Unification and Matching in Process Algebras

Qing Guo, Paliath Narendran, and Sandeep K. Shukla*

Department of Computer Science
University at Albany – State University of New York
Albany, NY 12222
Email:{guo,dran,sandeep}@cs.albany.edu**

Abstract. We consider the *compatibility checking* problem for a simple fragment of CCS, called BCCSP[12] , using equational unification techniques. Two high-level specifications given as two process algebraic terms with *free* variables are said to be compatible modulo some equivalence relation if a substitution on the free variables can make the resulting terms equivalent modulo that relation. We formulate this compatibility (modulo an equivalence relation) checking problems as *unification problems* in the equational theory of the the corresponding equivalence relation. We use van Glabbeek's equational axiomatizations [12] for some interesting process algebraic relations. Specifically, we consider equational axiomatizations for *bisimulation* equivalence and *trace* equivalence and establish complexity lower bounds and upper bounds for the corresponding unification and matching problems. We also show some special cases for which efficient algorithmic solutions exist.

1 Introduction

Often we are interested in processes which terminate after a finite number of steps. Such finitistic processes are useful in modeling circuits which engage in a transformational computation. *Sorting circuits* are examples of these. We consider *hierarchical specification* of such processes. In a hierarchical specification such processes are built from modules, where each module is itself a finitistic process. However, at an abstract specification level, some of the modules may be incompletely specified. For example, in a sorting circuit, there could be modules corresponding to the *comparators* as well as modules corresponding to particular *stages* of computation. The main circuit is then an interconnection of these modules. A process term describing this circuit at a high level of abstraction might contain variables representing underspecified modules. If such a specification is written in a process algebraic syntax, like CCS, usually the set of modules is described as a set of equations like

$$X = f(a, b, c, W, Y, Z)$$

* P. Narendran was supported in part by the NSF grant CCR-9404930 and S. Shukla was partially supported by the NSF grant CCR-94-06611.

** Q. Guo and S. Shukla's present addresses: {Qing Guo, Sandeep K. Shukla} GTE Laboratories Inc. 40 Sylvan Road, Waltham, MA 02254.

$$Y = g(a, b, c, W)$$
$$Z = h(a, W)$$

where f, g, h are process expressions involving the variables and action symbols shown in the brackets. Notice that W is not constrained by any equation and hence it stands for an underspecified module.

For simplicity, in this abstract we consider a very simple subset of Milner's CCS [8], called BCCSP in [12], for our syntax. Since we are investigating only finitistic processes, we will only need this fragment of CCS which does not allow recursion.

As explained above, the specification of a process is given in terms of equations. For example, a primitive module which does three (3) a actions and stops is specified as $X = a.a.a.0$. If a higher level module uses this module after a b action, that module is represented by a variable Y where $Y = b.X$. In our framework, underspecified modules are expressed as variables which are not constrained by such equations. Thus we obtain process expressions with free variables.

Given an implementation of such a specification, a standard method of checking the correctness of the implementation with respect to the specification is by checking whether the implementation and the specification are in the same equivalence class for a suitable equivalence relation on processes. Examples of such equivalence relations are bisimulation equivalence, trace equivalence, and many other equivalence relations in the linear time/branching time spectrum [12]. However, usually the implementation has full details of the behavior of the process and hence will be a ground expression denoting a fully specified process. To check if the implementation expression I and the specification expression S are bisimilar, we may use the sound and complete equational axiomatization of bisimulation for BCCSP given in [12], and solve an equational *matching* problem.

However, a more interesting (and often harder) problem is the following: Given two hierarchical specifications of the same system, one might be interested in their *compatibility*. We call two high-level specifications S_1 and S_2 as *ρ-compatible* if they can be instantiated into the same process *modulo* the relation ρ, by the same substitutions of the free variables (in other words, if they have a common implementation). Since we are using variables to stand for modules which are unspecified, two high level designers might come up with distinct expressions, involving these variables, such that both can be claimed to be a specification for the same system. Now checking ρ-compatibility can be formulated as the *unification problem* in the equational theory of the relation ρ in the fragment of the process algebra we are working with.

So far the equivalences we have considered in this framework are the two widely used equivalences, namely, *bisimulation* equivalence and *trace* equivalence. Complete equational axiomatizations of these equivalences for BCCSP appear in [12] and we consider matching and unification problems in these axiomatizations. The two axiomatizations give rise to two different algebras which we call *bisimulation algebra* and *trace algebra.* (In [7], there are sound and complete axiomatizations of similar algebras which are complete with respect to ground expressions. Our approach is applicable there too.) The problems we consider are *equational matching* (E-matching) and *equational unification* modulo these theories.

In the present work, we consider the issue of designing efficient algorithms for matching and unification and also derive complexity bounds. In Section 2 we show that unification modulo bisimulation algebra is an NP-complete problem. The problem is decidable in exponential time for trace algebra. We have obtained a *co-NP*-hard lower bound for unification modulo trace algebra. We also address the issue of restricting the problems to special cases for which more tractable algorithms exist.

