UNIVERSITE D'ORLEANS
Faculté des Sciences
LIFO

Laboratoire d'Informatique Fondamentale d'Orléans
4, rue Léonard de Vinci, BP 6759
F-45067 Orléans Cedex 2
FRANCE

Rapport de Recherche
www : http://www.univ-orleans.fr/SCIENCES/LIFO/

How Useful are Dag Automata ?

S. ANANTHARAMAN, LIFO, Orléans (Fr.)
P. NARENDRAN, SUNY at Albany-NY (USA)
M. RUSINOWITCH, LORIA, Nancy (Fr.)

Rapport N^o **2004-12**

How Useful are Dag Automata ?

Siva Anantharaman[1], Paliath Narendran[2], Michael Rusinowitch[3]

[1] LIFO - Orléans (France), e-mail: siva@lifo.univ-orleans.fr
[2] University at Albany–SUNY (USA), e-mail: dran@cs.albany.edu
[3] LORIA - Nancy (France), e-mail: rusi@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 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, E-Unification.

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., [14]), 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 uniform 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, as compared to 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 determinization. The proof of our Theorem 1 (Section 3) settles these questions negatively.

Despite these negative results concerning their algebraic behavior, a positive message that we want to convey in this paper is that dag automata can be the appropriate tools for handling some practical situations. For the sake of completeness, we also reproduce in Appendix I, the full details of the LDA approach for solving any given $AC(U)ID$-unification problem $\mathcal{P}$, given in a standard form. A labeled term dag t accepted by the LDA associated to $\mathcal{P}$ may not directly give a solution to the problem $\mathcal{P}$: the sets of ground terms derived from the labels of t may not satisfy the necessary 'closure property'; but this is repaired with the help of a grammar associated with an accepting run. Several illustrative examples are presented in Appendix II.

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, $F \subseteq Q$ is the set of final (or accepting) states, and Δ is a set of transition rules of the form : $f(q_1, q_2, ..., q_n) \to q$, where $f \in \Sigma$ is of arity (rank) n, and the $q_i, \dots, q_n, q$ 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 $f(r(u_1), \dots, r(u_k)) \to r(u)$ must be a transition in Δ, where $u_1, \dots, u_k$ 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 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 $(\Sigma, Q, F, E, \Delta)$, where Q is a finite non-empty 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 $f(q_1, \dots, q_k) \xrightarrow{l} q$, where k is the rank of f, l is a label from E, and $q_1, \dots, q_k, q$ 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 Δ in the following sense:

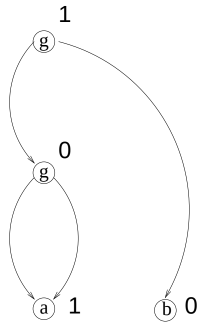

Figure 1: A labeled term-dag

- 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 Δ of the form $f(r(u_1), \ldots, r(u_k)) \xrightarrow{l} r(u)$, where u_1, …, 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 $\mathbf{A}$ is a deterministic DA and L its language, then the set of terms represented by the dags 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 complexity of the emptiness of the language for any given DA ([6]). Here are the details. Given the LDA $\mathbf{A}$, construct an associated unlabeled DA denoted $\mathbf{A}'$, as follows: the states of $\mathbf{A}'$ are the pairs of form (q, L_q), denoted as $\hat{q}$, where q is a state of $\mathbf{A}$ and L_q is the set of all labels of the transitions of $\mathbf{A}$ which have q as target; the (unlabeled) transitions of $\mathbf{A}'$ are of the form $f(\hat{q_1}, \ldots, \hat{q_k}) \longrightarrow \hat{q}$, whenever $f(q_1, \ldots, q_k) \xrightarrow{l} q$ is a (labeled) transition on the given LDA $\mathbf{A}$; the accepting states of $\mathbf{A}'$ are the $\hat{q}$'s corresponding to the accepting states q on $\mathbf{A}$. Note that $\mathbf{A}'$ is constructed from $\mathbf{A}$ in linear time: its number of states is the same as for $\mathbf{A}$ (since L_q is completely determined by q on $\mathbf{A}$), and its number of transitions is at most that of $\mathbf{A}$.

We claim that the language of the LDA $\mathbf{A}$ is non-empty if and only if the unlabeled DA $\mathbf{A}'$ accepts some term-dag. The ‘only if’ part of the assertion is trivial; so consider any term-dag t' accepted by $\mathbf{A}'$, and choose some accepting run r of $\mathbf{A}'$ 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 $f(\hat{q_1}, \ldots, \hat{q_k}) \longrightarrow \hat{q}$, then pick any label l such that $f(q_1, \ldots, q_k) \xrightarrow{l} q$ is a transition on $\mathbf{A}$. It is obvious that there is then an accepting run of the LDA $\mathbf{A}$ on the lt-dag t thus constructed. □

Remark 2. Note that, even if the LDA $\mathbf{A}$ is deterministic, the associated DA $\mathbf{A}'$ 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):

i) $a \in M$
ii) if $t \in M$ then $g(t,t) \in M$
iii) nothing else is in M

We first show that there is no DA that accepts precisely the dags 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.

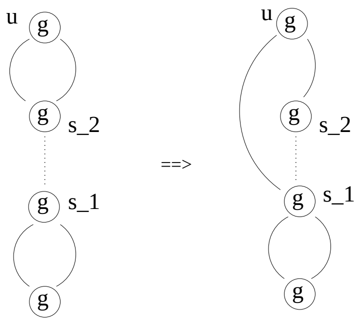

Figure 2: Transformation used in the proof of Th. 1

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':

$$
\begin{array}{rcl@{\qquad\qquad}rcl}
a & \longrightarrow & q_0 & g(q_0,\, q_1) & \longrightarrow & q_a \\
a & \longrightarrow & q_1 & g(q_1,\, q_0) & \longrightarrow & q_a \\
g(q_0,\, q_0) & \longrightarrow & q_0 & g(q_a,\, \_) & \longrightarrow & q_a \\
g(q_0,\, q_0) & \longrightarrow & q_1 & g(\_,\, q_a) & \longrightarrow & q_a \\
g(q_1,\, q_1) & \longrightarrow & q_0 & & & \\
g(q_1,\, q_1) & \longrightarrow & q_1 & & &
\end{array}
$$

(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 $g(q_0, q_1) \to q_a$ and $g(q_1, q_0) \to q_a$ 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 $g(q_0, q_1) \to q_a,\ g(q_1, q_0) \to q_a$ 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 determinized. 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 NP-complete in [6], where it was also observed that the membership problem (i.e., checking if t is accepted by $\mathbf{A}$ for an arbitrarily given term-dag t and DA $\mathbf{A}$) 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 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 any arbitrarily chosen boolean formula over a given set of boolean variables $\{x_1, \ldots, x_n\}$, and the usual boolean connectives $\{\wedge, \vee, \neg\}$.

Let t_B be a term-dag for B, and m its number of distinct nodes. The idea is then to construct an LDA $\mathbf{A}$ with $2m$ states, such that $\mathbf{A}$ 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 $\mathbf{A}$ are of the following form:

$$x_i \xrightarrow{0} q_{(x_i,0)}, \qquad\qquad x_i \xrightarrow{1} q_{(x_i,1)}$$

and, for $h \in \{\wedge, \vee\}$, 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: $\neg(q_{(s,b)}) \xrightarrow{\neg b} q_{(\neg s, \neg b)}$ The only accepting state is $q_{(t_B,1)}$.

It follows directly from these definitions that the lt-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 $x_i \xrightarrow{1} q_{(x_i,1)}$. □

The construction used in the above proof helps us also to derive a lower bound for the (uniform) membership problem for DAs and LDAs.

Theorem 3 *The membership problem for LDAs is NP-hard.*

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: $x = x \vee 0 = x \wedge 1$, $x \wedge 0 = 0 = x \wedge \neg x$, $\neg\neg x = x$, $x \wedge x = x = x \vee x$, $x \vee 1 = 1 = x \vee \neg x$. 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 $a \wedge \neg a$ 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, Δ) where Q is a finite set of states, $F \subseteq Q$ is the set of final states, q_0 is the initial state and Δ is a transition relation. Each transition of the machine must correspond to a 'correct' instruction, e.g., as described in [13] 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 $\mathbf{A}$ accepting precisely all the non-accepting or incorrect computations of the deterministic 2-counter machine.

The non-universality of $\mathbf{A}$ is then equivalent to the existence of an accepting correct computation of the 2-counter machine. The automaton $\mathbf{A}$ 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 $\mathbf{A}$. 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): $\langle q', x, y \rangle \ \vdash \ \langle q, s(x), y \rangle$,
type (ii): $\langle q', 0, y \rangle \ \vdash \ \langle q, 0, y \rangle$,
or $\langle q', s(x), y \rangle \ \vdash \ \langle q'', x, y \rangle$;

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 $t_0 = q_0(0, 0, \perp)$, where $\perp$ 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

$$q_{i_n}(s^{l_n}(0), s^{r_n}(0), q_{i_{n-1}}(s^{l_{n-1}}(0), s^{r_{n-1}}(0), t_{n-1}))$$

where $q_{i_{n-1}}(s^{l_{n-1}}(0), s^{r_{n-1}}(0), t_{n-1})$ encodes a correct computation of length $n-1$ and the transition from configuration $\langle q_{i_{n-1}}, s^{l_{n-1}}(0), s^{r_{n-1}}(0) \rangle$ to $\langle q_{i_n}, s^{l_n}(0), s^{r_n}(0) \rangle$ 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 $\mathbf{A}$ will be defined as the union of four auxiliary automata $\mathbf{A}_1, \mathbf{A}_2, \mathbf{A}_3, \mathbf{A}_4$ 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 $\Sigma = Q \cup \{s, 0, \perp\}$.

The auxiliary automata $\mathbf{A}_1, \mathbf{A}_2, \mathbf{A}_3, \mathbf{A}_4$ are designed as follows:

1. $\mathbf{A}_1$ accepts the dags that are ill-formed, i.e., dags that do not encode a sequence of machine configurations.

2. $\mathbf{A}_2$ accepts all the dags that correspond to runs not starting at the initial state $\langle q_0, 0, 0 \rangle$.

3. $\mathbf{A}_3$ 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 $x, y \in \{s^n(0) \mid n \geq 0\}$.

