New Bounds in Parallel Unification

Marco Bellia and M. Eugenia Occhiuto
Dipartimento di Informatica,
Università di Pisa
e-mail: occhiuto@di.unipi.it
Technical Report Univ. di Pisa, Dip. di Informatica TR-01-03

January 23, 2001

Abstract

$\{\mathcal{MGU}$ *(k-depth)*$\}$ is the family of problems, which contains, for each natural k, $\mathcal{MGU}$ restricted to inputs with depth no greater than k. $\{\mathcal{MGU}$ *(k-breadth)*$\}$ is the family which, for each natural k, contains $\mathcal{MGU}$ restricted to inputs with breadth no greater than k. We prove[1] that, for each $k>1$, $\mathcal{MGU}$ *(k-depth)* $\approx_{log}$ $\mathcal{MGU}$, and also $\mathcal{MGU}$ *(k-breadth)* $\approx_{log}$ $\mathcal{MGU}$. This shows that bounds either on the depth or on the breadth of the input do not significantly speed up the computation of unification on PRAM machines. The paper concludes with a look at the design of parallel unification algorithms, and at unification on *flat inputs*. As a final contribution, we introduce a new algorithm which improves the performance of known algorithms for parallel unification and also computes in polylog PRAM Time on flat inputs.

1 Introduction

Unification [Rob65] is used in a great variety of fields within computer science and automated reasoning (see [Kni89, Sie90] for a survey). The reason for such popularity lies in the fact that almost all systems are enhanced by the use of a small amount of deduction, and unification is fundamental to this process [JJ98, BM99, Bod99, IS99]. Because of both its practical and theoretical interests, extensive research [Rob79, Hue76, Kir90, Bau93, BS98, BO99] has been focused on the definition of good unification algorithms. Linear and almost linear, in both space and time, sequential algorithms were defined [Rob76, Hue76, Bax76, PW78, MM82, Col82, CB83, Muk83, Jaf84, RP90]. The advent of parallel processing led to the study of unification on parallel machines [Rob85, MO84]. In [Yas84, DKM84] unification was proved to be *LogSpace* complete for $\mathcal{P}$, which means that there are inputs where parallel unification runs not significantly faster than serial ones. This raises the interest in investigating the classes (of inputs) of the $\mathcal{MGU}$ problem [Yas84, AM85, DKS86, VS86, HJ87, KR89, Bar90, Cla91, BO] which are easily solvable using parallel computation and in the design of algorithms which behave well for such classes. With this aim, we consider two measures, *breadth* and *depth*, which are complementary in the size of the inputs of $\mathcal{MGU}$.

We define the family $\{\mathcal{MGU}$ *(k-breadth)*$\}$ of problems we obtain by restricting, for each natural k, the problem $\mathcal{MGU}$ to inputs with *breadth* superiorly bound by k. The