1.1 Main Results

We show that unification modulo bisimulation is an NP-complete problem, even in the simple case when the signature of the algebra contains only one action symbol. For the trace equivalence case, we demonstrate a *co-NP*-hard lower bound and an exponential time upper bound. This upper bound result is obtained via a reduction to the emptiness checking of a particular kind of tree automaton which accepts the solutions of set constraints generated from the trace unification problem. However, in contrast to the bisimulation case, when the trace unification problems are limited to one action signature, we obtain a polynomial time algorithm.

1.2 Preliminary Definitions

In this subsection we define some of the terms that are used in the later sections of this paper. First we define the syntax of BCCSP terms and then we explain the the trace semantics of such terms. This helps in the intuitive understanding of our reduction of the unification problem for trace equivalence to a special class of *set constraints.*

BCCSP Terms: Let $\Sigma = \{0, +\} \cup A$ be a set of special symbols called the *signature*. Let X be a finite set of variables. The set of *open* terms over signature Σ and variable set X, denoted by $T(\Sigma, X)$ is a minimal set satisfying the following conditions:

1. $0 \in T(\Sigma, X)$
2. if $x \in X$, then $x \in T(\Sigma, X)$
3. if $t \in T(\Sigma, X)$ and $a \in A$, then $a.t \in T(\Sigma, X)$
4. if $t_1, t_2 \in T(\Sigma, X)$, then $t_1 + t_2 \in T(\Sigma, X)$

If $X = \phi$, then the terms are *ground* terms and the set of BCCSP ground terms over signature Σ is denoted by $T(\Sigma)$.

Trace: The traces of a ground BCCSP term is a set of strings over A. This set contains all the possible sequences of actions that the process represented by the term can engage in. Given a ground term $t \in T(\Sigma)$, the set $traces(t)$ is a minimal set satisfying the following conditions:

1. $traces(0) = \{\epsilon\}$
2. if $t = a.t'$, then $traces(t) = a.traces(t') \cup \{\epsilon\}$
3. if $t = t_1 + t_2$, then $traces(t) = traces(t_1) \cup traces(t_2)$

Lemma 1. *For any term t in $T(\Sigma)$, $traces(t)$ is a finite nonempty prefix-closed*[3] *set of strings over A.*

Trace Equivalence: Two terms t_1 and t_2 in $T(\Sigma)$ are said to be *trace equivalent* if and only if $traces(t_1) = traces(t_2)$.

In the following sections we overload the symbol = for the different semantics equivalences that we discuss. In section 2, we write $t_1 = t_2$, when the two ground terms t_1 and t_2 are bisimulation equivalent. In section 3, we write $t_1 = t_2$, when t_1 and t_2 are trace equivalent.

Definition 2. The *size* of a term $t \in T(\Sigma, X)$, denoted by $size(t)$ can defined inductively as follows:

1. $size(0) = 1$
2. $size(x) = 1$ for $x \in X$
3. $size(a.t) = 1 + size(t)$
4. $size(t_1 + t_2) = size(t_1) + size(t_2)$

The *size* of an equation over $T(\Sigma, X)$, is the sum of the sizes of both sides of the equation. The *size* of a set of equations is the sum of the sizes of the equations in the set.

2 Unification and Matching in Bisimulation Algebra

The following equational theory is a sound and complete axiomatization of bisimulation equivalence for BCCSP [12].

$$\begin{aligned}(x + y) + z &= x + (y + z)\\ x + y &= y + x\\ x + x &= x\\ x + 0 &= x\end{aligned}$$

We call this equational theory BiS. It is the same as the $ACI1$ theory in the unification literature, because '+' satisfies associativity, commutativity, idempotence and identity. It has been shown that the elementary $ACI1$-unification problem can be solved in polynomial time [9].

However, in the bisimulation algebra, we also have an additional set of unary functions: corresponding to each action in the the action set $Act = \{a_i \mid 0 \leq i \leq n\}$, we have a function $a_i\cdot$ which prefixes a process with action a_i. We denote this set of prefixing functions as Act by overloading the notation.

[3] A set of strings is prefix-closed if whenever a string w is in the set, all its prefixes including ϵ (empty string) are also in the set.

Definition 3. *BiS-UNIFIABILITY*, the E-unification problem over the bisimulation algebra is defined as follows:

INSTANCE: A set of pairs $S = \{\langle s_1, t_1\rangle, \langle s_2, t_2\rangle, \ldots, \langle s_m, t_m\rangle\}$, where s_i, t_i are first-order terms over the signature $\{0, Act, +\}$.
QUESTION: Does there exist a unifier σ such that
$\sigma(s_1) =_{BiS} \sigma(t_1),\ \sigma(s_2) =_{BiS} \sigma(t_2),\ \ldots, \sigma(s_m) =_{BiS} \sigma(t_m)$?

If all t_is are ground in S, then the problem is called a *BiS-Matching* problem.

Theorem 4. *BiS-UNIFIABILITY is NP-complete.*

Proof: The elementary *ACI1*-unification problem with linear constant restrictions can be solved in polynomial time [9]. So the *BiS-UNIFIABILITY* problem can be shown to be in *NP* by Theorem 2.1 from [2].

