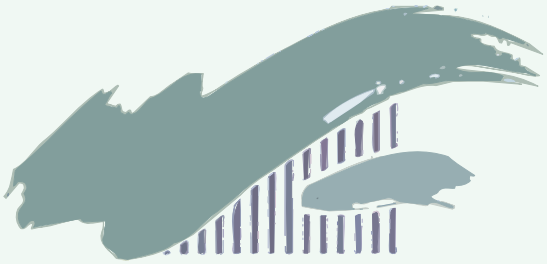

UNIVERSITE D'ORLEANS
Faculté des Sciences
LIFO

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/

$AC(U)ID$-Unification is NEXPTIME-Decidable

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

Rapport N^o **2003-02**

$AC(U)ID$-Unification is NEXPTIME-Decidable

Siva Anantharaman, Paliath Narendran*, Michael Rusinowitch

LIFO - Orlans (France), University at Albany–SUNY (USA), LORIA - Nancy (France)

Abstract. In an earlier work, we considered the theory $AC(U)ID_l$ obtained by adjoining a binary symbol '$*$' which is assumed left-distributive over a basic $AC(U)I$-operator '$+$'; we showed there that $AC(U)ID_l$-unification is NEXPTIME-decidable and DEXPTIME-hard. In this paper we consider unification over $AC(U)ID$, where the the binary symbol adjoined '$*$' is assumed *two*-sided distributive over the $AC(U)I$-operator '$+$'. We formulate the problem as one of solving a particular class of set constraints, and propose a method for solving it in terms of the language of a labeled dag automaton associated to the problem. $AC(U)ID$-unification is thus shown to be NEXPTIME-decidable; it is also shown to be DEXPTIME-hard via other considerations.

Keywords: E-Unification, Complexity, Set constraints, Tree automata, Dag automata, Labeled dag automata.

1 Introduction

In a recent work [3], we studied the unification problem over the theories that one obtains by adjoining finitely many homomorphisms, or a binary symbol '$*$' assumed one-sided distributive, over a basic $AC(U)I$-symbol '$+$' (AC for Associativity-Commutativity, I for Idempotence, U means Unit for '$+$' which is null for the homomorphisms and '$*$'). We showed in particular that the unification problem over the theory $AC(U)ID_l$ – obtained by adjoining a binary '$*$' assumed left-distributive over the $AC(U)I$-symbol '$+$' – is related to the unification problem over $AC(U)IH$ where we add finitely many *non-commuting* homomorphisms over $AC(U)I$. This latter unification problem is known to be DEXPTIME-complete (cf. [5]); we deduced therefrom that $AC(U)ID_l$-unification is NEXPTIME-decidable and DEXPTIME-hard.

When the binary symbol '$*$' added is assumed 2-sided distributive over the basic $AC(U)I$-symbol '$+$', we get the theories $ACUID$ or $ACID$, with or without a unit element U for '$+$' respectively. These theories are useful in program specification based on set constraints, so it is natural to look at their unification problem. In [3] we saw that – when '$*$' does not satisfy other laws than distributivity over '$+$' – the $ACUID$-unification problem can be formulated as one of solving a particular class of set constraints *in terms of finite sets*; and under such a vision, $ACID$-unification reduces to solving the same set constraints problem in terms of *finite and non-empty* sets; the decidability of $AC(U)ID$-unification can then be deduced from the results of [11], although no complexity bound seems derivable this way. In the current paper, we show that solving the set constraints

* Research supported in part by NSF grant CCR-9712396 and ONR grant N00014-01-1-0430.

problem deduced from a given $ACID$-unification problem, amounts to showing the non-emptiness of the language of a labeled dag automaton that can be constructed naturally from the unification problem.

This paper is arranged as follows. We first consider $ACID$-unification, and formulate the problem in terms of set constraints, then do some pre-processing based on Occur-Check (referred to as 'pruning' subsequently), meant to eliminate syntactically some unsolvable cases; this constitutes section 2. Generalities on dag automata and their languages are given section 3, where we also introduce the notions of labeled term dag, and of labeled term dag automaton. We proceed then to associate a labeled term dag automaton to any $ACID$-unification problem given in standard form, with the objective that solving the latter is equivalent to showing that the former has a non-empty language; section 4 is devoted to the construction of this automaton. Each state of the automaton will have two components; its first component is a $2n$-bit vector, n being the number of set variables in the problem; the role of the left-half n-bit vector is to indicate the sets which contain the subterm of the dag mapped (under a run) to the current node of the automaton; the right-half n-bit vector indicates the sets containing some subterm at or below the current node; the second component of a state is a 'defect set' giving an indication on the set constraints still remaining unsatisfied at the current node.

Actually, the automaton constructed serves our purposes only indirectly: a solution to the $ACID$-unification problem is *not* always the sets of terms deduced from the labels at the various nodes of an accepted labeled term dag; but one can build naturally a grammar from any accepting run, and from the productions of this grammar a solution to the $ACID$-unification problem can be derived as finite non-empty sets of terms over the constants (cf. section 5).

We conclude therefrom that $ACID$-unification is NEXPTIME-decidable; it follows that the same upper bound holds for $ACUID$-unification too. A DEXPTIME lower bound has been deduced for both these problems via the results of [5], in [3]. (*Note*: For the particular class of set constraints expressing the $AC(U)ID$-unification problem, if the solutions can be arbitrary sets – meaning possibly infinite or empty – then satisfiability is known to be NEXPTIME-decidable (cf. [2])and DEXPTIME-hard (cf. [8]).) Several illustrative examples are given in the Appendix.

Our thanks to Hubert Comon who pointed out that reasoning with the $2n$-bit vectors alone at the states of our automaton would lead to incorrect conclusions; and to Sophie Tison and Witold Charatonik for useful discussions.