4. $\mathbf{A}_4$ 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:

$$\{\sigma_q \,|\, q \in Q\} \ \cup \ \{\mathbf{Q}_s, \mathbf{Q}_0, \mathbf{Q}_1, \mathbf{Q}_2, \mathbf{0}, \mathtt{error}, q_\perp\} \\ \cup \ \{\tau_{(q',i,j,q)} \,|\, i \in \{1,2\}, \\ j \in \{0, 1, 2, \mathtt{zero}, \neq \mathtt{zero}\}, q', q \in Q\};$$

and their accepting states will be chosen from:

$$\{\mathtt{error}, q_\perp\} \cup \{\sigma_q \,|\, q \notin F\} \cup \{\mathbf{Q}_s\}$$

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

Before defining the automata $\mathbf{A}_i$, a few words of explanation on the semantics of their state symbols. State symbol σ_q corresponds to current machine state q. $\mathbf{Q}_i, i = 0, 1, 2$, respectively are states where a given counter has value $\geq i$; and $\mathbf{Q}_s$ is the state where it is checked that counter values are built from 0 and s. The symbol $\mathbf{0}$ stands for a state where the counter considered (corresponding to the first or second argument position of the ternary symbol q) has value 0. The symbol $q_\perp$ 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' $j \in \{0, 1, 2, \mathtt{zero}, \neq \mathtt{zero}\}$; as concerns the arguments of τ: the symbols $0, 1, 2$ as values of j mean that the counter i appearing as the $2nd$ argument of τ has value at least $0, 1$ or 2, respectively; while the `zero` symbol indicates that this counter i has exact value 0, and $\neq$ `zero` indicates that it has some value > 0.

Automaton $\mathbf{A}_1$:

1. The set of states of $\mathbf{A}_1$ is
   $\{\sigma_q \mid q \in Q\} \cup \{\mathbf{Q}_s, \mathbf{0}, \mathtt{error}, q_\perp\}$
2. The set of *accepting* states is
   $\{\mathbf{Q}_s, \mathbf{0}, \mathtt{error}, q_\perp\}$
3. The transition relation δ_1 consists of the rewrite rules specified below.

   (a) Rules for eliminating ill-formed terms:
   $0 \longrightarrow \mathbf{Q}_s$
   $s(\mathbf{Q}_s) \longrightarrow \mathbf{Q}_s$
   (check if counters are built with $0, s$)
   $\perp \longrightarrow q_\perp$
   $q(\_, \_, \_) \longrightarrow \sigma_q$ for all $q \in Q$
   $s(q_\perp) \longrightarrow \mathtt{error}$
   $s(\sigma_q) \longrightarrow \mathtt{error}$ for all $q \in Q$
   $q(\_, \_, \mathbf{Q}_s) \longrightarrow \mathtt{error}$ for all $q \in Q$
   (no counting symbol in third position)
   $q(\sigma_{q'}, \_, \_) \longrightarrow \mathtt{error}$ for all $q, q' \in Q$ (no state symbol in first position)
   $q(\_, \sigma_{q'}, \_) \longrightarrow \mathtt{error}$ for all $q, q' \in Q$
   (no state symbol in second position)
   $q(q_\perp, \_, \_) \longrightarrow \mathtt{error}$ for all $q \in Q$
   ($\perp$ not allowed in the first position)
   $q(\_, q_\perp, \_) \longrightarrow \mathtt{error}$ for all $q \in Q$
   ($\perp$ not allowed in the second position)

   (b) Rules for propagating errors up to the root:
   $q(\mathtt{error}, \_, \_) \longrightarrow \mathtt{error}$
   $q(\_, \mathtt{error}, \_) \longrightarrow \mathtt{error}$
   $q(\_, \_, \mathtt{error}) \longrightarrow \mathtt{error}$
   $s(\mathtt{error}) \longrightarrow \mathtt{error}$

We may assume now that the other automata $\mathbf{A}_2, \mathbf{A}_2, \mathbf{A}_3$ run on well-formed dags, since $\mathbf{A}_1$ accepts all the ill-formed ones.

Automaton $\mathbf{A}_2$:

1. The set of states of $\mathbf{A_2}$ is
   $\{\sigma_q \mid q \in Q\} \cup \{\mathtt{error}, q_\perp, \mathbf{Q}_0, \mathbf{Q}_1\}$,
2. The set of *accepting* states is $\{\mathtt{error}\}$

3. The transition relation δ_2 consists of the rewrite rules specified below.

(a) The initial state is q_0, counters are initially 0:

$\perp \longrightarrow q_\perp$
$0 \longrightarrow \mathbf{Q}_0$
$s(\mathbf{Q}_0) \longrightarrow \mathbf{Q}_1$
$s(\mathbf{Q}_1) \longrightarrow \mathbf{Q}_1$
$q(\_,\_,q_\perp) \longrightarrow \texttt{error}$ if $q \neq q_0$
$q_0(\mathbf{Q}_1,\_,q_\perp) \longrightarrow \texttt{error}$
$q_0(\_,\mathbf{Q}_1,q_\perp) \longrightarrow \texttt{error}$

(b) Rules for propagating errors up to the root:
Same as in the set 3.(b) of the automaton $\mathbf{A}_1$.

Automaton $\mathbf{A}_3$:

1. The set of states of $\mathbf{A_3}$ is
$\{\sigma_q \,|\, q \in Q\} \cup \{q_\perp, \mathbf{Q}_s, \mathbf{0}\}$,

2. The set of *accepting* states is $\{\sigma_q \,|\, q \notin F\}$
(recall that F is the set of accepting states of the counter machine).

3. The transition relation δ_3 consists of the rewrite rules specified below.

$0 \longrightarrow \mathbf{Q}_s$
$\perp \longrightarrow q_\perp$
$s(\mathbf{Q}_s) \longrightarrow \mathbf{Q}_s$
$q(\_,\_,\_) \longrightarrow \sigma_q$ for all $q \in Q$

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

Automaton $\mathbf{A}_4$:

1. The set of states of $\mathbf{A_4}$ is
$\{\sigma_q \,|\, q \in Q\} \cup \{\mathbf{Q}_0, \mathbf{Q}_1, \mathbf{Q}_2, \texttt{error}, q_\perp\}$
$\cup \ \{\tau_{(q',i,j,q)} \,|\, q', q \in Q,\ i \in \{1,2\},$
$j \in \{0,1,2,\texttt{zero}, \neq \texttt{zero}\}, \}$

2. The set of *accepting* states is $\{\texttt{error}\}$

3. The transition relation δ_4 consists of the rewrite rules to be specified below.

I) 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:

$0 \longrightarrow \mathbf{Q}_0$
$s(\mathbf{Q}_0) \longrightarrow \mathbf{Q}_0$
$s(\mathbf{Q}_0) \longrightarrow \mathbf{Q}_1$ count at least one s
$s(\mathbf{Q}_1) \longrightarrow \mathbf{Q}_2$ count at least two s
$s(\mathbf{Q}_2) \longrightarrow \mathbf{Q}_2$ count more than two s

(b) rule to ensure that next state is the right one:

$q''(\_,\_,\sigma_{q'}) \longrightarrow \texttt{error}$ for all $q'' \neq q$

(c) rules to ensure that the counter is not
incremented by more than 1:

$q'(\mathbf{Q}_0, \_, \_) \longrightarrow \tau_{(q',1,0,q)}$
$q(\mathbf{Q}_2, \_, \tau_{(q',1,0,q)}) \longrightarrow \texttt{error}$

(d) rules to ensure that the counter value is not
the same as before:

$q'(\mathbf{Q}_1, \_, \_) \longrightarrow \tau_{(q',1,\neq\texttt{zero},q)}$
$q(\mathbf{Q}_1, \_, \tau_{(q',1,\neq\texttt{zero},q)}) \longrightarrow \texttt{error}$
$q'(\mathbf{0}, \_, \_) \longrightarrow \tau_{(q',1,\texttt{zero},q)}$
$q(\mathbf{0}, \_, \tau_{(q',1,\texttt{zero},q)}) \longrightarrow \texttt{error}$

(e) rules to ensure that first counter value is not
less than previous value:

$q'(\mathbf{Q}_1, \_, \_) \longrightarrow \tau_{(q',1,1,q)}$
$q(\mathbf{Q}_0, \_, \tau_{(q',1,1,q)}) \longrightarrow \texttt{error}$

(f) rules to ensure that the second counter is
not modified:

$q'(\_, \mathbf{Q}_0, \_) \longrightarrow \tau_{(q',2,0,q)}$
$q(\_, \mathbf{Q}_1, \tau_{(q',2,0,q)}) \longrightarrow \texttt{error}$
$q'(\_, \mathbf{Q}_1, \_) \longrightarrow \tau_{(q',2,1,q)}$
$q(\_, \mathbf{Q}_0, \tau_{(q',2,1,q)}) \longrightarrow \texttt{error}$
$q(\_, \mathbf{Q}_2, \tau_{(q',2,1,q)}) \longrightarrow \texttt{error}$
$q'(\_, \mathbf{Q}_j, \_) \longrightarrow \tau_{(q',2,\neq\texttt{zero},q)}$ if $j \neq 0$
$q(\_, \mathbf{0}, \tau_{(q',2,\neq\texttt{zero},q)}) \longrightarrow \texttt{error}$

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

II) Rules for transitions of type (ii) with zero-test on first counter:

$\langle q', 0, y\rangle \vdash \langle q, 0, y\rangle$ and
$\langle q', s(x), y\rangle \vdash \langle q'', x, y\rangle$.

