automata \(A_i\), a few words of explanation on the semantics of their state symbols. State symbol \(\sigma_q\) corresponds to current machine state \(q\). \(Q_0, i = 0, 1, 2\), respectively are states where a given counter has value \(\geq i\); and \(Q_s\) is the state where it is checked that counter values are built from 0 and \(s\). The symbol \(0\) stands for a state where the counter considered (correspond-
ing to the first or second argument position of the ternary symbol \( q \) has value 0. The symbol \( q_\bot \) is the starting state, ‘error’ a state for an incorrect machine configuration. The \( \tau_{q',i,j,q} \) are states where the current counter-machine state is \( q' \), \( q \) is the state to which a transition is envisaged, and counter \( i \) has ‘value’ \( j \in \{0,1,2, \text{zero} \neq \text{zero} \} \); as concerns the arguments of \( \tau \): the symbols 0,1,2 as values of \( j \) mean that the counter \( i \) appearing as the 2nd argument of \( \tau \) has value at least 0,1 or 2, respectively; while the zero symbol indicates that this counter \( i \) has exact value 0, and \( \neq \text{zero} \) indicates that it has some value > 0.

**Automaton A:**

1. The set of states of \( A_1 \) is
   \( \{\sigma_q | q \in Q \} \cup \{Q_s, 0, \text{error}, q_\bot \} \)
2. The set of accepting states is
   \( \{Q_s, 0, \text{error}, q_\bot \} \)
3. The transition relation \( \delta_1 \) consists of the rewrite rules specified below.
   (a) Rules for eliminating ill-formed terms:
   \[0 \rightarrow Q_s\]
   \[s(Q_s) \rightarrow Q_s\]
   (check if counters are built with 0,s)
   \[\downarrow \rightarrow q_\bot\]
   \[q(\downarrow, \downarrow, \downarrow) \rightarrow \sigma_q \text{ for all } q \in Q\]
   \[s(q_\bot) \rightarrow \text{error}\]
   \[s(\sigma_q) \rightarrow \text{error for all } q \in Q\]
   \[q(\downarrow, \downarrow, q_s) \rightarrow \text{error for all } q \in Q\]
   (no counting symbol in third position)
   \[q(\sigma_{q'\downarrow}, \downarrow, \downarrow) \rightarrow \text{error for all } q, q' \in Q\]
   (no state symbol in first position)
   \[q(\downarrow, \sigma_q' \downarrow, \downarrow) \rightarrow \text{error for all } q, q' \in Q\]
   (no state symbol in second position)
   \[q(q_\bot, \downarrow, \downarrow) \rightarrow \text{error for all } q \in Q\]
   (\( \downarrow \) not allowed in the first position)
   \[q(\downarrow, q_\bot, \downarrow) \rightarrow \text{error for all } q \in Q\]
   (\( \downarrow \) not allowed in the second position)
   (b) Rules for propagating errors up to the root:
   \[q(\text{error}, \downarrow, \downarrow) \rightarrow \text{error}\]
   \[q(\downarrow, \text{error}, \downarrow) \rightarrow \text{error}\]
   \[q(\downarrow, \downarrow, \text{error}) \rightarrow \text{error}\]
   \[s(\text{error}) \rightarrow \text{error}\]

We may assume now that the other automata \( A_2, A_2, A_3 \) run on well-formed dags, since \( A_1 \) accepts all the ill-formed ones.

**Automaton A**₂:

1. The set of states of \( A_2 \) is
   \( \{\sigma_q | q \in Q \} \cup \{\text{error}, q_\bot, Q_0, Q_1 \} \)
2. The set of accepting states is \( \{\text{error} \} \)
3. The transition relation \( \delta_2 \) consists of the rewrite rules specified below.
   (a) The initial state is \( q_0, \) counters are initially 0:
   \[\downarrow \rightarrow q_\bot\]
   \[0 \rightarrow Q_0\]
   \[s(Q_0) \rightarrow Q_1\]
   \[s(Q_1) \rightarrow Q_1\]
   \[q(q_\bot, q_\bot) \rightarrow \text{error} \text{ if } q \neq q_0\]
   \[\phi(Q_1, q_\bot) \rightarrow \text{error}\]
   \[\phi(q_\bot, Q_1, q_\bot) \rightarrow \text{error}\]
   (b) Rules for propagating errors up to the root:
   Same as in the set 3.(b) of the automaton \( A_1 \).