To show *NP*-hardness, we present a polynomial time and linear size reduction of the *NP*-complete problem *GRAPH 3-COLORABILITY* to *BiS*-unification.

Let graph $G = (V, E)$ and color set $C = \{c_1, c_2, c_3\}$ be any instance of *GRAPH 3-COLORABILITY,* where $V = \{v_1, v_2, \ldots, v_n\}$ and $n \geq 3$. We construct an instance of *BiS*-unification as follows.

For every edge $e_k = \{v_i, v_j\} \in E$, we construct the pair

$$\langle a.v_i + a.v_j + a.z_k,\ a.0 + a.a.0 + a.a.a.0\rangle$$

where node names v_i, v_j are used as variables, $Z = \{z_1, z_2, \ldots, z_m\}$ is a set of extra variables, where $m = |E|$, and a is an action. The construction results in an instance of *BiS*-unification. Let $B(t)$ be the equivalence class of BCCSP terms which contains the term t. Then, the colors c_1, c_2 and c_3 are represented by the equivalence classes $B(0)$, $B(a.0)$ and $B(a.a.0)$ respectively. Note that if the above unification pair is *BiS*-unifiable then the variables v_i, v_j, z_k can only be substituted for by terms in $B(0) \cup B(a.0) \cup B(a.a.0)$. Moreover, these terms must be from distinct equivalence classes.

If the input graph is 3-COLORABLE, then there is a color assignment assigning every v_i and v_j two distinct colors, where $\{v_i, v_j\} \in E$. There is one color left out of c_1, c_2, c_3. By assigning to z_k a term from the equivalence class corresponding to the remaining color, the color assignment can be extended to a unifier for the instance of *BiS*-unification.

If the instance of *BiS*-unification problem is solvable then there is a simultaneous unifier θ for all pairs $(a.v_i + a.v_j + a.z_k,\ a.0 + a.a.0 + a.a.a.0)$. As mentioned before, $\theta(v_i)$ and $\theta(v_j)$ can only be ground terms from $B(0) \cup B(a.0) \cup B(a.a.0)$ such that they are from distinct equivalence classes. Since these three equivalence classes $B(0)$, $B(a.0)$ and $B(a.a.0)$ represent the 3 distinct colors, a coloring assignment can be obtained from θ which solves the given instance of *3-COLORABILITY.* □

Note that only one action symbol was needed for the reduction used in the above proof. Furthermore, because the right side of every pair above is ground, the proof also shows that *BiS*-matching is *NP*-hard.

A special case of BiS-unification, where the terms are *deterministic* (defined below), can be solved in polynomial time. The algorithm is similar to the one for standard (i.e., empty theory) unification. We omit the details.

Definition 5. *Deterministic* terms:

1. 0 is deterministic.
2. $a_k.0$ is deterministic, $a_k \in Act$.
3. X is deterministic, if X is a variable.
4. $a_{k_1}.E_1 + a_{k_2}.E_2 + \cdots + a_{k_m}.E_m$ is deterministic, where the $E_i's$ are deterministic, $a_{k_i} \in Act$ and $a_{k_i} \neq a_{k_j}$ if $i \neq j$.

3 Unification in Trace Algebra

The equational axiomatization of trace algebra consists of all the axioms of BiS, plus an axiom to capture the distributivity of the *prefixing* with actions a over $+$. In other words, the equational theory is

$$\begin{aligned}
(x + y) + z &= x + (y + z) \\
x + y &= y + x \\
x + x &= x \\
x + 0 &= x \\
a.(x + y) &= a.x + a.y \qquad \forall\, a \in Act
\end{aligned}$$

Note that $a.0 \neq 0$. We show that the elementary unification problem modulo trace algebra is *co-NP*-hard. We can prove that it is decidable; however, there is a gap between the lower bound and the upper bound which is *DEXPTIME*.

Theorem 6. *The Unification problem of terms in BCCSP modulo trace algebra is co-NP-hard.*

Proof: The DNF non-tautology problem is known to be *NP*-complete: given a DNF $c = c_1 \vee c_2 \vee \ldots \vee c_m$, where $c_i = x_{i1} \wedge x_{i2} \wedge \ldots \wedge x_{ij}$, it is *NP*-complete to determine if c is NOT a tautology. Hence it is *co-NP*-complete to determine if c is a tautology. We can reduce the tautology problem to the trace unification problem as follows.

Let n be the number of boolean variables in the DNF instance. We create the following set of simultaneous equations:

$$\begin{aligned}
X_1 &= a.X_2 + b.X_2 \\
X_2 &= a.X_3 + b.X_3 \\
&\vdots \\
X_n &= a.0 + b.0 \\
X_1 &= B_1^1 + B_1^2 + \ldots + B_1^m
\end{aligned}$$

where 0 stands for the zero process and the B_1^i's and X_j's are variables. We now introduce equations for each B_1^i as follows:

if x_1 appears in c_i unnegated then $B_1^i = b.B_2^i$
if x_1 appears in c_i negated then $B_1^i = a.B_2^i$
if x_1 does not appear in c_i then $B_1^i = a.B_2^i + b.B_2^i$.