(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:

$q'(\mathbf{0}, \_, \_) \longrightarrow \tau_{(q',1,\texttt{zero},q)}$
-records first counter at q' is $\texttt{zero}$
$q_1(\_, \_, \tau_{(q',1,\texttt{zero},q)}) \longrightarrow \texttt{error}$
for all $q_1 \neq q$ (*forces branch to* q)
$q'(\mathbf{Q}_1, \_, \_) \longrightarrow \tau_{(q',1,\neq\texttt{zero},q'')}$
$q'(\mathbf{Q}_2, \_, \_) \longrightarrow \tau_{(q',1,\neq\texttt{zero},q'')}$
-records first counter at q' is $\neq\texttt{zero}$
$q_1(\_, \_, \tau_{(q',1,\neq zero,q'')}) \longrightarrow \texttt{error}$
for all $q_1 \neq q''$ (*forces branch to* q'')

(b) rules to ensure that second counter is
not modified:

$q'(\_, \mathbf{Q}_0, \_) \longrightarrow \tau_{(q',2,0,q)}$
$q'(\_, \mathbf{Q}_0, \_) \longrightarrow \tau_{(q',2,0,q'')}$
$q'(\_, \mathbf{Q}_1, \_) \longrightarrow \tau_{(q',2,1,q)}$
$q'(\_, \mathbf{Q}_1, \_) \longrightarrow \tau_{(q',2,1,q'')}$
$q(\_, \mathbf{Q}_1, \tau_{(q',2,0,q)}) \longrightarrow \texttt{error}$
$q(\_, \mathbf{Q}_0, \tau_{(q',2,1,q)}) \longrightarrow \texttt{error}$
$q(\_, \mathbf{Q}_2, \tau_{(q',2,1,q)}) \longrightarrow \texttt{error}$
$q(\_, \mathbf{0}, \tau_{(q',2,\neq\texttt{zero},q)}) \longrightarrow \texttt{error}$

$q''(\_, \mathbf{Q}_1, \tau_{(q',2,0,q'')}) \longrightarrow \texttt{error}$
$q''(\_, \mathbf{Q}_0, \tau_{(q',2,1,q'')}) \longrightarrow \texttt{error}$
$q''(\_, \mathbf{Q}_2, \tau_{(q',2,1,q'')}) \longrightarrow \texttt{error}$
$q'(\_, \mathbf{Q}_j, \_) \longrightarrow \tau_{(q',2,\neq\texttt{zero},q)}$ if $j \neq 0$
$q'(\_, \mathbf{Q}_j, \_) \longrightarrow \tau_{(q',2,\neq\texttt{zero},q'')}$ if $j \neq 0$
$q''(\_, \mathbf{0}, \tau_{(q',2,\neq\texttt{zero},q'')}) \longrightarrow \texttt{error}$

(c) rules to ensure that first counter remains 0 (if the branch is to q):
$q(\mathbf{Q}_1, \_, \tau_{(q',1,\texttt{zero},q)}) \longrightarrow \texttt{error}$

(d) rules to ensure that first counter is decremented by 1 (if the branch is to q''):
$q'(\mathbf{Q}_1, \_, \_) \longrightarrow \tau_{(q',1,1,q'')}$
$q''(\mathbf{Q}_j, \_, \tau_{(q',1,1,q'')}) \longrightarrow \texttt{error}$ for $j \neq 0$
$q'(\mathbf{Q}_2, \_, \_) \longrightarrow \tau_{(q',1,2,q'')}$
$q''(\mathbf{Q}_0, \_, \tau_{(q',1,2,q'')}) \longrightarrow \texttt{error}$

Similar sets of rules are also added for test instructions on the second counter. The rules 3.(b) (of automaton $\mathbf{A}_1$) for propagating errors up to the root are to be added too.

Note: the rule **I**)(b) for $\mathbf{A}_4$ is correct, since the counter machine is assumed deterministic.

It is not hard to check that the language accepted by the dag automaton $\mathbf{A}$ constructed above, as the union of the dag automata $\mathbf{A}_i, i = 1..4$, 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 $\mathbf{A}_4$, the rule $s(\mathbf{Q}_0) \longrightarrow \mathbf{Q}_0$ 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 $\mathbf{A}$, for the incorrect machine configuration $q_1(s^2(0), 0, q_0(0, 0, \perp))$. We have:

$\perp \longrightarrow q_\perp$ and $0 \longrightarrow \mathbf{Q}_0$,

so the subdag with root at q_0 can be mapped, via

$s(\mathbf{Q}_0) \longrightarrow \mathbf{Q}_0$ and $q_0(\mathbf{Q}_0, \_, q_\perp) \longrightarrow \tau_{(q_0,1,0,q_1)}$

to the state $\tau_{(q_0,1,0,q_1)}$; then, the entire term-dag rooted at q_1 can be mapped to the accepting state '`error`' on $\mathbf{A}$, under the transition $q_1(\mathbf{Q}_2, \_, \tau_{(q_0,1,0,q_1)}) \longrightarrow \texttt{error}$. □

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 **II**)(b) on $\mathbf{A}_4$ 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 $\mathbf{Q}_2$ and to $\mathbf{Q}_0$ respectively; and a run can be conceived to map the root of the *term* $q_2(s^2(0), 0, q_1(s(0), 0, q_0(0, 0, \perp)))$ 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. [12] 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

[1] A. Aiken, D. Kozen,, M. Vardi, E. Wimmers. *The Complexity of Set Constraints.* In Proc. CSL'93, EACSL, September 1993, pp. 1–18.

[2] S. Anantharaman, P. Narendran, M. Rusinowitch, *ACID-Unification is NEXPTIME-Decidable*, In Proc. MFCS'03, Springer-Verlag, LNCS 2747, pp. 169–179.

[3] S. Anantharaman, P. Narendran, M. Rusinowitch, *Unification modulo ACUI plus Distributivity Axioms*, In Journal of Automated Reasoning, Vol. 33, $n^0 1$, 2004.

[4] F. Baader, P. Narendran. *Unification of Concept Terms in Description Logics.* Journal of Symbolic Computation 31 (3):277–305, 2001.

[5] P. Buneman, M. Grohe, C. Koch. *Path queries on compressed XML.* In Proc. of the 29*th* Conf. on VLDB, 2003, pp. 141–152, Ed. Morgan Kaufmann.

[6] W. Charatonik. *Automata on DAG representations of finite trees.* Technical Report MPI-I-99-2-001, Max-Planck-Institut für Informatik, Saarbrücken, Germany.

[7] H. Comon, M. Dauchet, R. Gilleron, F. Jacquemard, D. Lugiez, S. Tison, M. Tommasi. *Tree Automata Techniques and Applications.* http://www.grappa.univ-lille3.fr/tata/

[8] T. Genet and F. Klay. *Rewriting for Cryptographic Protocol Verification.* In Proc. 17th CADE, vol. 1831 of *LNAI*, pp. 271-290, Springer-Verlag, 2000.

[9] M. Frick, M. Grohe, C. Koch. *Query Evaluation of Compressed Trees.* In Proc. LICS'03, IEEE, pp. 188–197.

[10] R. Gilleron, S. Tison, M. Tommasi. *Set Constraints and Tree Automata.* Information and Computation 149, 1–41, 1999. (cf. also Technical Report IT 292, Laboratoire-LIFL, Lille, 1996.)

[11] R. Gilleron, S. Tison, M. Tommasi. *Solving Systems of Set Constraints using Tree Automata.* In Proc. STACS'93, Springer-Verlag, LNCS 665, pp. 505–514.

[12] M. Marx. *XPath and Modal Logics for Finite DAGs.* In Proc. TABLEAUX 2003, vol. 2796 of *LNAI*, pp. 150–164, Springer-Verlag, 2003.

[13] M. Minsky. *Computation: Finite and Infinite Machines. Prentice-Hall International*, London, 1972.

[14] F. Neven. Automata Theory for XML Researchers, In SIGMO Record 31(3), September 2002.

6 Appendix I: Using Dag Automata for $ACID$-Unification

In what follows, by $ACUID$ we mean the following equational theory:

$$x + (y + z) \approx (x + y) + z, \qquad x + y \approx y + x$$

$$x + x \approx x, \qquad x + 0 \approx x, \qquad x * 0 \approx 0, \qquad 0 * x \approx 0$$

$$x * (y + z) \approx (x * y) + (x * z), \qquad (u + v) * w \approx (u * w) + (v * w)$$

If we eliminate the element 0 and drop the equations involving it in $ACUID$, we get the theory that we denote as $ACID$. This set of equations can be converted naturally to a convergent rewrite system, modulo the ACI-axioms for '+'; every ground term (over any given set of free constants) in normal form w.r.t. this system can be viewed as a *finite set of terms* over '$*$' and the constants: indeed '+' can be viewed as set union. An $ACUID$-unification problem with free constants is that of solving, modulo the above equational theory, a family of equations of the form: $\{s_1 = t_1, \ldots, s_k = t_k\}$ as finite sets of such ground terms for the variables of the problem. An $ACID$-unification problem with free constants can also be seen as one of solving a family of equations of the same form, with the additional restriction that the solution for the variables must all be *finite non-empty sets*.

In [3], unification modulo $ACUID$ or $ACID$ were both shown to be DEXPTIME-hard; and their NEXPTIME-decidability was deduced from that of $ACID$-unification. In the current paper we will thus be concerned only with $ACID$-unification. An $ACID$-unification problem is said to be in *standard form,* iff every equation in the problem has one of the following forms (respectively referred to as of type *'product', 'sum',* or *'constant'*):

$$x = y * z, \qquad u = v + w, \qquad u = c$$

where u, v, w, x, y, z are variables and c is any constant or 0. A given $ACID$-unification problem can be reduced to a standard form in more than one manner (via normalization and decomposition steps). Since '+' is idempotent and '$*$' distributes left and right over '+', we may view this $ACID$-unification problem as a set constraint problem; e.g. in the first case, if y and z are interpreted as sets of terms over $*$ and the constants, then $y * z = \{s * t \mid s \in y, t \in z\}$.

To every such problem $\mathcal{P}$, we shall associate a labeled dag automaton (LDA) in such a way that solving the former amounts to showing that the language of the latter is non-empty. For doing that we shall assume, as we may, that the problem is '**pruned**' in the following sense: If $\mathcal{P}$ contains a 'constant'-equation of the form $X_k = a$, then X_k is *not* the lhs ('left-hand-side') of any 'product'-equation in $\mathcal{P}$.

6.1 The LDA associated to an $ACID$-Unification Problem

We suppose given a (pruned) $ACID$-unification problem $\mathcal{P}$, and denote by $\{X_i\}_{i=1..n}$ the set of its variables. The lt-dags on which we shall consider runs of our LDA's (to be defined), are term dags over the symbol '$*$' and the given ground constants, labeled with n-bit vectors at their various nodes. These labels will in general be denoted by $(m_1, m_2, \ldots, m_n)$, $m_i \in \{0, 1\}$, $i = 1..n$. They have the semantics that $m_i = 1$ iff the subterm of the lt-dag at the current node is an element of the set X_i.