2 From $ACID$-Unification to Set Constraints

We consider the unification problem w.r.t. the following theory, denoted $ACID$:

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

This set of equations can be converted naturally to a convergent rewrite system; any ground term (over a given set of free constants) in normal form w.r.t. this system can then be viewed as a *set* of terms over '$*$' and the constants: indeed '$+$' can be viewed as

set union. An ACID-unification problem with free constants is that of solving modulo the above equational theory, a family of equations of the form: $\{s_1 = t_1, \ldots, s_k = t_k\}$, where the terms in the equations and/or the solutions can involve the given free constants.

We shall be assuming henceforth that our $ACID$-unification problem (after some normalization and decomposition steps) is in a *standard form*, i.e. every equation in our 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. A given $ACID$-unification problem can be reduced to a standard form in more than one manner. Since '$+$' is idempotent and '$*$' distributes left and right over '$+$', we may view this $ACID$-unification problem as a set constraints problem; e.g. in the first case, if y and z are interpreted as sets of terms over $*$ and the constants, then $y * z = \{s * t \mid s \in y,\, t \in z\}$.

The set constraints in this context are *with union* only, following the terminology of [8]; but with the additional restriction that all sets must be *finite, and non-empty* (since '$+$' has no unit element in the theory $ACID$). The problem of satisfiability of set constraints allowing arbitrary sets in the solutions, has been studied intensively over the past decade, in particular in [1, 2, 6, 9, 11, 7, 8]. Very few results seem to be known however, for solvability in terms of finite sets, or finite non-empty sets. The only result we actually know of is very general, and is based on the GTSA's of Gilleron, Tison and Tommasi (cf. Proposition 14, [11]); by appealing to this result, we may formulate our next result (but with no information on complexity):

Proposition 1 *The $ACID$-unification problem is decidable.*

We show in the following sections that this problem is NEXPTIME-decidable, via an approach based on the formalism of dag automata ([7]), enriched with labels at its nodes in order to serve our purposes. Before doing that, we first "pre-process" our unification problem, and eliminate some unsolvable cases thereby.

2.1 Pre-Processing an $ACID$-Unification Problem

Pre-processing our $ACID$-unification problem $\mathcal{P}$ (given in standard form) is based on a *dependency graph* associated to $\mathcal{P}$; The nodes of this graph will be the variables $\{x, y, z, \ldots\}$ appearing in $\mathcal{P}$; its edges are directed and labeled by one of the two symbols '$\{*, \supset\}$' as follows: For any two variables u, v, (u, v) is an edge labeled with '$*$' (resp. $\supset$) iff there is an equation of the form $u = v * w$ (resp. $u = v + w$) in our $ACID$-problem. The arcs of this graph will be called *dependency arcs*.

If u, v are two variables in the problem $\mathcal{P}$, then we write $u \succ v$ if there is a directed path from u to v. We write $u \succ_* v$ if there is a directed path from u to v with at least one edge along the path labeled with '$*$'. A variable of the problem from which there is no outgoing edge is called an *end-variable*.

Pre-processing $\mathcal{P}$ will consist in carrying out three operations on $\mathcal{P}$ and/or its associated dependency graph, with a view to eliminate syntactically some of the unsolvable cases. The first pre-processing step ensures that no variable is '$*$'-*above* itself; it is performed by the following operation.

Occur-Check-1: Return '*Fail*' if there is a variable u such that $u \succ_* u$; or, if $u \succ_* v$ for some v, and $u = a$ is in $\mathcal{P}$.

Example 1a. Occur-Check-1 returns 'Fail' on the problem: $z = a$, $z = x + y$, $y = u * v$.

Observe next that, if the problem $\mathcal{P}$ admits a solution then: for every 'sum'-equation $Z = X + Y$ in $\mathcal{P}$, terms of maximal height in Z have to be also in X or in Y; so we cannot have both $Z \succ_* X$ and $Z \succ_* Y$. This gives the following additional check on $\mathcal{P}$:

Occur-Check-2: Return '*Fail*' if there is a 'sum'-equation $Z = X + Y$ in $\mathcal{P}$, such that; $Z \succ_* X$ and $Z \succ_* Y$.

The final pre-processing step consists in 'locating' the variable(s) to the right of every 'sum'-equation $Z = X + Y$ in $\mathcal{P}$, which can contribute terms of maximal '$*$'-height to Z in a *finite, non-empty* solution for $\mathcal{P}$. For doing that, we consider *all possible* equivalence relations $\sim$ between the set variables such that: for every 'sum'-equation $Z = X + Y$ in $\mathcal{P}$, either $Z \sim X$ or $Z \sim Y$ non exclusively; these will be referred to as *height relations*.

Occur-Check-3. If for *every* height relation $\sim$ there are two variables in $\mathcal{P}$ such that $X \sim Y$ and $X \succ_* Y$ then return '*Fail*'.

We say that *pre-processing fails* on $\mathcal{P}$ iff 'Fail' is the value returned by one of the three Occur-Checks. The problem $\mathcal{P}$ is said to be in *pruned* form otherwise; in such a case, we introduce additional (height equivalence) arcs on the dependency graph of $\mathcal{P}$, as follows. For every $Z = X + Y$ in $\mathcal{P}$, if we have $X \sim Z$ (resp. $Y \sim Z$) for *some* height relation for which Occur-Check-3 does not '*Fail*', then introduce a $\sim$-arc from X (resp. Y) to Z.

Proposition 2 *i) Pre-processing any $ACID$-unification problem $\mathcal{P}$ can be performed in NP ii) The dependency graph of a pruned problem $\mathcal{P}$ is* '$*$'-loop free*: there is no node u such that $u \succ_* u$.*

In what follows, we show that solving an $ACID$-unification problem $\mathcal{P}$ in pruned form, amounts to showing the non-emptiness of the language of a labeled dag automaton associated with $\mathcal{P}$.

Example 1c. The problem: $V = X + Z$, $\quad V = W + U \quad W = X * Y$ is in pruned form: $V \sim Z$ in the first 'sum'-equation, and $V \sim W$ or $V \sim U$ in the second; and the dependency graph satisfies the '$*$'-loop free condition. We shall see later that this problem *is* solvable (cf. Example 2 below). □

3 Labeled Term Dags and Labeled Dag Automata

We first recall the notions of term dags and of dag automata as developed in [7]. A *term dag* (or *t-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 t-dag, so we often treat "node" and "term" as synonymous on a t-dag.