Note that B_j^i are new variables. We also add equations for each B_j^i as follows:

$$B_j^i = \begin{cases} b.B_{j+1}^i & \text{if } x_j \text{ appears in } c_i \text{ as a positive literal} \\ a.B_{j+1}^i & \text{if } x_j \text{ appears in } c_i \text{ as a negative literal} \\ a.B_{j+1}^i + b.B_{j+1}^i & \text{if } x_j \text{ does not appear in } c_j \end{cases}$$

and $B_{n+1}^i = 0$.

This system of equations has a (unique) solution and hence a (unique) unifier iff c is a tautology. The reason is that $traces(X_1)$ = the prefix closure[4] of $\{a, b\}^n$. However, by setting $X_1 = B_1^1 + ... + B_1^m$ and coding all possible assignments that make c true in the B_1^i's, we get the desired effect. Also, note that the size of this set of equations is $O(mn)$ and the reduction is polynomial time.□

Now we show how the trace unification problem can be solved in exponential time using tree automata working on finite trees [4].

Definition 7. A set of equations over $T(\Sigma, X)$ is said to be in *reduced form* if all the equations in the set are of one of the following three forms:

1. $x = 0$ for some $x \in X$
2. $x = a.y$ for some $x, y \in X$ and $a \in A$
3. $x = y + z$ for some $x, y, z \in X$

It can easily seen that any set of equations over $T(\Sigma, X)$ can be reduced (preserving the solutions) to a *reduced form* with at most a linear increase in the size of the set of the equations.

Lemma 8. *Any set of equations over $T(\Sigma, X)$ can be reduced (preserving the solutions) to a* reduced form *with at most a linear increase in the size of the set.*

By virtue of this lemma, from now on, we only consider equations in reduced form.

Given an equation of the form $x = 0$, from the definition,

$$traces(x) = \{\epsilon\}$$

Similarly, given an equation of the form $x = a.y$, from the definition of traces

$$traces(x) = a.traces(y) \cup \{\epsilon\}$$

[4] The prefix closure of a string is the set of all prefixes of the string, including the empty string and the string itself. The prefix closure of a set of strings is the union of the prefix closures of the elements of the set.

Also, for an equation of the form $x = y + z$, it must be that

$$traces(x) = traces(y) \cup traces(z)$$

By denoting each $traces(x)$ by set variables T_x which can take up values over nonempty, prefix-closed sets of strings over A, we can reduce the unification problem over $T(\Sigma, X)$ to a set of set constraints *with equality only*. For example, the following set of equations

$$x = y + z, y = a.w, z = 0$$

can be reduced to

$$T_x = T_y \cup T_z, T_y = a.T_w \cup \{\epsilon\}, T_z = \{\epsilon\}$$

Let Ψ be the above transformation which takes a unification problem modulo trace algebra and reduces it to a system of set equations as shown. It is not hard to show that if these set equations have a solution, say θ, over finite nonempty sets of strings over A, then they also have a solution over finite, nonempty, *prefix-closed* sets of strings over A—to get the new solution we merely have to take the prefix-closures of each $\theta(x)$ for variables x.[5] In short, if P is an instance of unification modulo trace algebra, then P is unifiable if and only if $\Psi(P)$ has a solution over finite nonempty sets of strings over A. (Thus the requirement that the sets be prefix-closed is removed.)

We can transform this set of equations further, with at most a linear increase in size, to a similar reduced form as defined in this section. In the above example, we can introduce new set variables T_u and T_w^a and change the equations (preserving solutions) as follows:

$$T_x = T_y \cup T_z, T_y = T_w^a \cup T_u, T_u = \{\epsilon\}, T_w^a = a.T_w, T_z = \{\epsilon\}$$

In other words,

Theorem 9. *Every elementary unification problem modulo trace algebra can be reduced to a system of set equations, in which every equation is either of the form $X = \{\epsilon\}$, $X = a.Y$, or $X = Y \cup Z$ where X, Y, Z are variables over finite, nonempty sets of strings over A and $a \in A$.*

We want to point out that these set constraints are **not** a subcase of those considered by Aiken et. al. [1] since they allow finite as well as infinite sets in the solutions.

Given a set of such equations, one can design, in exponential time, a tree automaton that accepts their solutions (encoded in a tree form). Since the emptiness problem for tree automata can be solved in polynomial time, we get our main decidability result. However, before we construct the tree automata, one further step is needed. Let *rev* be the string reverse function defined on a set of strings. $rev(X)$ *reverses* every string in the set X. For instance

[5] Note that this technique does not work *in general*; it works for equations that Ψ produces. For instance, the equation $X = a.Y$ does not have a prefix-closed solution.