The term *label* will denote any n-bit vector. For any lt-dag t, the subterm of the dag at a node w will be denoted by t_w; the underlying term at the root node will be denoted t. The label of t at node w will be denoted by l^w; and for any i, the i-th entry of this label is denoted by l_i^w.

Definition 3 *A label m is said to be* initial w.r.t. a ground constant a *iff:*
- $m_k = 0$ *for* any k *for which* $\mathbf{P}$ *contains an equation of the form* $X_k = X_i * X_j$*;*
- $m_i = 1$ *for every* $i \in 1..n$ *for which* $X_i = a$ *is in* $\mathbf{P}$*;*
- $m_j = 0$ *for every* $j \in 1..n$ *for which* $\mathbf{P}$ *contains an equation* $X_j = b,\ b \neq a$*.*

The above definition is coherent: if there is an equation of the form $X_k = a$, then X_k cannot be the lhs of a 'product'-equation (since the problem $\mathcal{P}$ is assumed pruned).

Definition 4 *Let t be any given lt-dag. For any $i = 1..n$, the* **set value** *deduced for X_i from the labels of t is the set of ground terms: $\{t_w \mid w$ node on t such that: $l_i^w = 1\}$.*

Definition 5 *An lt-dag t is said to have the* 'closure property', *(or is said to be a* closed *lt*-dag*), iff the following condition holds:*

- *Suppose $i, j, k \in \{1..n\}$ are any 3 indices such that t has two nodes u, v with $l_i^u = 1 = l_j^v$,* and *$\mathcal{P}$ contains a 'product'-equation $X_k = X_i * X_j$; then there is a node w on t such that $l_k^w = 1$, with subterm $(t_u * t_v)$.*

Remark 1. It seems unlikely that a 'global' condition such as the above closure property can be checked by an automaton-based approach; thus, an *lt*-dag t accepted by the automaton to be associated below to the *ACID*-problem $\mathcal{P}$, may *not* be closed; and so, the solution that we shall derive from an accepted t for the problem $\mathcal{P}$, will *not* necessarily be the set values deduced from the labels of t.

6.2 The States of the LDA with their Defect Sets

The LDA that we shall be associating with our *ACID*-problem is somewhat similar to the tree automata with free variables defined in [11]; the differences come from the fact that we are trying to solve for the X_i in terms of *finite, non-empty* sets. Recall that n denotes the number of set variables of the given *ACID*-problem. Let $\mathbf{S}$ be the set of all $2n$-bit vectors of the form $\{..., l_i, ...; ..., h_i, ...\}$ such that $l_i \leq h_i$ for all $i \in 1..n$; the elements of $\mathbf{S}$ will be referred to as **pstates** ('p' stands for 'preliminary'); we shall denote them by barred capital letters such as $\overline{A}, \overline{B}, \ldots$. For any pstate $\overline{A} \in \mathbf{S}$, we shall also refer to its first and second half n-bit vectors as its lower and upper half, and denote them by $\overline{A}.l, \overline{A}.h$ respectively.

Any state A of our LDA will have two components: its first component is a pstate, i.e. an $2n$-bit vector, that we denote $\overline{A}$; this corresponds to two sets of boolean valuations on the set expressions $\{X_i\}_{i=1..n}$; the 'lower i-th bit' $\overline{A}.l_i$ of the state A on the LDA will signify (when 1) that the term *at* the current node on an *lt*-dag mapped to A under a given run r is accepted as element of X_i; the 'upper i-th bit' $\overline{A}.h_i$ is meant to signify (when 1) that some subterm *below* the current node has been accepted in X_i by the run; this explains why we only consider pstates $\overline{A}$ with $\overline{A}.l \leq \overline{A}.h$.

An accepting state on the LDA will then be in particular such that $\overline{A}.h_i = 1$ for all $i = 1..n$; but as we shall see below, this is *not* a sufficient condition for acceptance: many of the 'product'-equations of $\mathcal{P}$ may still remain unsatisfied. To circumvent this, we add as a second component to any state A of the LDA a set of equalities formed from new symbols, giving information on the 'products of terms accepted below the current node, that still have to be covered'. This second component at A will be referred as the *defect set* at A; for defining it formally, we need a few preliminaries.

Definition 6 *i) A triple (k, i, j) of indices in $1..n$ is said to be* conjugate *w.r.t. the ACID-problem $\mathcal{P}$ iff there is an equation of the form $X_k = X_i * X_j$ in the problem $\mathcal{P}$; a pair of indices (i, j) is said to be conjugate iff they are the second and third components of some conjugate triple.*

ii) A label l is said to be conjugate to a label m w.r.t. an equation $X_k = X_i * X_j$ of the ACID-problem $\mathbf{P}$ *if and only if: $l_i = 1 = m_j$.*

iii) A pstate $\overline{A}$ is said to be 'unmarked' *iff the following holds: whenever there is an equation $X_k = X_i * X_j$ in the problem $\mathcal{P}$, we have $\overline{A}.l_i = 0 = \overline{A}.l_j$.*

Next we introduce new symbols $X_{\overline{A}}^{i}$ for every $i = 1..n$, and every pstate $\overline{A} \in \mathbf{S}$; and also an additional new symbol $\mathbf{T}$ (signifying '*to be covered*'); these symbols will be referred to as **dsymbols** (the 'd' stands for 'discriminating'). Two dsymbols $X_{\overline{A}}^{i}, X_{\overline{B}}^{j}$ are said to be *conjugate* iff the pair of indices (i, j) is conjugate. A *defect equality* over the dsymbols is, *by definition*, an equality having one of the two following forms:

$$X_{\overline{C}}^{k} = X_{\overline{A}}^{i} X_{\overline{B}}^{j}, \qquad \mathbf{T} = X_{\overline{A}}^{i} X_{\overline{B}}^{j}$$

where $\overline{A}, \overline{B}, \overline{C} \in \mathbf{S}$, and (k, i, j) is a conjugate triple. Equalities of the first type are said to be '*closed*', and those of the second type '*open*'. Note that the number of all such equalities is polynomial in the size of $\mathbf{S}$.

Definition 7 *(i) A* defect set *is any finite (possibly empty) set* $\mathbf{M}$ *containing dsymbols and defect equalities. Such a set* $\mathbf{M}$ *is said to be* closed *iff:* $\mathbf{M}$ *contains no open defect equalities, and for any two dsymbols* $X_{\overline{A}}^{i}, X_{\overline{B}}^{j}$ *in* $\mathbf{M}$ *which are conjugate, there is a closed equality in* $\mathbf{M}$ *with* $X_{\overline{A}}^{i} X_{\overline{B}}^{j}$ *as its rhs. The empty defect set is closed, by definition.*

(ii) A state *A of the LDA is a pair* $(\overline{A}, \mathbf{M}_A)$ *such that* $\overline{A}$ *is a pstate, and* $\mathbf{M}_A$ *is a defect set;* $\overline{A}$ *is called the first component of A, and the defect set* $\mathbf{M}_A$ *its second.*

(iii) A state $(\overline{A}, \mathbf{M}_A)$ *is said to be* **closed** *iff: the pstate* $\overline{A}$ *is* unmarked*,* $\overline{A}.h_i = 1$ *for all i,* and *the defect set* $\mathbf{M}_A$ *is closed in the sense defined in (i) above.*

(iv) To any label $m = (m_1, ..., m_n)$ *we associate a state* $(\overline{C}_m, \mathbf{M}_m)$*, such that:*
- $\overline{C}_m$ *is the (unique) pstate with* $\overline{C}_m.l = m = \overline{C}_m.h$*;*
- *the dsymbols in* $\mathbf{M}_m$ *are the* $X_{\overline{C}_m}^{p}$*, for all* $p \in \{1, .., n\}$ *such that* $\overline{C}_m.l_p = 1$*;*
- *the defect equalities in* $\mathbf{M}_m$ *are of the form* $\mathbf{T} = X_{\overline{C}_m}^{i} X_{\overline{C}_m}^{j}$*, taken over* all *conjugate pairs of indices* (i, j) *such that* $\overline{C}_m.l_i = 1 = \overline{C}_m.l_j$*.*

For example, let $X_3 = X_1 * X_2$ be the given *ACID*-problem, and $\{X_1, X_2, X_3\}$ its variable list. If $m = (110)$ is the given label, then the associated state is such that: $\overline{C}_m = \overline{C} = (110; 110)$ and $\mathbf{M}_m$ is the defect set $\{X_{\overline{C}}^{1}, X_{\overline{C}}^{2}, \ \mathbf{T} = X_{\overline{C}}^{1} X_{\overline{C}}^{2}\}$; this state is *not* closed. States associated to labels will be those reached under initialization steps for the runs of the LDA that we shall be associating with the given *ACID*-problem.

6.2.1 Construction of the LDA associated: The Details

As previously, $\mathbf{P}$ denotes the *ACID*-unification problem given in (standard) pruned form. In the following definition, for any defect set $\mathbf{M}$ we shall denote by $\mathbf{M}^{(s)}$ (resp. by $\mathbf{M}^{(e)}$) the set of all dsymbols (resp. defect equalities) elements of $\mathbf{M}$; $\mathbf{M}$ is thus the disjoint union of its subsets $\mathbf{M}^{(s)}$ and $\mathbf{M}^{(e)}$.

Definition 8 *The LDA associated to* $\mathcal{P}$ *is the quintuple* $(\Sigma, \mathbf{Q}, \mathbf{F}, \Delta, \mathbf{L})$*, where:*

- Σ *is the signature of the problem* $\mathbf{P}$*; and the set of states* $\mathbf{Q}$ *is the set of all pairs* $A = (\overline{A}, \mathbf{M}_A)$ *of* pstates and defect sets *such that the following holds:*
  - $\overline{A}.l_k = \overline{A}.l_i \vee \overline{A}.l_j$ *for* every *sum-equation* $X_k = X_i + X_j$ *in the problem* $\mathbf{P}$*;*
- *The set* $\mathbf{F}$ *of* **accepting states** *is that of all* closed *states in* $\mathbf{Q}$ *(cf. Definition 7).*
- *The set of labels* $\mathbf{L}$ *for the LDA is the set of all n-bit vectors* $(l_1, l_2, \dots, l_n)$*.*

- **(initialization)** *For any constant* $a \in \Sigma$ *and any label m* initial *w.r.t. a, there are transitions in* Δ *of the form:* $a \xrightarrow{m} (\overline{C}_m, \mathbf{M}_m)$ *(cf. Definition 7).*

- **(progression)** *For any* $A, B, C \in \mathbf{Q}$ *and label* $m = (m_1, m_2, \ldots, m_n)$*, there is a transition rule in* Δ *with label* m *of the form* $A * B \xrightarrow{m} C$*,* if and only if*:*
  (1) *Conditions on the pstates at* A, B, C*: (*'Coherence of labels*')*
    - $m = \overline{C}.l$ *(*'transition's label must be the one at the node reached*');*
    - *If* $p \in 1..n$ *is such that there is some 'constant'-equation* $X_p = a$ *in* $\mathcal{P}$*, then* $\overline{C}.l_p = 0$*;*
    - *for* every *equation* $X_k = X_i * X_j$ *in* $\mathbf{P}$*, we have:* $\overline{C}.l_k = \overline{A}.l_i \wedge \overline{B}.l_j$*;*
    - $\overline{C}.h_i = \overline{C}.l_i \vee (\overline{A}.h_i \vee \overline{B}.h_i)$*, for* all $i = 1..n$.