Definition 1 *A t-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 set of states, $F \subset Q$ is the set of final (or accepting) states, and Δ is a set of transition rules of the form : $f(q_1, q_2, ..., q_n) \rightarrow q$, where $f \in \Sigma$ is of arity of n, and the $q_i, 1 \leq i \leq n, q$ are in Q.*

A *run* r of a DA $\mathcal{A} = (\Sigma, Q, F, \delta)$ on a t-dag t is a mapping from the set of nodes of t to the set of states Q of DA, that respects the transition relation δ, i.e., for every node u, if the symbol at u is f of arity k, then $f(r(u_1), \ldots, r(u_k)) \rightarrow r(u)$ 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 $r(t) \in F$, i.e, it maps the root node to an accepting state. A t-dag t is accepted by a DA iff there is an accepting run on t. (Note that our runs are defined bottom-up.) The 'language' of a DA is the set of all t-dags that it accepts. It is easy to show that the class of t-dag automata is stable under union and intersection, but emptiness is hard to decide; that emptiness of a DA *is* NP-decidable is proved in [7].

A *labeled term dag*, or lt-dag in short, is a term dag equipped additionally with a mapping from the nodes of the dag to a given set of labels E. A motivation for adding labels is the fact that a labeled term dag can be used to specify finite sets of terms, when the labels are boolean, cf. the following section.

A *labeled dag automaton* (or LDA in short) is a quintuple $(\Sigma, Q, F, \Delta, E)$, where the first four components form a DA, E is a finite set of *labels*, and the transition relation Δ consists of *labeled* rewrite rules, of the form: $f(q_1, \ldots, q_k) \xrightarrow{l} q$, where k is the arity of f, and l is a label from E, and the q_i, q are elements of Q. A *run* r of an LDA $(\Sigma, Q, F, \Delta, E)$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:

- 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.

We shall paraphrase this condition as: "*the label on an LDA-transition must tally with the one at the node reached*". A run r is said to be *pseudo-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 pseudo-accepted by an LDA iff there is a pseudo-accepting run on t. The 'language' of an LDA is the set of all lt-dags that it pseudo-accepts. That its emptiness is NP-decidable for any given LDA is a direct consequence of the NP-decidability of emptiness of the language for any given DA. This is a key result on which our approach is based; it will be used in the coming section where will define *acceptance* of an lt-dag more restrictively, by taking a closer look at the labels on both the lt-dag and the LDA.

4 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^w_i.

Definition 2 *i) A label $m = (m_1, m_2, \ldots, m_n)$ is said to be a* permitted label *iff : $m_k = 0$ for* any *k for which there is an equation of the form $X_k = X_i * X_j$ in $\mathcal{P}$.*

ii) A label m is said to be a initial w.r.t. a ground constant *a iff m is a permitted label, and :*

$m_i = 1$ for every $i \in 1..n$ for which $X_i = a$ is in $\mathcal{P}$;

$m_j = 0$ for every $j \in 1..n$ for which $\mathcal{P}$ contains an equation $X_j = b$ with $b \neq a$.

iii) For any $i = 1..n$, the `set value` *deduced for X_i from the labels of an lt-dag t, is by definition the following set of ground terms: $\{t_w \mid w$ node on t such that: $l^w_i = 1\}$.*

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 (the problem $\mathcal{P}$ is assumed pruned).

Remark 1. If the set values deduced from the labels of an lt-dag t satisfy the $ACID$-problem $\mathcal{P}$, then t must satisfy the following condition: If two nodes u, v on an lt-dag t are such that $l^u_i = 1 = l^v_j$ and if there is a 'product'-equation $X_k = X_i * X_j$ in $\mathcal{P}$, then there must be a node w on the dag t such that $l^w_k = 1$ with subterm $t_u * t_v$. We shall refer to this condition as the `'closure property'`; an lt-dag having this closure property will be said to be a `closed` lt-dag. It does not seem likely that such a 'global' condition can be checked under an automaton-based approach; thus, for the notion of acceptance that we propose below the solution to the $ACID$-problem derived from an accepted lt-dag t is *not* in general the set values deduced from the labels of t.

4.1 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 [10]; 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 $\mathcal{S}$ 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 $\mathcal{S}$ 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 $\overline{A} \in \mathcal{S}$, 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.

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. 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 $\mathcal{P}$ if and only if: $l_i = 1 = m_j$. We shall agree to refer to any pstate $\overline{A}$ as *'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 $X^i_{\overline{A}}$ for every $i = 1..n$, and every pstate $\overline{A} \in \mathcal{S}$; and also an additional new symbol $\mathcal{T}$ (signifying '*to be covered*'); these symbols will be referred to as `dsymbols` (the 'd' stands for 'discriminating'). Two dsymbols $X^i_{\overline{A}}, X^j_{\overline{B}}$ 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:

$$X^k_{\overline{C}} = X^i_{\overline{A}} X^j_{\overline{B}}, \qquad \mathcal{T} = X^i_{\overline{A}} X^j_{\overline{B}}$$

where $\overline{A}, \overline{B}, \overline{C} \in \mathcal{S}$, 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 $\mathcal{S}$.