$rev(\{c.b.a,\ a.b,\ c\}) = \{a.b.c,\ b.a,\ c\}$. It is clear that $X = a.Y$ is solvable if and only if $rev(X) = rev(Y).a$ is solvable. Recall that by Theorem 9, an elementary *Trace*-unification problem can be reduced to a system of set equations, in which every equation is of the form $X = \{\epsilon\}$, $X = a_i.Y$, or $X = Y \cup Z$. By applying the reverse function this set of equations can be transformed into one where every equation is either of the form $X = \{\epsilon\}$, $X = Y.a$, or $X = Y \cup Z$ where X, Y, Z are variables over finite, nonempty sets of strings over A and $a \in A$.

Now, we discuss the method of solving such set equations using Doner's tree acceptor [4]. First we discuss how to encode sets of strings as a Σ_n-tree.

Let Σ_n be the set of n-tuples over $\{0,1\}$; 0^n denotes the n-tuple consisting entirely of 0's.

Definition 10. A Σ_n-tree of order p is a function $\tau : A \rightarrow \Sigma_n$, where A is a finite prefix-closed subset of $\{a_0, a_1, \ldots, a_{p-1}\}^*$, i.e., if $uv \in A$ then $u \in A$.

Intuitively, a Σ_n-tree τ of order p is a tree such that every node has at most p children. The root node represents the empty string, and if the root node has p children, then the children of the root node from left to right represent the strings a_0,...,a_{p-1}. Generally, if a node represents a string u, and has degree p then the p children of that node from left to right will represent the strings $u.a_0$, ...$u.a_{p-1}$ respectively. The mapping τ labels each node by n-tuples from Σ_n. Given n finite sets $X_1, \ldots, X_n$ of strings over A, they can be coded in the form of a Σ_n-tree τ. If the i^{th} component of $\tau(v)$ at the node v is 1 then v belongs to the set X_i, otherwise it does not belong to X_i.[6]

The class of all Σ_n-trees of order p is denoted by $\Sigma_n^{\#}$. For notational convenience, we restrict our examples in this paper to trees of order 2, i.e., binary trees. The domain of trees of order 2 is denoted by $\{a,b\}^*$. Let τ be a binary tree. The subtree of τ beginning at a is denoted by $\tau\uparrow_a$ and the subtree of τ beginning at b is denoted by $\tau\uparrow_b$. We use λ to denote the empty tree. For all $\sigma \in \Sigma_n$ and $\tau, \tau' \in \Sigma_n^{\#}$, $\sigma[\tau,\ \tau']$ is the unique tree π such that $\pi(w) = \sigma$, $\pi\uparrow_a = \tau$ and $\pi\uparrow_b = \tau'$.

Definition 11 (Doner [4])**.** A Σ_n-tree acceptor is a 4-tuple $\mathcal{A} = \langle Q,\ t,\ q_0,\ F\rangle$ where

1. Q is a finite set of states;
2. $t : Q \times Q \times \Sigma_n \rightarrow Q$ is the transition function;
3. $q_0 \in Q$ is the initial state;
4. $F \subseteq Q$ is the set of accept states.

Function $\bar{t} : \Sigma_n^{\#} \rightarrow Q$ is defined by

$$\bar{t}(\lambda) = q_0$$
$$\bar{t}(\sigma[\tau, \tau']) = t(\bar{t}(\tau), \bar{t}(\tau'), \sigma)$$

$\mathcal{A}$ *accepts* a tree $\tau \in \Sigma_n^{\#}$ if $\bar{t}(\tau) \in F$. $T(\mathcal{A})$ denotes the set of Σ_n-trees accepted by $\mathcal{A}$.

[6] This coding technique was first used by Rabin [10].

Theorem 12. [11] *If $\mathcal{A}$ is any tree acceptor then the emptiness checking of $T(\mathcal{A})$ can be done in time polynomial in the size of $\mathcal{A}$.*

Proof: One way to show this is by reducing the problem (in polynomial time and space) to the satisfiability problem for Horn formulae (HORNSAT). Since HORNSAT is decidable in linear time [5], the result follows. See [6], for instance, for a detailed proof. □

Theorem 13 (Doner [4]**).** *Let $\mathcal{A}$ and $\mathcal{B}$ be two tree acceptors. A tree acceptor $\mathcal{C}$ can be effectively constructed (in polynomial time) such that $T(\mathcal{C}) = T(\mathcal{A}) \cap T(\mathcal{B})$.*

Given an equation on set variables $X_1, \ldots, X_n$, the variables can be coded in the form of n-tuples. Then we can construct a Σ_n-tree acceptor $\mathcal{A}$ such that if $\tau \in T(\mathcal{A})$ then the labels of the tree τ give a solution of the equation (under the interpretation given above). Hence $T(\mathcal{A}) \neq \emptyset$ if and only if the equation is solvable.

Given two equations e_1, e_2 over set variables $X_1, \ldots, X_n$, we can construct two Σ_n-tree acceptors $\mathcal{A}_1$ and $\mathcal{A}_2$ for e_1 and e_2 respectively. By Theorem 13 we can construct another Σ_n-tree acceptor $\mathcal{A}$ such that $T(\mathcal{A}) = T(\mathcal{A}_1) \cap T(\mathcal{A}_2)$. Then e_1 and e_2 are solvable simultaneously if and only if $T(\mathcal{A}) \neq \emptyset$. This is the basic idea of solving set equations using Σ_n-tree acceptors.