  (2) *Conditions on the defect sets* $\mathbf{M}_A, \mathbf{M}_B, \mathbf{M}_C$*:*
    - $\mathbf{M}_C^{(s)} = \mathbf{M}_A^{(s)} \cup \mathbf{M}_B^{(s)} \cup \{X_{\overline{C}}^{j} \mid \overline{C}.l_j = 1\}$*;*
    - $\mathbf{M}_C^{(e)}$ is the set $\mathbf{M}'$ obtained as follows, from $\mathbf{M}_A^{(e)}, \mathbf{M}_B^{(e)}$, and $\mathbf{M}_C^{(s)}$*:*

      - Start with $\mathbf{M}' = \emptyset$*; put into* $\mathbf{M}'$ *every* closed *defect equality from* $\mathbf{M}_A^{(e)}$ *(resp. from* $\mathbf{M}_B^{(e)}$*) for which* $\mathbf{M}_B^{(e)}$ *(resp.* $\mathbf{M}_A^{(e)}$*) does* not contain *an* open equality with the same 'rhs' (right-hand-side).

      - Add to $\mathbf{M}'$ *all the closed equalities of the form* $X_{\overline{C}}^{k} = X_{\overline{A}}^{i} X_{\overline{B}}^{j}$ *for all conjugate triples* (k, i, j) *with* $\overline{A}.l_i = \overline{B}.l_j = \overline{C}.l_k = 1$*.*

      - Eliminate from $\mathbf{M}'$ *the (closed) defect equalities whose* rhs *contains a dsymbol indexed by the current pstate* $\overline{C}$*.*

      - For every conjugate pair $X_{\overline{A'}}^{i}, X_{\overline{B'}}^{j}$ *of dsymbols in* $\mathbf{M}_C^{(s)}$ not forming the rhs of any closed equality in $\mathbf{M}'$*, add* $\mathbf{T} = X_{\overline{A'}}^{i} X_{\overline{B'}}^{j}$ *to* $\mathbf{M}'$*.*

A *run* of the LDA on an *lt*-dag is defined as in the case of DA's, subject to the above initialization and progression conditions; we also assume (as we may) that any *run* r *is coherent in the sense that*: for any node w on t, we have $\overline{r(w)}.l = l^w =$ label of t at w.

Remarks 2. i) If the defect set $\mathbf{M}_T$ at state T contains a dsymbol $X_{\overline{A}}^{i}$, the semantics is that: some node at or below the current node got mapped to a state whose pstate is $\overline{A}$, and the subterm there got accepted in the set X_i.

ii) The condition that the left-half n-bit vector at an accepting state is 'unmarked' has the following semantics: Let C be a state image of a node w under a run, and suppose $\overline{C}.h = 1$; suppose there is an equation $X_k = X_i * X_j$ in the problem $\mathcal{P}$, and suppose we also have $\overline{C}.l_i = 1$; then t_w is accepted in X_i by the run at w; on the other hand, since $\overline{C}.h_j = 1$ there is a node v below w such that t_v got accepted in X_j; then (for the *ACID*-equation to be satisfied) the set X_k must contain the superterm $t_w * t_v$; which cannot be accepted at the node w, so C cannot be an accepting state.

iii) The progression conditions (2) have the following semantics: The open defect equalities of $\mathbf{M}_C$ in the definition represent the products which have gone uncovered at or below C; such an open equality can come either from $\mathbf{M}_A$ or from $\mathbf{M}_B$, or get created at C because the term accepted there contributes to a product.

In particular, If $(A, B) \longrightarrow C$ is a transition and (k, i, j) is a conjugate triple of indices such that $\overline{A}.l_i = \overline{B}.l_j = 1 = \overline{C}.l_k$, the defect set $\mathbf{M}_C$ at C may *not necessarily* contain the closed equality $X_{\overline{C}}^{k} = X_{\overline{A}}^{i} X_{\overline{B}}^{j}$ (cf. Examples 4, 6 below).

iv) It is important to note that our LDA is *deterministic*: Given two states A and B and a label m, there is a unique transition with label m from A, B to a state C. (This fact will be used in proving the completeness of our approach, cf. Section 4.1.)

Example 2. The *ACID*-unification problem: $X + Z = X * Y + U$ may be transformed into the following standard (pruned) form:

$$V = X + Z, \quad V = W + U, \quad W = X * Y$$

Let us show that the lt-dag in Figure 3 is accepted by the associated LDA. Arrange the set variables into an ordered list, say as $\{X, Y, Z, U, V, W\}$.

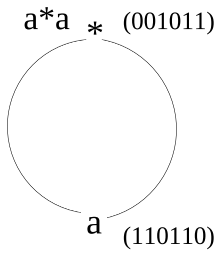

Figure 3: lt-dag solving $V = X + Z,\ V = W + U,\ W = X * Y$

The states on the LDA are therefore 12-bit vectors. From the node 'a' on the lt-dag with label $m = (110110)$ we first have an initial transition to a state A:

$$a \xrightarrow{110110} A = (110110\,; 110110\,; \mathbf{M}_A)$$

where $\mathbf{M}_A = \{X_{\overline{A}}, Y_{\overline{A}}, U_{\overline{A}}, V_{\overline{A}}, \mathbf{T} = X_{\overline{A}}Y_{\overline{A}}\}$ representing the fact that at the current node the values $X = Y = a = V = U$ have been accepted (and a product $X * Y$ remains to be covered). Next we have a transition, with label (001011), from $A * A$ to the state $B = (001011; 111111; \mathbf{M}_B)$, where $\mathbf{M}_B = \{X_{\overline{A}}, Y_{\overline{A}}, U_{\overline{A}}, V_{\overline{A}}, Z_{\overline{B}}, V_{\overline{B}}, W_{\overline{B}}, W_{\overline{B}} = X_{\overline{A}}Y_{\overline{A}}\}$; at this node, the assignments $W = a * a = Z = V$ have been accepted. The state reached at the root node is an accepting state. So the lt-dag is accepted. $\square$

In Example 2 above, the set values deduced from the labels of the accepted lt-dag *is* a solution to the unification problem. We shall see below that this may not be true in general. (It happened to be true above because the lt-dag was itself closed.) The following result throws some light on the connection between the closure property of an lt-dag, and the defect set at the state image of its root node. It will be used in proving the completeness of our LDA approach (cf. section 5.5).

Proposition 1 *Let r be a run on an lt-dag, and u any node on t such that the state $r(u)$ on the LDA associated to $\mathcal{P}$ is* not *closed, i.e. to say contains an open defect equality of the form $X^i_{\overline{A'}} X^j_{\overline{B'}}$. Then there exist two nodes u', v' on t (at or) below u, satisfying the following conditions:*

- *The labels at u', v' are non-null and conjugate; and $(\overline{r(u')}, \overline{r(v')}) = (\overline{A'}, \overline{B'})$.*
- *No node (at or) below u on t has as subterm the product term $t_{u'} * t_{v'}$.*

Proof: Let $C = r(u)$. We reason by induction on the height of the node u on t. If u is of height 0, there is nothing to prove. Similarly, if the open defect equality of the hypothesis 'gets created' at C, then it if of the form $\mathbf{T} = X^i_{\overline{C}} X^j_{\overline{D}}$ (or: $\mathbf{T} = X^j_{\overline{D}} X^i_{\overline{C}}$) for some state D image under the run r of some node w below u; in this case we can take $u' = u,\ v' = w$ (resp. $u' = w,\ v' = u$), to satisfy the assertions of our proposition.

We thus assume that node u is of height ≥ 1, of the form $u = u_1 * u_2$, where u_1, u_2 are nodes of smaller height than u on t; and that the open defect equality of the hypothesis does *not get created at* $C = r(u)$; this implies in particular:

(i) $\qquad \overline{A'} \neq \overline{C},\ \overline{B'} \neq \overline{C}.$

Let $A_1 = r(u_1), A_2 = r(u_2)$; so the open defect equality of $\mathbf{M}_C$ of the hypothesis comes from $\mathbf{M}_{A_1}$ (or $\mathbf{M}_{A_2}$, or both), and is *not* replaced by a closed defect equality under the transition from $(A_1 * A_2)$ to C; then the following must hold:

(ii) $\qquad (\overline{A'}, \overline{B'}) \quad \neq \quad (\overline{A_1}, \overline{A_2}).$

(*Otherwise* we would have by (i) above: $\overline{A_1} \neq \overline{C},\ \overline{A_2} \neq \overline{C}$; so by the progression condition (2) the open defect equality under study – by assumption not created at C – would have been replaced by a closed one of the form: $X^k_{\overline{C}} = X^i_{\overline{A_1}} X^j_{\overline{A_2}}$.)

Suppose the open defect equality under study comes e.g. from $\mathbf{M}_{A_1}$. Then, by induction hypothesis, there exist nodes u', v' *at or below* u_1 such that $(\overline{r(u')}, \overline{r(v')}) = (\overline{A'}, \overline{B'})$, and such that *no node at or below* u_1 *on* t has as subterm the product term $t_{u'} * t_{v'}$. There remains a priori the possibility that this product term is the subterm of t at some node at or below u, but above u_1; but this would imply $u' = u_1, v' = u_2$, which is ruled out by (ii). $\square$

We now show that an lt-dag accepted by the LDA may *not* necessarily be a closed lt-dag, i.e. may not have the closure property (cf. Definition 5).