Definition 3 *(i) A* defect set *is any finite (possibly empty) set $\mathcal{M}$ containing dsymbols and defect equalities. Such a set $\mathcal{M}$ is said to be* `closed` *iff: $\mathcal{M}$ contains no open defect equalities, and for any two dsymbols $X^i_{\overline{A}}, X^j_{\overline{B}}$ in $\mathcal{M}$ which are conjugate, there is a closed equality in $\mathcal{M}$ with $X^i_{\overline{A}} X^j_{\overline{B}}$ as its rhs. The empty defect set is closed, by definition.*

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

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

To any label $m = (m_1, ..., m_n)$ we associate a state $(\overline{C}_m, \mathcal{M}_m)$, where:

- $\overline{C}_m$ is the pstate such that: $\overline{C}_m.l = \overline{C}_m.h = m$;
- the defect set $\mathcal{M}_m$ contains all the dsymbols $X^p_{\overline{C}_m}$ for indices p such that $\overline{C}_m.l_p = 1$, and all the open defect equalities $\mathcal{T} = X^i_{\overline{C}_m} X^j_{\overline{C}_m}$ for any conjugate pair of indices (i, j) such that $\overline{C}_m.l_i = 1 = \overline{C}_m.l_j$.

For example, let $Z = X * Y$ be the given $ACID$-problem, and $\{X, Y, Z\}$ its variable list. If $m = (110)$ is a given label, then the associated state is such that: $\overline{C}_m = \overline{C} = (110; 110)$ and $\mathcal{M}_m$ is the defect set $\{X_{\overline{C}}, Y_{\overline{C}}, \mathcal{T} = X_{\overline{C}} Y_{\overline{C}}\}$. This state is *not* closed. States associated to labels will be those reached under initialization steps for the runs of the LDA that we associate with the given $ACID$-problem.

4.2 The LDA associated to the $ACID$-Unification Problem

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

Definition 4 *The LDA associated to $\mathcal{P}$ is the quintuple $(\Sigma, \mathcal{Q}, \mathcal{F}, \Delta, \mathcal{L})$, where:*

- *Σ is the signature of the problem $\mathcal{P}$,*
- *The set of states $\mathcal{Q}$ is the set of all pairs $A = (\overline{A}, \mathcal{M}_A)$ of pstates and defect sets such that, if $\overline{A} = (l_1, l_2, \ldots, l_n; h_1, h_2, \ldots, h_n)$, then the following conditions hold:*
  - *$l_i \leq h_i$ for all i, and*
  - *$l_k = l_i \vee l_j$ for* every *sum-equation $X_k = X_i + X_j$ in the problem $\mathcal{P}$;*
- *The set $\mathcal{F}$ of* `accepting states` *is the set of all* `closed` *states in $\mathcal{Q}$ (in the sense of the definition above);*
- *The set of labels $\mathcal{L}$ for the LDA is the set of all n-bit vectors $(l_1, l_2, \ldots, l_n)$;*
- `(initialization)` *For any constant $a \in \Sigma$ and any label m* initial *w.r.t. a, there are transitions in Δ of the form: $a \xrightarrow{m} (\overline{C}_m, \mathcal{M}_m)$ (cf. Definition above).*
- `(progression)` *For any $A, B, C \in \mathcal{Q}$ 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:*
  - *$m = \overline{C}.l$ (*'label on the transition must tally with the one at the node reached*'),*
  - *for every equation in $\mathcal{P}$ of the form $X_k = X_i + X_j$, we have $m_k = m_i \vee m_j$;*
  - *for* every *equation in $\mathcal{P}$ of the form $X_k = X_i * X_j$, we have:*
  - $\overline{C}.l_k = \overline{A}.l_i \wedge \overline{B}.l_j$, *and*
  - $\overline{C}.h_i = \overline{C}.l_i \vee (\overline{A}.h_i \vee \overline{B}.h_i)$, *for* all *$i$.*

  (2) *Conditions on the defect sets $\mathcal{M}_A, \mathcal{M}_B, \mathcal{M}_C$:*
  - *$\mathcal{M}^{(s)}_C = \mathcal{M}^{(s)}_A \cup \mathcal{M}^{(s)}_B \cup \{X^j_{\overline{C}} \mid \overline{C}.l_j = 1\}$;*

- *Let $\mathcal{T}_C$ be the set of all open defect equalities $\mathcal{T} = X^i_{\overline{A'}} X^j_{\overline{B'}}$ whose rhs are conjugate dsymbols from $\mathcal{M}^{(s)}_C$; and let $\mathcal{C}_{C,A,B}$ be the set of all closed defect equalities* – whose rhs contains no dsymbols indexed by $\overline{C}$ – *and which either come from $\mathcal{M}_A$ or $\mathcal{M}_B$, or are of the form $X^k_{\overline{C}} = X^i_{\overline{A}} X^j_{\overline{B}}$ for conjugate triples (k,i,j) with $\overline{A}.l_i = \overline{B}.l_j = 1 = \overline{C}.l_k$; then $\mathcal{M}^{(e)}_C$ is the set obtained from $\mathcal{T}_C$* by replacing *each defect equality $\mathcal{T} = X^i_{\overline{A'}} X^j_{\overline{B'}}$ in $\mathcal{T}_C$* by all the closed defect equalities with the same rhs *in $\mathcal{C}_{C,A,B}$, if they exist.*

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. In addition, we shall assume (as we may) that our *runs are coherent in the sense that*: for any node w on t, if $r(w) = C$, then $\overline{C}.l = l^w$.

Remarks 2. i) If the defect set $\mathcal{M}_T$ at state T contains a dsymbol $X^i_{\overline{A}}$, 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 justification: Let C be a state image of a node w under a run, and suppose $\overline{C}.h_r = 1$ for all $r = 1..n$; let k be such that we have an equation $X_k = X_i * X_j$ in the problem $\mathcal{P}$, and suppose we also have $\overline{C}.l_i = 1$; then the term 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 where the term t_v got accepted in X_j; then (for the $ACID$-equation to be satisfied) the set X_k must contain the overterm $t_w * t_v$; 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 $\mathcal{T}_C$ in the definition represent all the potential products to be covered at or below C; an open equality in $\mathcal{T}_C$ not getting replaced by closed one represents a product which goes uncovered at C; it can come either from $\mathcal{M}_A$ or from $\mathcal{M}_B$, or get created at C because the term accepted there contributes to a product.

iv) 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 $\mathcal{M}_C$ at C may *not necessarily* contain the closed equality $X^k_{\overline{C}} = X^i_{\overline{A}} X^j_{\overline{B}}$ (cf. Examples 4, 6 below).

v) 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 6.)