**Automaton A**₃:

1. The set of states of \( A_3 \) is
   \( \{\sigma_q | q \in Q \} \cup \{q_\bot, Q_s, 0 \} \)
2. The set of accepting states is \( \{\sigma_q | q \not\in F \} \)
   (recall that \( F \) is the set of accepting states of the counter machine).
3. The transition relation \( \delta_3 \) consists of the rewrite rules specified below.
   \[0 \rightarrow Q_s\]
   \[\downarrow \rightarrow q_\bot\]
   \[s(Q_s) \rightarrow Q_s\]
   \[q(q_\bot, q_\bot) \rightarrow \sigma_q \text{ for all } q \in Q\]

Note that if the extracted state at the root of the dag is not accepting (i.e., \( \sigma_q \) with \( q \not\in F \)) then the dag will be accepted by our automaton \( A_3 \) (since it will not encode a successful computation).

**Automaton A**₄:

1. The set of states of \( A_4 \) is
   \( \{\sigma_q | q \in Q \} \cup \{Q_0, Q_1, Q_2, \text{error}, q_\bot \} \)
   \( \cup \{\tau_{q',q,j,q} | q', q \in Q, i \in \{1,2\}, j \in \{0,1,2, \text{zero} \neq \text{zero} \} \} \)
2. The set of accepting states is \( \{\text{error} \} \)
3. The transition relation \( \delta_4 \) consists of the rewrite rules to be specified below.
   1) Rules for transitions of type (i): \( \langle q', x, y \rangle \vdash \langle q, s(x), y \rangle \)
   incrementing the first counter. (The \( q, q' \) in these rules are the same as in the machine transition; and \( q'' \) is any state of the counter machine.)
(a) rules to count $s$:
\[
\begin{align*}
0 & \rightarrow Q_0 \\
q_0(Q_0) & \rightarrow Q_0 \\
q_1(Q_0) & \rightarrow Q_1 \quad \text{count at least one } s \\
q_2(Q_0) & \rightarrow Q_2 \quad \text{count at least two } s \\
q_3(Q_0) & \rightarrow Q_3 \quad \text{count more than two } s \\
\end{align*}
\]

(b) rule to ensure that next state is the right one:
\[
q''(\_\_\_, \sigma q) \rightarrow \text{error} \quad \text{for all } q'' \neq q
\]

(c) rules to ensure that the counter is not incremented by more than 1:
\[
\begin{align*}
q'(Q_{1\_\_\_\_\_\_}) & \rightarrow \tau(q', 1, q) \\
q'(Q_{2\_\_\_\_\_\_}) & \rightarrow \tau(q', 1, \text{zero}, q) \\
q'(Q_{0\_\_\_\_\_\_}) & \rightarrow \tau(q', 1, \text{zero}, q) \\
\end{align*}
\]

(d) rules to ensure that the counter value is not the same as before:
\[
\begin{align*}
q'(Q_{1\_\_\_\_\_\_}) & \rightarrow \tau(q', 1, \text{zero}, q) \\
q'(Q_{0\_\_\_\_\_\_}) & \rightarrow \tau(q', 2, \text{zero}, q) \\
q'(Q_{0\_\_\_\_\_\_}) & \rightarrow \tau(q', 2, \text{zero}, q) \\
\end{align*}
\]

(e) rules to ensure that first counter value is not less than previous value:
\[
\begin{align*}
q'(Q_{1\_\_\_\_\_\_}) & \rightarrow \tau(q', 1, q) \\
q'(Q_{0\_\_\_\_\_\_}) & \rightarrow \tau(q', 1, q) \\
\end{align*}
\]

We omit the similar sets of rules for transitions of type (i) which increment the second counter, of the form: \( (q', x, y) \rightarrow (q, x, s(y)) \).

II) Rules for transitions of type (ii) with zero-test on first counter:
\[
\begin{align*}
(q', 0, y) & \rightarrow (q, 0, y) \\
(q', s(x), y) & \rightarrow (q', x, y) \\
\end{align*}
\]

(In these sets of rules, $q, q', q''$ are as above, and $q_1$ is any state of the counter machine.)

(a) rules to force the correct branch:
\[
\begin{align*}
q'(Q_{0\_\_\_\_\_\_}) & \rightarrow \tau(q', 1, \text{zero}, q) \\
& \quad \text{-records first counter at } q \text{ is zero} \\
q_1(\_\_\_\_\_\_\_\_\_) & \rightarrow \tau(q', 1, \text{zero}, q) \\
& \quad \text{-forces branch to } q \\
\end{align*}
\]

\[
\begin{align*}
q'(Q_{1\_\_\_\_\_\_}) & \rightarrow \tau(q', 1, \text{zero}, q') \\
q'(Q_{2\_\_\_\_\_\_}) & \rightarrow \tau(q', 1, \text{zero}, q') \\
q_1(\_\_\_\_\_\_\_\_\_) & \rightarrow \tau(q', 1, \text{zero}, q') \\
\end{align*}
\]