Example 1. Let $X = Y.a$ be an equation, where X, Y are set variables. We construct a Σ_2-tree acceptor (of order 2) $\mathcal{A}$ such that $T(\mathcal{A})$ contains trees which satisfy the above equation.
$\mathcal{A} = \langle\ \{q_0, q_x, q_a, q_r\},\ t,\ q_0, \{q_a\}\rangle$ where

- q_0: the initial state.
- q_x: current node is in X.
- q_a: the tree below this node is acceptable.
- q_r: reject state.

The transition function t is defined as follows:

$$
\begin{array}{ll}
t(q_0,\ q_0,\ 00) = q_0 & t(q_0,\ q_a,\ 00) = q_a \\
t(q_0,\ q_0,\ 10) = q_x & t(q_0,\ q_a,\ 10) = q_x \\
t(q_0,\ q_0,\ \_1) = q_r & t(q_0,\ q_a,\ \_1) = q_r \\
\\
t(q_x,\ q_0,\ \_0) = q_r & t(q_x,\ q_a,\ \_0) = q_r \\
t(q_x,\ q_0,\ 01) = q_a & t(q_x,\ q_a,\ 01) = q_a \\
t(q_x,\ q_0,\ 11) = q_x & t(q_x,\ q_a,\ 11) = q_x \\
\\
t(q_a,\ q_0,\ 00) = q_a & t(q_a,\ q_a,\ 00) = q_a \\
t(q_a,\ q_0,\ 10) = q_x & t(q_a,\ q_a,\ 10) = q_x \\
t(q_a,\ q_0,\ \_1) = q_r & t(q_a,\ q_a,\ \_1) = q_r \\
\\
t(\_\_,\ q_x,\ \_\_) = q_r & t(\_\_,\ q_r,\ \_\_) = q_r \\
t(q_r,\ \_\_,\ \_\_) = q_r &
\end{array}
$$

Equations $X = Y \cup Z$ and $X = \{\epsilon\}$ can be handled similarly. It is straightforward to construct a Σ_3-tree acceptor for the former and a Σ_1-tree acceptor for the latter.

Given a system P of m equations on set variables $X_1, X_2, \ldots, X_n$, in which each equation is either in the form $X = \{\epsilon\}$, or $X_i = X_j.a$, or $X_i = X_j \cup X_k$, a tree acceptor $\mathcal{A}$ can be constructed such that P is solvable if and only if $T(\mathcal{A}) \neq \emptyset$. (If the number of actions is κ then we have to construct Σ_n-tree acceptors of order κ for all equations before we perform the intersection.) It can be shown that the time complexity of constructing acceptor $\mathcal{A}$ for the equation system is $O(2^{mn})$. Hence, by Theorem 12, the time complexity of the emptiness checking of $T(\mathcal{A})$ is in polynomial of 2^{mn}. Thus the equation system P can be solved in exponential time.

Theorem 14. *Elementary unifiability problem modulo trace algebra can be solved in exponential time.*

Surprisingly, the problem turns out to be solvable in polynomial time when there is only one action symbol. This is in marked contrast to the bisimulation case, where the problem is *NP*-hard even when there is only one action.

4 Single-action Trace Algebra

We consider an interesting subclass of the trace algebra, called the *single-action trace algebra*, where only one action symbol is allowed, i.e., $Act = \{a\}$. We show that with this restriction the unifiability problem can be solved in polynomial time. The proof is by reduction to HORNSAT.

Lemma 15. *Let S be a set of elementary unification pairs in the single-action trace algebra. S can be reduced to a set of equations W, in which each equation is in one of the following forms:*

$$\begin{aligned} x_m &= max(x_i, x_j) \\ x_i &= x_j + 1 \\ x_i &= 0 \end{aligned}$$

where x_i, x_j, x_m are variables over non-negative integers. S is unifiable in the single-action trace algebra iff W is solvable.

Proof: Straightforward by Theorem 9 and since there is only one action symbol a, every ground term is equivalent to a term of the form $a^i.0$. □

Now, we study the solvability of equation systems which are in the form of the above equation system W.

Lemma 16. *Let $X = \{x_1, x_2, \ldots, x_n\}$ be a set of variables over non-negative integers, and W be a set of equations, in which each equation is in one of the* *following forms:*

$$x_l = max(x_i, x_j)$$
$$x_i = x_j + 1$$
$$x_i = 0 \qquad\qquad x_i,\ x_j,\ x_l \in X$$

If W is solvable then there is a solution in which $\forall\ 1 \leq i \leq n:\ x_i < n$.

Proof: If W only contains equations of the form $x_l = max(x_i, x_j)$ then W is solvable and this lemma is trivially true. Without loss of generality, we assume that there is at least one equation in W, which is in the form of $x_i = x_j + 1$ or $x_i = 0$.