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 the following figure is (pseudo-)accepted by the associated LDA. Arrange the set variables into an ordered list, say as $\{X, Y, Z, U, V, W\}$.

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:

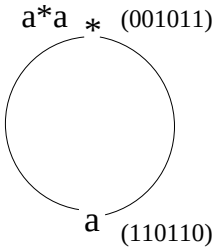

Fig. 1. *lt*-dag solving $V = X + Z,\ V = W + U,\ W = X * Y$

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

where $\mathcal{M}_A = \{X_{\overline{A}}, Y_{\overline{A}}, U_{\overline{A}}, V_{\overline{A}}, \mathcal{T} = X_{\overline{A}}Y_{\overline{A}}\}$ 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 $B = (001011; 111111; \mathcal{M}_B)$, where $\mathcal{M}_B = \{X_{\overline{A}}, Y_{\overline{A}}, U_{\overline{A}}, V_{\overline{A}}, Z_{\overline{B}}, V_{\overline{B}}, W_{\overline{B}}, W_{\overline{B}} = X_{\overline{A}}Y_{\overline{A}}\}$; 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 pseudo-accepted. □

In Example 2 above, the set values deduced from the labels of the pseudo-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 6).

Proposition 3 *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:*

i) the labels at u', v' are non-null and conjugate; and $(\overline{r(u')}, \overline{r(v')}) = (\overline{A'}, \overline{B'})$;

ii) no node (at or) below u on t has as subterm the product term $t_{u'} * t_{v'}$.

Proof. Let $C = r(u)$. If u is of height 0, there is nothing to prove; so assume that $u = u_1 * u_2$, and write $A_1 = r(u_1), A_2 = r(u_2)$; then the nodes u_1, u_2 are of smaller height than u. If the defect equality of the hypothesis 'gets created' at C, then it must be e.g. of the form $\mathcal{T} = X^i_{\overline{C}}X^j_{\overline{D}}$ for some state D image under r of some node v below u; and in this case we can take: $u' = u,\ v' = v$.

So we need to consider only the case where the open defect equality in $\mathcal{M}_C$ of the hypothesis comes from $\mathcal{M}_{A_1}$ (or $\mathcal{M}_{A_2}$), and does *not* get replaced by some closed defect equality at C, by the progression rule on the transitions. Suppose then it comes e.g. from $\mathcal{M}_{A_1}$; we then have: (i) $\overline{A'} \neq \overline{C}$ and $\overline{B'} \neq \overline{C}$; (ii) $\mathcal{M}_{A_2}$ does not contain any closed defect equality with $X^i_{\overline{A'}}X^j_{\overline{B'}}$ as rhs; and (iii) $(\overline{A'}, \overline{B'}) \neq (\overline{A_1}, \overline{A_2})$.

By an inductive argument on the height of the nodes, we then have two 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 $t_{u'} * t_{v'}$. Then, from the above properties (i)-(iii), one deduces that there can be no node (at or) below u on t, where the subterm is the product $t_{u'} * t_{v'}$. □

We now show that an lt-dag pseudo-accepted by the LDA may *not* always be a closed lt-dag, i.e. may not have the closure property.

Example 3. Consider the $ACID$-problem: $Z =^? X * Y$, and the solution set $X = \{a, c\},\ Y = \{b, d\}$. The following run is pseudo-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):

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

This example shows that if a solution to the $ACID$-problem is to be recovered from a pseudo-accepted lt-dag, in general it is *not* the set values deduced from the labels of the lt-dag. But it also gives a clue to what to do to make the LDA approach work.

The idea is the following: 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 pseudo-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). This idea will be formalized in the coming section.

5 A Grammar for deriving a Solution from a Pseudo-Accepted *lt*-dag

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 $\mathcal{G} = \mathcal{G}(r)$ 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 5 *Let w be any node of t such that its image $r(w) = E$ is an accepting state.*

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

ii) For every state B reached under initialization of r, and every ground constant b such that $r(b) = B$, add a production rule $\overline{B} \longrightarrow b$ to $\mathcal{G}$.

iii) For any $i \in 1..n$, the contribution $\mathcal{C}_i(w)$ 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 $\mathcal{G}$, for which $X^i_{\overline{C}} \in \mathcal{M}_E$, i.e. such that $\overline{C}.l_i = 1$.*

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

Proposition 4 *The $\mathcal{C}_i(w)$ as defined above are* non-empty *sets of terms over the ground constants and '$*$', for all i.*

Proof. Note that at the accepting state $E = r(w)$ we have $\overline{E}.h_k = 1$ for all k; so for any k there exists some dsymbol $X^k_{\overline{C}}$ in the defect set $\mathcal{M}_E$. We proceed by induction w.r.t. the strict ordering $\succ_*$ on the set of variables of $\mathcal{P}$, as follows.

Suppose that X_k is $\succ_*$-minimal (i.e., there is no $j \in 1..n$ such that $X_k \succ_* X_j$); then there must be a state B reached under initialization of r such that $\overline{B}.l_k = 1$, so there is a production in $\mathcal{G}(r)$ of the form $\overline{B} \longrightarrow b$; and then $\mathcal{C}_k(w)$ contains the ground constant b by definition. If X_k is not $\succ_*$-minimal, we consider two cases.

case 1): X_k is the lhs of a 'product'-equation $X_k = X_i * X_j$ in $\mathcal{P}$: We must then have a production of the form $\overline{C} \longrightarrow \overline{A} * \overline{B}$ (since the defect set $\mathcal{M}_E$ is closed); then we get the non-emptiness of $\mathcal{C}_k(w)$ by induction.

case 2): X_k is not $\succ_*$-minimal, and is not the lhs of a 'product'-equation in $\mathcal{P}$: Here we can find an index $r \in 1..n$, such that: there is a directed path on the dependency graph of $\mathcal{P}$ from X_k to X_r containing only '$\supset$' and '$\sim$' arcs, at all the intermediary nodes X_j along the path we have $X^j_{\overline{C}} \in \mathcal{M}_E$, and X_r is the lhs of a 'product'-equation in $\mathcal{P}$, We then deduce the non-emptiness of $\mathcal{C}_r(w)$ from case 1); that of $\mathcal{C}_k(w)$ follows subsequently, from the definition. □

Proposition 5 *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 $X_i = \mathcal{C}_i(w), i = 1..n$, 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 $X_i = \mathcal{C}_i(w)$.