Example 3. Consider the $ACID$-problem: $Z =^? X * Y$, and the solution set $X = \{a, c\}$, $Y = \{b, d\}$. The following run is accepting on the lt-dag rooted at the term $(a*b)*(c*d)$, with nodes at the subterms $a, b, c, d, (a*b), (c*d)$ respectively labeled $(100), (010), (100), (010), (001), (001)$, but which has no node with $(c*b)$ or $(a*d)$ as subterm; this lt-dag is *not* closed):

$$a \xrightarrow{100} A = (100; 100; \{X_{\overline{A}}\}) \qquad c \xrightarrow{100} A = (100; 100; \{X_{\overline{A}}\})$$
$$b \xrightarrow{010} B = (010; 010; \{Y_{\overline{B}}\}) \qquad d \xrightarrow{010} B = (010; 010; \{Y_{\overline{B}}\})$$
$$A * B \xrightarrow{001} C = (001; 111; \{X_{\overline{A}}, Y_{\overline{B}}, Z_{\overline{C}},\ Z_{\overline{C}} = X_{\overline{A}} Y_{\overline{B}}\})$$
$$C * C \xrightarrow{000} D = (000; 111; \mathbf{M}_D = \mathbf{M}_C\})$$
$\square$

This example shows that if a solution to the $ACID$-problem is to be recovered from an accepted lt-dag, *in general it is not* the set values deduced from its labels. But it also gives a clue to what can be done to make the LDA approach work. The idea is as follows: instead of looking individually at the ground constants, we may consider them as 'equivalent' when they get mapped to a same state. For instance in Example 3, we may consider a, c as equivalent, and represent their class by $\overline{A}$; and similarly represent b and d with $\overline{B}$. From the accepting run in the above example, we can then derive as solution to the given unification problem, the assignment: $\{X = \overline{A},\ Y = \overline{B},\ Z = \overline{A} * \overline{B}\}$, where the value for the set Z is deduced from the closed defect equality $Z_{\overline{C}} = X_{\overline{A}} Y_{\overline{B}}$ at D (image of the root node of the lt-dag). We formalize such an idea in the coming section, by associating a grammar to any accepting run.

6.3 Grammar for Deriving a Solution from Accepting Run

Throughout this section, we consider a given run r of the LDA associated to the given $ACID$-problem $\mathcal{P}$ on a given lt-dag t, mapping t to an accepting state. From the run r we propose to build a grammar $\mathbf{G} = \mathbf{G}(r)$ that will allow us to derive a solution to our unification problem. The non-terminals of our grammar are the pstate symbols of the LDA; and its terminals are the ground constants appearing in $\mathcal{P}$.

Definition 9 *Let w be any given node on t such that $r(w) = E$ is an accepting state.*

i) For every closed defect equality in $\mathbf{M}_E$ of the form $X^k_{\overline{C}} = X^i_{\overline{A}} X^j_{\overline{B}}$, add a 'product'-production *rule $\overline{C} \longrightarrow \overline{A} * \overline{B}$ to $\mathbf{G}$.*

ii) Let u be any node on t below w, and $r(u) = B$; if $\overline{B}.l \neq 0$ and $\overline{B}$ is not the lhs of any *'product'-production, then add to $\mathbf{G}$ the production rule $\overline{B} \longrightarrow t_u$.*

iii) For any $i \in 1..n$, the contribution $\mathbf{C}_i(w)$ of the run r *to any set variable X_i, at the node w, is the set of all terms over the ground constants, that are frontiers of derivation trees from non-terminals $\overline{C}$ in $\mathbf{G}$, for which $X^i_{\overline{C}} \in \mathbf{M}_E$, i.e. such that $\overline{C}.l_i = 1$.*

We proceed now to prove, successively, that: (i) the sets $\mathbf{C}_i(w), i = 1..n$, are all *non-empty*, (ii) the assignments $X_i = \mathbf{C}_i(w)$ satisfy the set constraints problem associated to $\mathcal{P}$; and (iii) the $\mathbf{C}_i(w), i = 1..n$, are all *finite* sets.

Proposition 2 *The $\mathbf{C}_i(w)$ as defined above are* non-empty *sets of terms over the ground constants and '$*$', for all i.*

Proof: (We assume that w is the root node of t.) By assumption $r(w) = E$ is closed, so for every $i = 1..n$, we have $\overline{E}.h_i = 1$, so there is at least one state reached by r at some node u on t such that $\overline{r(u)}.l_i = 1$. Let $p \in 1..n$ be given, and u any node on t such that $\overline{r(u)}.l_p = 1$. Our claim is that the subterm t_u of t at u is in $\mathbf{C}_p(w)$, and is produced by the non-terminal $\overline{r(u)}$ of $\mathbf{G}$; we prove this by induction on the height of the node u on t.

Let $r(u) = C$. If u is of height 0, then t_u is a ground constant; the state C is then reached under initialization of r, so $\overline{C}.l$ must be an initial label (cf. Definition 2); therefore $\overline{C}$ cannot be the lhs of any 'product'-production, and $\mathbf{G} = \mathbf{G}(r)$ must contain the production $\overline{C} \rightarrow t_u$, and our claim is trivial in this case.

Assume then that u is of height > 0; as above, we may assume that $\overline{C}$ is the lhs of some 'product'-production in the grammar $\mathbf{G}$; this must correspond to some 'product'-equation $X_k = X_i * X_j$ in the problem $\mathcal{P}$ such that $\overline{C}.l_k = 1$. Now $u = u' * u''$ for two sub-nodes u', u'' of smaller height; let $r(u') = A, r(u'') = B$; then we must have $\overline{A}.l_i = 1 = \overline{B}.l_j$. Then the grammar $\mathbf{G}$ must contain the 'product'-production $\overline{C} \rightarrow \overline{A} * \overline{B}$; now, by inductive assumption, the subterm $t_{u'}$ (resp. $t_{u''}$) is produced by $\overline{A}$ (resp. by $\overline{B}$), and is in $\mathbf{C}_i(w)$ (resp. in $\mathbf{C}_j(w)$). It follows that the term $t_u = t_{u'} * t_{u''}$ is produced by $\overline{C}$, and is in $\mathbf{C}_k(w)$ of course, but also in $\mathbf{C}_p(w)$ by definition, since $\overline{C}.l_p = 1$. □

Proposition 3 *Let r be any run on an lt-dag t, and w any node on t such that $r(w) = E$ is an* accepting state *on the LDA. Then the assignments $X_i = \mathbf{C}_i(w), i = 1..n$, satisfy the set constraints associated to the ACID-problem.*

Proof: We suppose that w is the root node of t; and show that every equation in the $ACID$-problem is satisfied under the set assignments $X_i = \mathbf{C}_i(w)$.

Step i): The equations of the form $X = a$ in $\mathcal{P}$ are all satisfied.

Let i be given such that we have an equation $X_i = a$ in $\mathcal{P}$. Then there is a state A reached under the run r at initialization; then the production $\overline{A} \longrightarrow a$ in $\mathbf{G}(r)$ contributes a to X_i. We have to check that no other productions of $\mathbf{G}$ contribute any terms to the set X_i. These may be:

- either of the form $\overline{B} \longrightarrow b$ for $B \neq A$, and $b \neq a$: but these cannot contribute to the given X_i by our definition of initial labels (cf. Definition 2), and by the coherence of labels under transitions (cf. Definition 7);

- or 'product'-productions in $\mathbf{G}$ of the form $\overline{R} \rightarrow \overline{P} * \overline{Q}$ corresponding to 'product'-equations in $\mathcal{P}$; such productions cannot have $\overline{R}.l_i = 1$ again by the coherence of labels under transitions; and so, cannot contribute to the X_i considered.

Step ii): The 'sum'-equations in $\mathcal{P}$ of the form $X_k = X_i + X_j$ are all satisfied.

A term contributed to X_k is derived from a non-terminal $\overline{C}$, corresponding to a state C reached by the run such that $\overline{C}.l_k = 1$. But $\overline{C}.l_k = 1$ iff $\overline{C}.l_i \vee \overline{C}.l_j = 1$; so either or both, of the dsymbols $X^i_{\overline{C}}, X^j_{\overline{C}}$ must be in $\mathbf{M}_E$. So any term in the contribution $\mathbf{C}_k(w)$ is either in $\mathbf{C}_i(w)$, or in $\mathbf{C}_j(w)$, or both. So we deduce: $\mathbf{C}_k(w) = \mathbf{C}_i(w) \cup \mathbf{C}_j(w)$.

Step iii): The 'product'-equations in $\mathcal{P}$, of the form $X_k = X_i * X_j$ are all satisfied.

We first prove that the contribution under r to the set X_k is contained in the product of those to the sets X_i and X_j.

Such a variable X_k is not $\succ_*$-minimal; so by definition, any term in $\mathbf{C}_k(w)$ is derived from a $\overline{C}$, for some state reached C such that $\overline{C}.l_k = 1$, and for which we have closed defect equalities of the form $X^k_{\overline{C}} = X^i_{\overline{A}} X^j_{\overline{B}}$ in $\mathbf{M}_E$; the presence of such a closed defect equality implies that there has been a transition to state C from two other states A, B also reached under r, with $X^i_{\overline{A}}, X^j_{\overline{B}} \in \mathbf{M}_E$, that is to say we have $\overline{A}.l_i = 1 = \overline{B}.l_j = 1$; and this holds whatever be the pair of states (A, B) from which the run has transited to C. By definition, one deduces then that any term in the contribution $\mathbf{C}_k(w)$ of the run to X_k is the product of a term in the set

$\mathbf{C}_i(w)$ with a term in the set $\mathbf{C}_j(w)$. This proves the inclusion $\mathbf{C}_k(u) \subset \mathbf{C}_i(u) * \mathbf{C}_j(u)$.