A solution of W assigns a nonnegative integer to each variable. If there is a solution for W then based on the value of each variable, variables in X can be arranged in the following order:

$$x_{i_1} \leq x_{i_2} \leq \ldots \leq x_{i_n}$$

Furthermore, all variables which have the same value can be renamed to be the same. For instance, if $x_i = 3, x_j = 3, x_k = 3$ then x_j and x_k can be renamed by x_i. After renaming, we have a strict ordering on variables:

$$x_{i_1} < x_{i_2} < \ldots < x_{i_m}$$

where $m \leq n$ and x_{i_m} has the largest value in the solution of W.

After renaming, equations in the form $x_l = max(x_i, x_j)$ become redundant and hence can be removed from W.

Thus we are left with equations of the form

$$x_k = x_j + 1, \text{ or}$$
$$x_k = 0$$

where $j, k \in \{i_1, \ldots, i_m\}$.

Now we prove that this system of inequalities, comprising the ordering chain and W, has a solution $\{x_{i_j} \leftarrow j-1\}$ by induction on m, the number of variables in the strict ordering chain.

Claim: Let I_W be a system of inequalities over variables $\{x_1, \ldots, x_m\}$ consisting of

(i) a total order $x_1 < x_2 < \ldots < x_m$

(ii) equations of the form $x_i = x_j + 1$ and $x_k = 0$.

Then if I_W is solvable, then $\theta = \{x_i \leftarrow i - 1\}$ is a solution.

Base case: $m = 1$. There is only one variable in the chain. The only possible equation in I_W is $x_1 = 0$. (An equation of the form $x_1 = x_1 + 1$ is not solvable.) Hence $\{x_1 \leftarrow 0\}$ is a solution.

Induction step: Assume the solution holds for $m = p$. Consider $m = p + 1$. Because x_{p+1} is the largest element in the variable chain, there can be at most one equation of the form $x_{p+1} = x_p + 1$ in I_W. Remove that equation from I_W to get I'_W. Applying the induction hypothesis on I'_W, there must be a solution where $x_{i_q} = q - 1$ for all $q \leq p$, and clearly this solution can be extended to a solution of I_W where $x_{p+1} = p$. □

Given this bound, we can reduce the solvability problem for W to the satisfiability problem for a set of Horn clauses. We introduce propositional variables $Q_{x \leq i}$ for every variable x and $0 \leq i \leq n-1$. The Horn clauses are constructed as follows:

(A) For each variable x, we add a unit clause $Q_{x \leq n-1}$ as well as clauses

$$\forall\, 0 \leq i < (n-1): \qquad Q_{x \leq i} \Rightarrow Q_{x \leq i+1}$$

(B) For each equation of the form $z = max(x, y)$, we form the clauses

$$\forall\, 0 \leq i \leq (n-1): \qquad \begin{aligned} Q_{x \leq i} \wedge Q_{y \leq i} &\Rightarrow Q_{z \leq i} \\ Q_{z \leq i} &\Rightarrow Q_{x \leq i} \\ Q_{z \leq i} &\Rightarrow Q_{y \leq i} \end{aligned}$$

(C) For each equation of the form $y = x + 1$, we form the clauses

$$\begin{aligned} &\neg Q_{y \leq 0} \\ \forall\, 0 \leq i \leq n-2: \qquad & Q_{x \leq i} \Rightarrow Q_{y \leq i+1} \\ & Q_{y \leq i+1} \Rightarrow Q_{x \leq i} \end{aligned}$$

(D) For each equation of the form $x = 0$, we form the clause $Q_{x \leq 0}$.

It can now be shown that this set of clauses is satisfiable if and only if the set of equations W we started with has a solution. It should not be hard to see that given a bounded solution of W we can construct a satisfying assignment for the Horn clauses. On the other hand, let ϕ be a satisfying assignment for the set of Horn clauses. For each variable x pick the smallest i such that $Q_{x \leq i}$ is set to true by ϕ. It can be shown that an assignment constructed like this is a solution for W. We informally outline the argument below.

(i) Let $z = max(x, y)$ be an equation in W. Let $i = i_x$ be the smallest i such that $Q_{x \leq i_x}$ is set to true by ϕ, $i = i_y$ be the smallest i such that $Q_{y \leq i_y}$ is set to true by ϕ, and $i = i_z$ be the smallest i such that $Q_{z \leq i_z}$ is set to true by ϕ. We need to show that $i_z = max(i_x, i_y)$. This can be proved by contradiction as follows. Assume $i_z \neq max(i_x, i_y)$. Without loss of generality, we can assume that $i_x \geq i_y$. There are two cases:

1. $i_z > max(i_x, i_y)$, i.e., $i_z > i_x$.
   Because i_z is the smallest i such that $Q_{z \leq i_z}$ is set to true by ϕ. $Q_{z \leq i_x}$ must be set to false by ϕ. Therefore the clause $Q_{x \leq i_x} \wedge Q_{y \leq i_x} \Rightarrow Q_{z \leq i_x}$ is not satisfiable by ϕ, which is a contradiction.

2. $i_z < max(i_x, i_y)$, i.e., $i_z < i_x$.
Because i_x is the smallest i such that $Q_{x \leq i_x}$ is set to true by ϕ, $Q_{x \leq i_z}$ must be set to false by ϕ. Therefore the clause $Q_{z \leq i_z} \Rightarrow Q_{x \leq i_z}$ is not satisfiable by ϕ, which is a contradiction.