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

Suppose we have an equation of the form $X_i = a$ in $\mathcal{P}$. Since E is accepting, we have $\overline{E}.h_i = 1$, so there is a dsymbol of the form $X^i_{\overline{A}}$ in $\mathcal{M}_E$, for some state A reached under the run r. By the assumed coherence of the run, any such A can only be reached under initialization; so we have a production in $\mathcal{G}(r)$ of the form $\overline{A} \longrightarrow a$, and the contribution $\mathcal{C}_i(w)$ of the run to X_i reduces to the singleton $\{a\}$.

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 $\mathcal{M}_E$. So any term in the contribution $\mathcal{C}_k(w)$ is either in $\mathcal{C}_i(w)$, or in $\mathcal{C}_j(w)$, or both. So we deduce: $\mathcal{C}_k(w) = \mathcal{C}_i(w) \cup \mathcal{C}_j(w)$.

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 $\mathcal{C}_k(w)$ 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 $\mathcal{M}_E$; 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 $X^i_{\overline{A}}, X^j_{\overline{B}} \in \mathcal{M}_E$, 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 $\mathcal{C}_k(w)$ of the run to X_k is the product of a term in the set $\mathcal{C}_i(w)$ with a term in the set $\mathcal{C}_j(w)$. This proves the inclusion $\mathcal{C}_k(u) \subset \mathcal{C}_i(u) * \mathcal{C}_j(u)$.

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 $\mathcal{M}_E$ contains then the two conjugate dsymbols $X^i_{\overline{A}}, X^j_{\overline{B}}$. Now r being pseudo-accepting, the defect set $\mathcal{M}_E$ at $E = r(w)$ must contain a closed defect equality of the form $X^k_{\overline{C}} = X^i_{\overline{A}} X^j_{\overline{B}}$ for some state C on the LDA for which $\overline{C}.l_k = 1$. The grammar $\mathcal{G}(r)$ contains then, by definition, a production $\overline{C} \longrightarrow \overline{A} * \overline{B}$; one deduces then that the contribution $\mathcal{C}_k(w)$ of the run to X_k contains the product term $s' * s''$. This proves the reverse inclusion $\mathcal{C}_i(w) * \mathcal{C}_j(w) \subset \mathcal{C}_k(w)$. □

Proposition 6 *The sets $\mathcal{C}_i(w)$ as defined above are all* finite*, for $i = 1..n$.*

Proof. Let $\overline{C}$ be any given non-terminal in $\mathcal{G} = \mathcal{G}(r)$ and π any $\overline{C}$-production of $\mathcal{G}$ (i.e. to say, of the form $\overline{C} \longrightarrow \overline{A} * \overline{B}$); then, for any $p \in 1..n$, define $\mathcal{C}_p(w)\,|_\pi$ as the subset of $\mathcal{C}_p(w)$ containing all ground terms derivable in $\mathcal{G}$ from $\overline{C}$ via π. By definition $\mathcal{C}_p(w)\,|_\pi$ is non-empty if and only if $\overline{C}.l_p = 1$.

Let us assume there is an index p for which $\mathcal{C}_p(w)$ is infinite. Then, for some non-terminal $\overline{C}$ there must be a $\overline{C}$-production π such that $\mathcal{C}_p(w)\,|_\pi$ is an infinite set. Such a $\overline{C}$ cannot be the pstate of a state reached under initialization of the pseudo-accepting run r; so π must be of the form $\overline{C} \longrightarrow \overline{A} * \overline{B}$, and the production π must correspond to a closed defect equality of the form $X^k_{\overline{C}} = X^i_{\overline{A}} X^j_{\overline{B}}$ in $\mathcal{M}_E$ (where E is the image of the root node of t under the run); we then observe that $\mathcal{C}_k(w)\,|_\pi = \mathcal{C}_p(w)\,|_\pi$, so is infinite.

Now by our previous proposition, the assignments $X_i = \mathcal{C}_i(w), i = 1..n$, satisfy the set constraints associated to $\mathcal{P}$. One deduces then that there must exist an $\overline{A}$-production (or a $\overline{B}$-production) ρ such that $\mathcal{C}_i(w)\,|_\rho$ (or $\mathcal{C}_j(w)\,|_\rho$) is infinite. Proceeding via an inductive reasoning w.r.t. the strict relation '$\succ_*$' on the set of variables of $\mathcal{P}$, we may then assume that there is a non-terminal $\overline{D}$, a $\overline{D}$-production β, and an $m \in 1..n$, such that: $\overline{D}.l_m = 1$, X_m is '$\succ_*$'-minimal, and $\mathcal{C}_m(w)\,|_\beta$ is infinite. But this is impossible, since no such $\overline{D}$-production can be of the form $\overline{D} \longrightarrow \overline{D_1} * \overline{D_2}$. □

We deduce our principal claim by putting together the last three propositions:

Theorem 1. *The LDA-approach is sound:* if there is an lt-dag pseudo-accepted by the LDA associated to a given $ACID$-unification problem $\mathcal{P}$, then $\mathcal{P}$ is solvable.

6 The LDA-approach is Complete

Theorem 2. *Suppose given an $ACID$-unification problem $\mathcal{P}$ in pruned form, and let $\mathcal{A}$ 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 pseudo-accepted by $\mathcal{A}$, 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 $\mathcal{D}$ 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 $\mathcal{D}$ 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 $\mathcal{D}$ 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 $\mathcal{D}$ (e.g. see Example 5 in the Appendix).