Conversely suppose that the contribution of r to the set X_i contains a term s' derived from some $\overline{A}$ such that $\overline{A}.l_i = 1$, and that to the set X_j contains a term s'' derived from some $\overline{B}$, such that $\overline{B}.l_j = 1$, where A, B are two states reached under r; the defect set $\mathbf{M}_E$ contains then the two conjugate dsymbols $X\frac{i}{A}, X\frac{j}{B}$. Now r being accepting, the defect set $\mathbf{M}_E$ at $E = r(w)$ must contain a closed defect equality of the form $X\frac{k}{C} = X\frac{i}{A} X\frac{j}{B}$ for some state C on the LDA for which $\overline{C}.l_k = 1$. The grammar $\mathbf{G}(r)$ contains then, by definition, a production $\overline{C} \longrightarrow \overline{A} * \overline{B}$; one deduces then that the contribution $\mathbf{C}_k(w)$ of the run to X_k contains the product term $s' * s''$. This proves the reverse inclusion $\mathbf{C}_i(w) * \mathbf{C}_j(w) \subset \mathbf{C}_k(w)$. □

Proposition 4 *The sets $\mathbf{C}_i(w)$ as defined above are all* finite*, for $i = 1..n$.*

Proof: (As above E is the image of the root node of t under the run.) Suppose there is an index $p \in 1..n$ such that $\mathbf{C}_p(w)$ is infinite. By our definition of the sets $\mathbf{C}_i(w)$, this is possible only if e.g. a situation of the following type arises:

i) there is a chain of 'product'-productions in $\mathbf{G}$, of the form:

$$\overline{C_1} \longrightarrow \overline{A_1} * \overline{B_1}, \quad \overline{C_2} \longrightarrow \overline{A_2} * \overline{B_2}, \ldots, \quad \overline{C_m} \longrightarrow \overline{A_m} * \overline{B_m}, \quad \text{such that:}$$

ii) for each $j, 2 \leq j \leq m$, we have $\overline{C_j} = \overline{A_{j-1}}$; and $\overline{C_1} = \overline{A_m}$.

If such a situation arises, then (with the above notation) we shall say that the non-terminal $\overline{C_1}$ of $\mathbf{G}(r)$ creates a cycle for $\mathbf{G}$. Let u be a node of *least height on* t such that the run r maps u to a state C_1, whose pstate $\overline{C_1}$ creates a cycle. There can be no 'product'-productions from the pstate of a state reached under initialization, so this node u cannot be a leaf-node on t. We infer then that $u = u_1 * v_1$ for two nodes of smaller height than u on t, such that: $\overline{r(u_1)} = \overline{A_1}$, $\overline{r(v_1)} = \overline{B_1}$. Now the 'product'-production from $\overline{C_1}$ by definition must correspond to a closed defect equality of the form $X\frac{k}{C_1} = X\frac{i}{A_1} X\frac{j}{B_1}$ in $\mathbf{M}_E$, which implies that the subterm t_u of t is accepted in X_k at the state C_1; it follows that t_{u_1} is accepted in X_i at the state $r(u_1)$, and t_{v_1} at $r(v_1)$. But then u_1 is a node of smaller height than u, and gets mapped to $r(u_1)$ whose pstate $\overline{A_1}$ also creates a cycle for $\mathbf{G}$ - contradiction. □

We deduce our principal claim by putting together the above three Propositions 3, 4 and 5:

Theorem 5 *The LDA-approach is sound:* if there is an lt-dag accepted by the LDA associated to a given $ACID$-unification problem $\mathcal{P}$, then $\mathcal{P}$ is solvable.

Remark 3. We saw already that an lt-dag t accepted by the LDA associated a given problem $\mathcal{P}$ may not have the closure property, so the set values derived from its labels cannot be a solution to $\mathcal{P}$. The role of the grammar $\mathbf{G}(r)$ derived from an accepting run r on t, is precisely to help us construct an lt-dag t' (from the labels of t and the run), such that t' has the closure property and is a solution to $\mathcal{P}$.

6.4 The LDA-approach is Complete

Theorem 6 *Suppose given an ACID-unification problem $\mathcal{P}$ in pruned form, and let $\mathbf{A}$ be the associated LDA. Suppose the problem $\mathcal{P}$ solvable, and assume given a solution to $\mathcal{P}$ as sets of terms over the ground constants and '$*$'. Then we can construct a closed lt-dag t that is accepted by $\mathbf{A}$, and such that the given solution is exactly the set values deduced from the labels of t.*

Proof: Let $X_i, i = 1..n$, denote as usual the variables of the problem $\mathcal{P}$ given in pruned form, and assume given a solution to $\mathcal{P}$ in terms of finite non-empty sets of terms (that we also denote by X_i). From the given solution it is not difficult to construct, in a natural and unique manner a set $\mathbf{D}$ of lt-dags, which share all their common nodes and are such that: for every i,

X_i is the set of subterms at the nodes on $\mathbf{D}$ where the i-th bit of the label is 1. We can then complete this construction if necessary, (in a non unique manner) to obtain one 'global' lt-dag t such that: every lt-dag in $\mathbf{D}$ is a sub-lt-dag of t, and the label of t is null at any node where the subterm is not in the solution set given, in particular at nodes of t strictly above those of $\mathbf{D}$ (e.g. see Example 5 in the Appendix II).

Since t has been constructed from a solution for the set variables in $\mathcal{P}$, t is *necessarily a closed lt-dag*; that is to say: if u, v are any two nodes on t with conjugate labels, then t contains a node w such that $t_w = t_u * t_v$. Actually the set of all nodes of $\mathbf{D}$ must already be 'closed' in this sense, for the same reason.

Remains then to construct an accepting run of the LDA associated to $\mathcal{P}$, on this lt-dag t. Now recall that our LDA is deterministic; so the transitions of the run looked for are uniquely determined at every node of $\mathbf{D}$, by its label. (Note: because of the determinism, this is true also for the nodes 'under $\mathbf{D}$' with label 0, i.e. where the subterm is not in the given solution set.) On the other hand, the labels at the nodes of maximal height in $\mathbf{D}$ – which correspond to the terms of *maximal* height in the given solution of $\mathcal{P}$ – have to be unmarked (cf. Definition 6: otherwise the solution given for $\mathcal{P}$ would contain product terms situated at nodes *strictly above* them.

Once this part of the run is constructed at all the nodes of the solution set $\mathbf{D}$, completing it to cover the nodes of t which are not in $\mathbf{D}$ is straightforward: these will again be transitions to nodes with a null label.

Remains finally to show that the run constructed in this manner is accepting on the 'global' lt-dag; i.e. to say, that the defect equalities are all closed at the state image of the root node under our run. But this follows from Proposition 2: indeed, if $u_i, i = 1, 2$ are any two nodes with non-null (and conjugate) labels, then the u_i must be nodes on $\mathbf{D}$ where our run is defined; so there is also a node on $\mathbf{D}$ where the label is non-null and the subterm is the product $t_{u_1} * t_{u_2}$; this node must therefore be at or below the root node of 'global' lt-dag constructed; by Proposition 7, we conclude that the image of this root node cannot contain any open defect equalities. □

Theorem 7 *ACID-unification and ACUID-unification are NEXPTIME-decidable.*

Proof: We first consider the theory $ACID$, and assume that the $ACID$-unification problem is in standard, pruned, form. We saw that such a problem admits a solution iff there is an accepting (coherent) run of the LDA associated to the problem, on an lt-dag. Now the LDA is of size exponential w.r.t. the number of variables in the problem; on the other hand we saw that finding an lt-dag accepted by this LDA is decidable in time NP w.r.t. the number of states of the LDA. This proves the assertion of NEXPTIME-decidability for $ACID$-unification.

That of $ACUID$-unification can then be deduced as follows: Choose non-deterministically the set of variables which are assigned the value $U = 0$; and solve for the others using the $ACID$-algorithm just described. □

Appendix II: Examples of $ACID$-Unification

Example 2. (revisited) For the $ACID$-unification problem, given in standard (pruned) form: $V = X + Z, \quad V = W + U, \quad W = X * Y$, we obtained above an accepting run on an lt-dag t rooted at $(a * a)$, for which the set values deduced from the labels of t is a solution to the $ACID$-problem; here the run is injective and the lt-dag t is itself closed (i.e. has the closure property).

Let us look at the solution given by the grammatical approach that we developed above. There are two productions in the grammar of the run: $\overline{B} \longrightarrow \overline{A} * \overline{A}, \;\; \overline{A} \longrightarrow a$. The contribution of the run to the variables are therefore the sets of terms representable as: $X = \{\overline{A}\} = Y = U, \;\; V = \{\overline{A}, \; \overline{A} * \overline{A}\}, \;\; Z = W = \{\overline{A} * \overline{A}\}$; that is to say:

$$X = \{a\} = Y = U, \qquad V = \{a,\ a*a\}, \qquad Z = W = \{a*a\}$$

which is the same as the set values deduced from the labels of the *lt*-dag. □

Example 3 (revisited). We go back to the problem $Z =^? X * Y$, and the accepting run r of the LDA on the *lt*-dag rooted at the term $t = (a*b)*(c*d)$, with nodes at $a, b, c, d, (a*b), (c*d)$ as given previously. The grammar at the root node of t has the following productions: $\overline{C} \longrightarrow \overline{A}*\overline{B},\ \ \overline{A} \longrightarrow a \mid c,\ \ \overline{B} \longrightarrow b \mid d$. The contributions to the set variables at the root node give then the following assignment: $X = \{a, c\},\ Y = \{b, d\},\ Z = \{a*b, a*d, c*b, c*d\}$, which *is* a correct solution.

If we considered the accepting sub-run on the sub-dag of t rooted at $(a*b)$, we would have derived the (correct) solution: $X = \{a\},\ Y = \{b\},\ Z = \{a*b\}$; □

The more complex example below illustrates in detail how a run of an LDA associated to $ACID$-problems climbs up an *lt*-dag.

Example 4. We consider again the $ACID$-problem $Z =^? X * Y$, and the list of variables ordered as $\{X, Y, Z\}$. Let a, b, c, d be ground constants; we construct an accepted *lt*-dag rooted at $(c*(c*(a*b)))*(c*d)$, with leaves at a, c put into X; and at b, d put into Y; in addition the nodes at $(a*b), (c*(a*b))$ are put into Y; cf. figure below. From the labels of this accepted *lt*-dag we can only get the following terms in the set Z: $(a*b),\ c*(a*b),\ c*(c*(a*b)),\ (c*d)$, so this *lt*-dag is *not* closed.