(ii) Let $y = x + 1$ be an equation in W. Let $i = i_x$ be the smallest i such that $Q_{x \leq i_x}$ is set to true by ϕ, and $i = i_y$ be the smallest i such that $Q_{y \leq i_y}$ is set to true by ϕ. We need to show that $i_y = i_x + 1$. This can be proved by contradiction as follows. Assume $i_y \neq i_x + 1$. There are two cases:

1. $i_y > i_x + 1$.
$Q_{y \leq i_x + 1}$ must be set to false by ϕ. Therefore the clause $Q_{x \leq i_x} \Rightarrow Q_{y \leq i_x + 1}$ is not satisfiable by ϕ, which is a contradiction.
2. $i_y < i_x + 1$.
Because $i_y > 0$ ($\neg Q_{y \leq 0}$ is set to true in ϕ), it must be that $i_x > 0$. $Q_{x \leq i_x - 1}$ must be set to false by ϕ. Therefore the clause $Q_{y \leq i_x} \Rightarrow Q_{x \leq i_x - 1}$ is not satisfiable by ϕ, which is a contradiction.

(iii) Let $x = 0$ is an equation in W. By construction of the Horn clauses, $Q_{x \leq 0}$ is set to true in ϕ.

If m is the number of equations in W, then the construction above yields a set of Horn clauses with cardinality at most $3mn + n^2$ (over n^2 variables). Since the satisfiability problem of a set of Horn clauses is solvable in polynomial time [5], we get

Lemma 17. *Let $X = \{x_1, x_2, \ldots, x_n\}$ be a set of variables over non-negative integers, and W be a set of equations, in which each equation is in one of the following forms:*

$$\begin{aligned} x_l &= max(x_i, x_j) \\ x_i &= x_j + 1 \qquad\qquad x_i,\ x_j,\ x_l \in X, \\ x_i &= 0 \end{aligned}$$

Solvability of W can be checked in polynomial time.

Theorem 18. *The unification problem in the single-action trace algebra can be solved in polynomial time.*

5 Future Work

The astute reader would have observed that there is a gap in the complexity bounds for unification modulo trace algebra — the lower bound is co-NP-hard and the upper bound is DEXPTIME. Future work, from the complexity point of view, should concentrate on eliminating this gap. From the practical point of view, there are several directions we plan to explore:

1. Allow recursion in the specification of processes. This could turn out to be difficult, but even partial results would be immensely useful.

2. Consider cases where the actions satisfy additional constraints.
3. Consider other equivalences such as *branching bisimulation* [13].

Acknowledgments: We thank Siva Anantharaman, Franz Baader, Rob van Glabbeek, Ralf Treinen and the referees for their comments and suggestions.

References

1. A. Aiken and D. Kozen and M. Vardi and E. Wimmers. The Complexity of Set Constraints. *Proceedings of Computer Science Logic,* (E. Börger, Y. Gurevich and K. Meinke, eds.) 1993, 1–17.
2. F. Baader and K.U. Schulz. Unification in the Union of Disjoint Equational Theories: Combining Decision Procedures. *J. of Symbolic Computation,* 21:211–243, 1996.
3. J.C.M. Baeten and W.P. Weijland. *Process Algebra.* Cambridge Tracts in Theoretical Computer Science, Vol. 18, Cambridge University Press, Cambridge, England.
4. J. Doner. Tree acceptors and some of their applications. *Journal of Computer and System Sciences* 4(3):406–451, 1970.
5. W. Dowling and J. Gallier. Linear-time algorithms for testing the satisfiability of propositional Horn formulae. *Journal of Logic Programming,* 1(3):267–284, 1984.
6. Q. Guo. *Nilpotence, Bisimulation and the Unification Workbench.* PhD thesis, State University of New York at Albany, 1997.
7. M. Hennessy. *Algebraic Theory of Processes.* Foundations of Computing Series, The MIT Press, 1988.
8. R. Milner. *Communication and Concurrency.* International Series in Computer Science. Prentice Hall, 1989. SU Fisher Research 511/24.
9. P. Narendran. Unification modulo ACI + 1 + 0. *Fundamenta Informaticae* 25 (1):49–57, 1996.
10. M.O. Rabin. Weakly definable relations and special automata. In: *Mathematical Logic and Foundations of Set Theory* (Y. Bar-Hillel, ed.), North-Holland, 1970, 1–23.
11. W. Thomas. Automata on infinite objects. *Handbook of Theoretical Computer Science* (J. van Leeuwen, ed.), Elsevier Science Publishers, 1990, 133–191.
12. R.J. van Glabbeek. The linear time—branching time spectrum. Technical Report CS-R9029, Computer Science Department, CWI, Centre for Mathematics and Computer Science, Netherlands, 1990.
13. R.J. van Glabbeek and Peter Weijland. Branching Time and Abstraction in Bisimulation Semantics. *Journal of the ACM* 43 (3), May 1996, 555–600.