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 $\mathcal{D}$ must already be 'closed' in this sense, for the same reason.

Remains then to construct a pseudo-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 $\mathcal{D}$, by its label. (Note: because of the determinism, this is true also for the nodes 'under $\mathcal{D}$' 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 $\mathcal{D}$ – which correspond to the terms of *maximal* height in the given solution of $\mathcal{P}$ – have to be unmarked: 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 $\mathcal{D}$, completing it to cover the nodes of t which are not in $\mathcal{D}$ 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 pseudo-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 3: 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 $\mathcal{D}$ where our run is defined; so there is also a node on $\mathcal{D}$ 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 3, we conclude that the image of this root node cannot contain any open defect equalities. □

Theorem 3. *$ACID$-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 a pseudo-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 pseudo-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. □

Proposition 7 *$ACID$-unification and $ACUID$-unification are DEXPTIME-hard.*

Proof. The DEXPTIME-lower bound for $ACUID$-unification was established in [3]. It was also shown there how to adapt this proof so as to obtain the same lower bound for $ACID$-unification; cf. *loc. cit.* for the details. □

References

1. A. Aiken, E. Wimmers. *Solving Systems of Set Constraints.* In Proc. of the 7th IEEE Symposium on Logic in Computer Science, 1992 (LICS'92), pp. 329 – 340.
2. A. Aiken, D. Kozen,, M. Vardi, E. Wimmers. *The Complexity of Set Constraints.* In Proc. Conf. CSL'93, EACSL, September 1993, pp. 1–18.
3. S. Anantharaman, P. Narendran, M. Rusinowitch, *Unification over ACUI plus Distributivity/Homomorphisms*, Research Report RR-2002-11, LIFO, Université d'Orleans (Fr.) (`ftp://ftp-lifo.univ-orleans.fr/pub/Users/siva/RR2002-11.ps`)
4. F. Baader. *Unification in Commutative Theories.* Journal of Symbolic Computation 8:479–497, 1989.
5. F. Baader, P. Narendran. *Unification of Concept Terms in Description Logics.* Journal of Symbolic Computation 31 (3): 277–305, 2001.
6. L. Bachmair, H. Ganzinger, U. Waldemann. *Set Constraints are the Monadic Class.* In Proc. of the 8th IEEE Symposium on Logic in Computer Science, 1993 (LICS'93), pp. 75 – 83.
7. W. Charatonik. *Automata on DAG representations of finite trees.* Technical Report MPI-I-99-2-001, Max-Planck-Institut für Informatik, Saarbrücken, Germany.
8. W. Charatonik, A. Podelski. *Set Constraints with Intersection.* In Proc. of the 12th IEEE Symposium on Logic in Computer Science, Warsaw 1997 (LICS'97), pp. 362–372. (To appear in: *Information and Computation*).
9. 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/
10. R. Gilleron, S. Tison, M. Tommasi. *Solving Systems of Set Constraints using Tree Automata.* In Proc. STACS'93, Springer-Verlag, LNCS 665, 505–514.
11. R. Gilleron, S. Tison, M. Tommasi. *Set Constraints and Tree Automata* Information and Computation 149, pp. 1–41, 1999. (cf. also Technical Report IT 292, Laboratoire-LIFL, Lille, 1996.)

12. M. Minsky. *Computation: Finite and Infinite Machines. Prentice-Hall International*, London, (1972).
13. W. Nutt. *Unification in Monoidal Theories.* In Stickel (Ed.), Proc. CADE-10, Springer-Verlag, LNAI 449, pp. 618–632.
14. J.-M. Talbot, Ph. Devienne, S. Tison. *Generalized Definite Set Constraints.* In CONSTRAINTS: an International Journal 5(1-2):161–202, 2000.

Appendix: Examples

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 a pseudo-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: $\overline{B} \longrightarrow \overline{A} * \overline{A}, \quad \overline{A} \longrightarrow a$. The contribution of the run to the variables are therefore the sets of terms representable as: $X = \{\overline{A}\} = Y = U, \quad V = \{\overline{A},\ \overline{A} * \overline{A}\}, \quad Z = W = \{\overline{A} * \overline{A}\}$; 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 pseudo-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: $\overline{C} \longrightarrow \overline{A} * \overline{B}, \quad \overline{A} \longrightarrow a \mid c, \quad \overline{B} \longrightarrow b \mid d$. The contributions to the set variables at the root node give then the following assignment: $X = \{a, c\}, \quad Y = \{b, d\}, \quad Z = \{a * b, a * d, c * b, c * d\}$, which *is* a correct solution.

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

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

$$a, c \xrightarrow{100} A = (100; 100; \{X_{\overline{A}}\}); \qquad b, d \xrightarrow{010} B = (010; 010; \{Y_{\overline{B}}\});$$
$$A * B \xrightarrow{011} C = (011; 111; \{X_{\overline{A}}, Y_{\overline{B}}, Y_{\overline{C}}, Z_{\overline{C}},\ Z_{\overline{C}} = X_{\overline{A}} Y_{\overline{B}},\ \mathcal{T} = X_{\overline{A}} Y_{\overline{C}}\});$$

$$A * B \xrightarrow{001} D = (001; 111; \{X_{\overline{A}}, Y_{\overline{B}}, Z_{\overline{D}},\ Z_{\overline{D}} = X_{\overline{A}} Y_{\overline{B}}\});$$
$$A * C \xrightarrow{011} C = (011; 111; \{X_{\overline{A}}, Y_{\overline{B}}, Y_{\overline{C}}, Z_{\overline{C}},\ Z_{\overline{C}} = X_{\overline{A}} Y_{\overline{B}},\ \mathcal{T} = X_{\overline{A}} Y_{\overline{C}}\});$$

(A word of explanation on this last transition: The open $\mathcal{T} = X_{\overline{A}} Y_{\overline{C}}$ in $\mathcal{M}_C$ 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.)