The accepting run is as follows (in the figure, the states to which the nodes get mapped are indicated in italic capitals).

$$a, c \xrightarrow{\ 100\ } A = (100; 100; \{X_{\overline{A}}\}); \qquad b, d \xrightarrow{\ 010\ } B = (010; 010; \{Y_{\overline{B}}\});$$

$$A*B \xrightarrow{\ 011\ } C = (011; 111; \{X_{\overline{A}}, Y_{\overline{B}}, Y_{\overline{C}}, Z_{\overline{C}},\ Z_{\overline{C}} = X_{\overline{A}}Y_{\overline{B}},\ \mathbf{T} = X_{\overline{A}}Y_{\overline{C}}\});$$

$$A*B \xrightarrow{\ 001\ } D = (001; 111; \{X_{\overline{A}}, Y_{\overline{B}}, Z_{\overline{D}},\ Z_{\overline{D}} = X_{\overline{A}}Y_{\overline{B}}\});$$

$$A*C \xrightarrow{\ 011\ } C = (011; 111; \{X_{\overline{A}}, Y_{\overline{B}}, Y_{\overline{C}}, Z_{\overline{C}},\ Z_{\overline{C}} = X_{\overline{A}}Y_{\overline{B}},\ \mathbf{T} = X_{\overline{A}}Y_{\overline{C}}\});$$

(A word of explanation on this last transition: The open $\mathbf{T} = X_{\overline{A}}Y_{\overline{C}}$ in $\mathbf{M}_C$ does not get replaced here by a closed defect equality $Z_{\overline{C}} = X_{\overline{A}}Y_{\overline{C}}$, since the term accepted at C contributes to a product with a term in the set X.)

Then: $\qquad A*C \xrightarrow{\ 001\ } F = (001; 111; \mathbf{M}_F)$, where

$$\mathbf{M}_F = \{X_{\overline{A}}, Y_{\overline{B}}, Y_{\overline{C}}, Z_{\overline{C}},\ Z_{\overline{C}} = X_{\overline{A}}Y_{\overline{B}}, Z_{\overline{F}}, Z_{\overline{F}} = X_{\overline{A}}Y_{\overline{C}}\};$$

And finally: $\qquad F*D \xrightarrow{\ 001\ } G = (000; 111; \mathbf{M}_G)$, where

$$\mathbf{M}_G = \{X_{\overline{A}}, Y_{\overline{B}}, Y_{\overline{C}}, Z_{\overline{C}},\ Z_{\overline{C}} = X_{\overline{A}}Y_{\overline{B}},\ Z_{\overline{D}},\ Z_{\overline{D}} = X_{\overline{A}}Y_{\overline{B}}, Z_{\overline{F}}, Z_{\overline{F}} = X_{\overline{A}}Y_{\overline{C}}\};$$

where $\overline{D} = \overline{F}$. The states D, F, G are all accepting; so the sub-dags rooted at the nodes mapped to these states are accepted. The solution for the problem derived at the node mapped to D is the simplest; the assignment is: $X = \{\overline{A}\},\quad Y = \{\overline{B}\},\quad Z = \{\overline{A}*\overline{B}\}$. where $\overline{A} \longrightarrow a \mid c,\ \overline{B} \longrightarrow b \mid d$; this is a correct solution.

Let us compute the contribution of the run to X, Y, Z on the sub-dag rooted at node F, by looking at $\mathbf{M}_F$: The productions of the grammar w.r.t. this accepting sub-run are as follows:

$$\overline{F} \longrightarrow \overline{A}*\overline{C}, \quad \overline{C} \longrightarrow \overline{A}*\overline{B}, \quad \overline{A} \longrightarrow a \mid c, \quad \overline{B} \longrightarrow b \mid d.$$

So the contribution of the run is the assignment: $X = \{\overline{A}\},\quad Y = \{\overline{B},\ \overline{A}*\overline{B}\},\quad Z = \{\overline{A}*\overline{B},\ \overline{A}*(\overline{A}*\overline{B})\}$, where $\overline{A} \longrightarrow a \mid c,\ \overline{B} \longrightarrow b \mid d$; which is again a correct solution.

The grammar for the accepting run at the root node G has an additional production: $\overline{D} \longrightarrow \overline{A}*\overline{B}$, with $\overline{D} = \overline{F}$. The solution derived here is the same as at F: the terms derivable from this production are also derivable from $\overline{C}$. □

Example 5. `(5.1)` The following $ACID$-unification problem in pruned form

$$W = X*Y, \quad W = U+V, \quad U = U1*U2, \quad V = V1*V2,$$

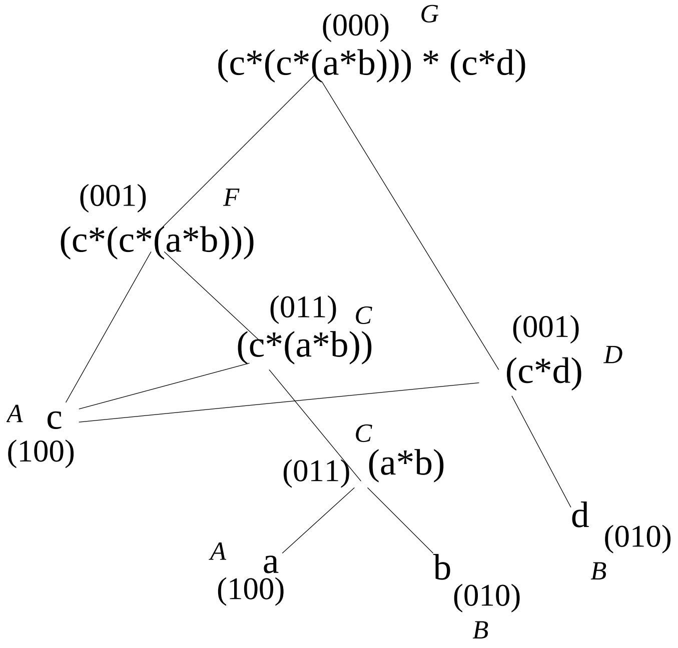

Figure 4: An accepted non-closed lt-dag for $Z \stackrel{?}{=} X * Y$

$$U1 = a, \quad U2 = b, \quad V1 = a, \quad V2 = c.$$

admits as a solution the substitution $X = \{a\}, Y = \{b, c\}$ on the end-variables X, Y. The lt-dag to the left of Figure 3 is constructed *from* this given solution; it is accepted by the LDA associated to the problem(the variables arranged as the ordered list $\{U1, U2, V1, V2, X, Y, U, V, W\}$).

(5.2) The $ACID$-unification problem in pruned form: $Z = U + Z, \quad Z = X * Y$, admits as a solution the substitution: $X = \{a\},\ Y = \{a, b\}, U = \{c\},\ Z = \{c, a * a, a * b\}$; the lt-dag to the right of the Figure 3, constructed *from* this solution, is accepted by the LDA associated (the ordered variable list is $\{U, X, Y, Z\}$).

The lt-dags with the 'full lines' in the figure correspond in both cases to what was denoted by **D** in the proof of Theorem 6. □

Our final example shows that reasoning with the $2n$-bit vectors (the first components) alone

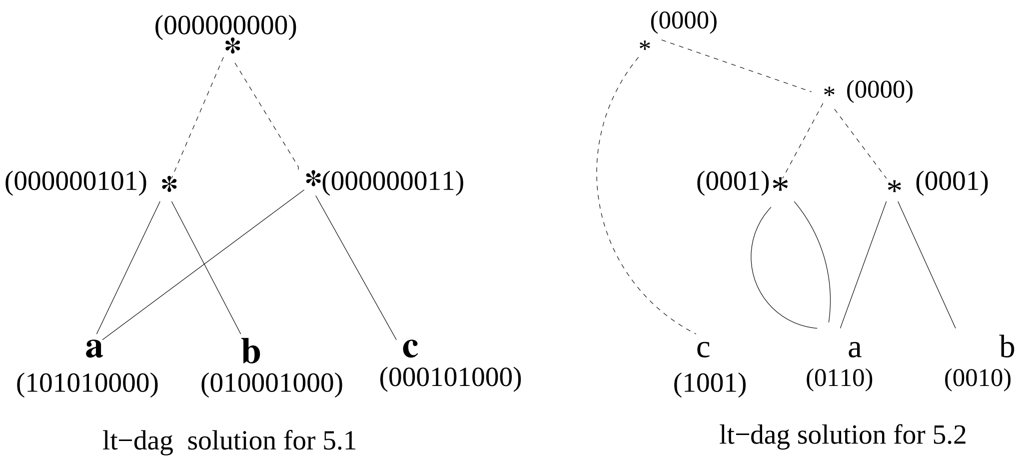

Figure 5: lt-dags constructed from given solutions: Example 5

at the LDA states would lead to incorrect conclusions.

Example 6. The following *ACID*-problem is unsolvable with *finite non-empty* sets:

$$X_1 = X_2 + X_3, \quad X_1 = X_4 + X_5, \quad X_1 = X_6 + X_7, \quad X_4 = X_2 * X_8, \quad X_6 = X_9 * X_3$$

Indeed Occur-Check returns 'Fail' on the problem: the variable X_1 is '$*$'-above X_2 as well as X_3 in the first 'sum'-equation. Let us try constructing a run of the associated LDA on an *lt*-dag, whose unique leaf is a, root node at $((a * a) * (a * a))$, and $(a * a)$ as the subterm at an intermediary node. The variables being arranged into a list under their natural order, this run goes as follows:

$a \xrightarrow{111010111} A = (111010111; 111010111, \mathbf{M}_A)$ where
$\mathbf{M}_A = \{X^1_A, X^2_A, X^3_A, X^5_A, X^7_A, X^8_A, X^9_A,\ \mathbf{T} = X^9_A X^3_A,\ \mathbf{T} = X^2_A X^8_A\})$
$A * A \xrightarrow{110101000} B = (110101000; 111111111; \mathbf{M}_B)$, where
$\mathbf{M}_B = \{X^1_A, X^2_A, X^3_A, X^5_A, X^7_A, X^8_A, X^9_A, X^1_B, X^2_B, X^4_B, X^6_B, X^6_B = X^9_A X^3_A,\ \mathbf{T} = X^2_B X^8_A\})$

And finally: $B * B \xrightarrow{000000000} C = (000000000; 111111111; \mathbf{M}_C = \mathbf{M}_B)$

The state C is *not* an accepting state: the defect set $\mathbf{M}_C$ at C is *not* closed; so the run is not accepting. What we want to point out here is that, *if we went only by the first components of the states* of the LDA, state C would have appeared as accepting: because $\overline{C}.l = 0$ is unmarked, and $\overline{C}.h = 1$. $\square$