(b) rule to ensure that second counter is not modified:
\[
\begin{align*}
q'(Q_{0\_\_\_\_\_\_}) & \rightarrow \tau(q', 2, \text{zero}, q) \\
q'(Q_{0\_\_\_\_\_\_}) & \rightarrow \tau(q', 2, \text{zero}, q') \\
q'(Q_{0\_\_\_\_\_\_}) & \rightarrow \tau(q', 2, \text{zero}, q') \\
q'(Q_{0\_\_\_\_\_\_}) & \rightarrow \tau(q', 2, \text{zero}, q') \\
q'(Q_{0\_\_\_\_\_\_}) & \rightarrow \tau(q', 2, \text{zero}, q') \\
\end{align*}
\]

(c) rules to ensure that first counter remains 0 (if the branch is to $q$):
\[
\begin{align*}
q'(Q_{1\_\_\_\_\_\_}) & \rightarrow \tau(q', 1, \text{zero}, q) \\
q'(Q_{1\_\_\_\_\_\_}) & \rightarrow \tau(q', 1, \text{zero}, q') \\
\end{align*}
\]

(d) rules to ensure that first counter is decremented by 1 (if the branch is to $q''$):
\[
\begin{align*}
q'(Q_{1\_\_\_\_\_\_}) & \rightarrow \tau(q', 1, q) \\
q'(Q_{0\_\_\_\_\_\_}) & \rightarrow \tau(q', 1, q') \\
q'(Q_{0\_\_\_\_\_\_}) & \rightarrow \tau(q', 1, q') \\
\end{align*}
\]

Similar sets of rules are also added for test instructions on the second counter. The rules 3.(b) (of automaton $A_k$) for propagating errors up to the root are to be added too.

Note: the rule I(b) for $A_k$ is correct, since the counter machine is assumed deterministic.

It is not hard to check that the language accepted by the dag automaton $A$ constructed above, as the union of the dag automata $A_i, i = 1 \ldots 4$, is the set of all term-dags which correspond to machine configurations which are either incorrect or unaccepted by the 2-counter machine: the transitions and the accepting states have actually been tailored exactly with such a purpose.

Note in this connection that, in the set of rules I(a) for $A_k$, the rule $s(Q_0) \rightarrow Q_0$ is needed in
order that the rules of $I(b)$ can correctly play the role they are specified for. For instance, here is an accepting run on $A$, for the incorrect machine configuration $q_1(s^2(0),0,q_0(0,0,\bot))$. We have:
\[
\bot \rightarrow q_1 \quad \text{and} \quad 0 \rightarrow Q_0,
\]
so the subdag with root at $q_0$ can be mapped, via $s(Q_0) \rightarrow Q_0$ and $q_0(Q_0,\bot,q_1) \rightarrow \tau_{q_1,0,q_1}$ to the state $\tau_{q_1,0,q_1}$; then, the entire term-dag rooted at $q_1$ can be mapped to the accepting state 'error' on $A$, under the transition $q_1(q_2,\bot) \rightarrow \tau_{q_1,0,q_1} \rightarrow \text{error}$. □

Remark 5. A tree automaton accepting precisely the incorrect or non-accepting counter machine computations cannot be constructed along the same lines of reasoning. For instance, rules like $I(b)$ on $A_4$ will not suffice to ensure that two values are the same. Besides, on a tree automaton with these transitions, the terms $s^2(0)$ and $s(0)$ can be mapped independently to $Q_2$ and to $Q_0$ respectively; and a run can be conceived to map the root of the term $q_2(s^2(0),0,q_1(s(0),0,q_0(0,0,\bot)))$ to the state error, although it defines a correct machine configuration; this is obviously not possible on a DA. □

5. Conclusion

We have shown in this work that dag automata behave algebraically very differently from tree automata. We saw in Section 4 however, that they could be conveniently used for encoding some complex situations; and we also saw (proofs of Theorems 2, 3) that labels at nodes and on transitions can be used (as do the LDAs), to render the analysis finer. We are therefore led to believe that DAs and LDAs may have some practical applications, such as e.g., for the representation and/or analysis of semi-structured XML documents. Indeed, XML contains a mechanism of references where unique identifiers are associated to elements as attributes; the natural representation for such documents are dags; cf. [11] for some complexity results on evaluating XPath on dags. Moreover, the dag representation is clearly space efficient for compressed XML documents, cf. [5,9]. Dag automata, with or without labels, may therefore be useful for the treatment of certain classes of XML/XPath queries; investigating this potential application area is part of our planned future work.

References