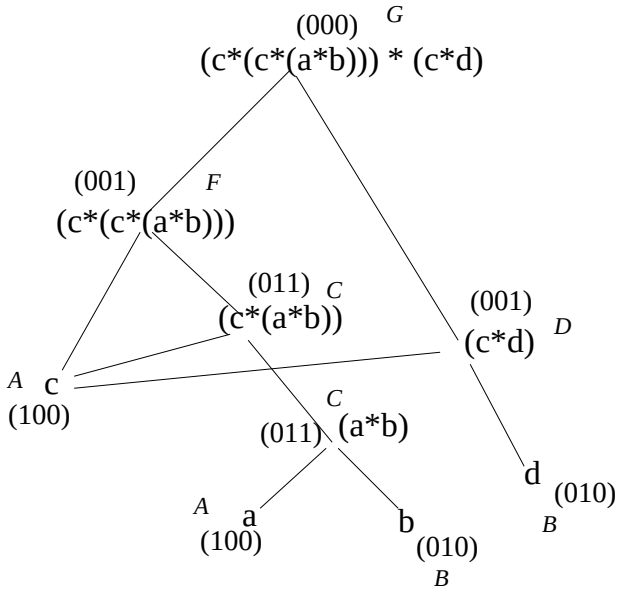

Fig. 2. A pseudo-accepted non-closed lt-dag for $Z =^{?} X * Y$

Then: $A * C \xrightarrow{001} F = (001; 111; \mathcal{M}_F)$, where
$\mathcal{M}_F = \{X_{\overline{A}}, Y_{\overline{B}}, Y_{\overline{C}}, Z_{\overline{C}},\ Z_{\overline{C}} = X_{\overline{A}} Y_{\overline{B}}, Z_{\overline{F}}, Z_{\overline{F}} = X_{\overline{A}} Y_{\overline{C}}\}$;

And finally: $F * D \xrightarrow{001} G = (000; 111; \mathcal{M}_G)$, where
$\mathcal{M}_G = \{X_{\overline{A}}, Y_{\overline{B}}, Y_{\overline{C}}, Z_{\overline{C}},\ Z_{\overline{C}} = X_{\overline{A}} Y_{\overline{B}},\ Z_{\overline{D}},\ Z_{\overline{D}} = X_{\overline{A}} Y_{\overline{B}}, Z_{\overline{F}}, Z_{\overline{F}} = X_{\overline{A}} Y_{\overline{C}}\}$;

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 pseudo-accepted. The solution for the problem derived at the node mapped to D is the simplest; the assignment is: $X = \{\overline{A}\},\ \ Y = \{\overline{B}\},\ \ Z = \{\overline{A} * \overline{B}\}$. 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 $\mathcal{M}_F$: The productions of the grammar w.r.t. this pseudo-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: $X = \{\overline{A}\},\ \ Y = \{\overline{B},\ \overline{A} * \overline{B}\},\ \ Z = \{\overline{A} * \overline{B},\ \overline{A} * (\overline{A} * \overline{B})\}$, where $\overline{A} \longrightarrow a \mid c,\ \overline{B} \longrightarrow b \mid d$; which is again a correct solution.

The grammar for the pseudo-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

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

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

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 pseudo-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 pseudo-accepted by the LDA associated (the ordered variable list is $\{U, X, Y, Z\}$).

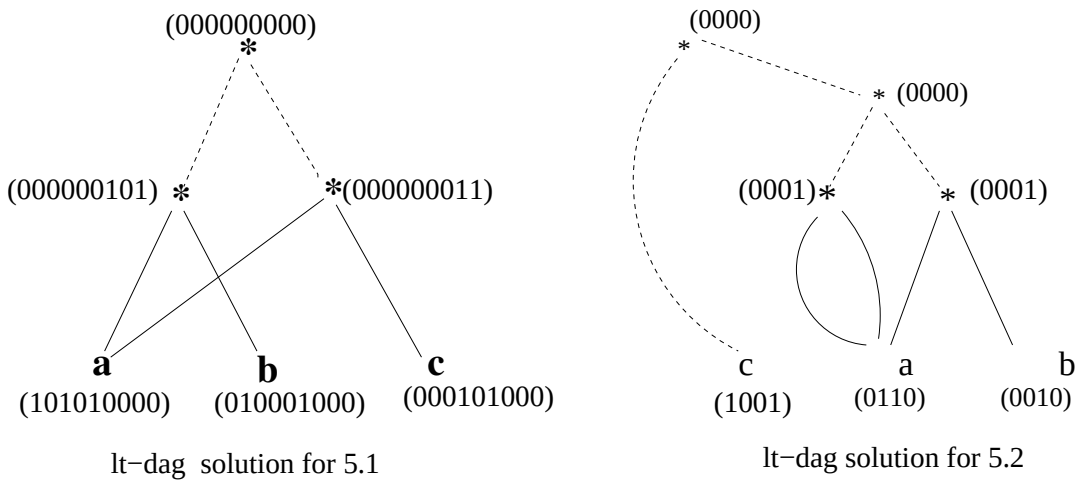

Fig. 3. lt-dags constructed from given solutions: Example 5

The lt-dags with the 'full lines' in the figure correspond in both cases to what was denoted by $\mathcal{D}$ in the proof of Theorem 2. □

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

$a \xrightarrow{111010111} A = (111010111; 111010111, \mathcal{M}_A)$ where
$\mathcal{M}_A = \{X_A^1, X_A^2, X_A^3, X_A^5, X_A^7, X_A^8, X_A^9, \mathcal{T} = X_A^9 X_A^3, \mathcal{T} = X_A^2 X_A^8\})$

$A * A \xrightarrow{110101000} B = (110101000; 111111111; \mathcal{M}_B)$, where
$\mathcal{M}_B = \{X_A^1, X_A^2, X_A^3, X_A^5, X_A^7, X_A^8, X_A^9, X_B^1, X_B^2, X_B^4, X_B^6, X_B^6 = X_A^9 X_A^3, \ \mathcal{T} = X_B^2 X_A^8\})$

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

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