[1] $\approx_{log}$ denotes equivalence in *LogSpace* reduction

problems of such a family are naturally included within each other, for increasing k, and all are included in $\mathcal{MGU}$. For $k = 1$, $\mathcal{MGU}$*(k-breadth)* is *monadic unification* and was proved in $\mathcal{NC}$ [AM85]. For $k > 1$ we prove that $\mathcal{MGU}$*(k-breadth)* is *LogSpace* equivalent to $\mathcal{MGU}$. Hence, each problem $\mathcal{MGU}$*(k-breadth)* is $\mathcal{P}$-Complete when $k > 1$. We study the family $\{\mathcal{MGU}$*(k-depth)*$\}$ where, for each natural k, the problem $\mathcal{MGU}$*(k-depth)* is $\mathcal{MGU}$ restricted to inputs which have an unbound breadth but a *depth* which is superiorly bound by k. We prove that for $k = 1$, $\mathcal{MGU}$*(k-depth)* is reducible to the *transitive closure* problem, hence is in $\mathcal{NC}$. While, for $k > 1$, we prove that $\mathcal{MGU}$*(k-depth)* is *LogSpace* equivalent to $\mathcal{MGU}$. This shows that *depth* does not influence the complexity of $\mathcal{MGU}$ and no relevant advantages can be foreseen by limiting the depth of terms involved in the computation. As a final contribution, we consider these results with respect to the algorithms for parallel unification discussed in the literature [DKM84, Yas84, HJ87, Rob85, MMR86, VS86, KR89, Bar90, Cla91]. Most of them do not include reasonable mechanisms to deal with depth, so they have a complexity that is linear with the input depth. We show a version of the currently best known algorithm [DKM84, Yas84, KR89] and discuss the relevance of *propagation closure* in dealing with inputs of unbound depth. Then, we consider inputs with flat terms, i.e. terms of unbound breadth but of depth that is bound by a constant (greater than 1): in *LogSpace* we can always reduce $\mathcal{MGU}$ inputs to equivalent inputs of this sort. We show that for such inputs none of the known $\mathcal{MGU}$ algorithms computes in polylog PRAM Time. We call *flat inputs* the class of inputs with flat terms, which are not computable in polylog, but may be obtained by *LogSpace* reduction of inputs that are computable in polylog. Then, we consider the analogue where the role of depth and breadth are inverted in the input bounds: in *LogSpace* we can always reduce $\mathcal{MGU}$ inputs to equivalent inputs of unbound depth but of a breadth which is bound by a constant (greater than 1). Nevertheless such a reduction does not significantly affect the performance of the best algorithms. On the one hand, such results imply that input depth is less critical than input breadth, and on the other hand, they suggest solutions to deal with flat terms. We conclude, introducing a new algorithm which is a variant of the algorithms in [DKM84, KR89] and extends the classes of inputs which are computable in polylog [KR89] with the class of flat inputs.

The next section contains the main results and self-contained definitions of the basic ideas involved in the paper. Definitions are adapted from [Ede85, KR89, JK91, BO95]. Readers who are familiar with unification, the unification problem, and valid and unification closure relations may skip this section and refer to these definitions later, if necessary. Section 3 compares the formulation we adopt for unification with some others, encountered in the literature, proving the latter ones *LogSpace* reducible to the former. Section 4 introduces the family $\{\mathcal{MGU}$*(k-breadth)*$\}$ and contains the main results regarding its complexity. Section 5 is devoted to the family $\{\mathcal{MGU}$*(k-depth)*$\}$. Section 6 discusses the behavior of parallel unification algorithms on inputs with unbound depth. Section 7 discusses unification on inputs with unbound breadth, and introduces the class of flat inputs. Section 8 contains the new algorithm, and Section 9 concludes the paper.

2 Preliminaries

Unification is a binary, commutative, operation that computes the most general *substitution* making the two terms equal [Rob65]. A *term* is either a symbol of $\mathcal{X}$ or a string $f(t_1, ..., t_n)$, provided $t_1, ..., t_n$ are $n \geq 0$ terms and f is a symbol of Σ with $\mathcal{A}(f) = n$. When $n = 0$, the term is f and is called *constant*. $\mathcal{X}$ is the possibly denumerable set of symbols, called *variables*, and Σ is the denumerable set of symbols, called *constructors*, with an associated *arity*, given by function $\mathcal{A}$. We assume $\mathcal{X} \cap \Sigma = \emptyset$. The set of all terms is denoted $\Sigma(\mathcal{X})$. Let t be a term then the occurence set, $\Im(t)$, is the smallest set which contains Λ, such that $t|_\Lambda$=t, and if $p \in \Im(t)$ and $t|_p = f(t_1, ..., t_n)$ then $p.i \in \Im(t)$ for all $1 \leq i \leq n$. A substitution is a function ϕ, from variables into terms, which maps identically, except for a finite set, and is univocally extended to an endomorphism, $endo(\phi)$, onto $\Sigma(\mathcal{X})$. Given an unordered pair $\{t_1, t_2\}$ of terms, unification, denoted $mgu\{t_1, t_2\}$, computes the most general[2] substitution ϕ such that: $endo(\phi)(t_1) = endo(\phi)(t_2)$. Substitution ϕ is also called *unifier* of $\{t_1, t_2\}$ and, if ϕ is not the over defined $\top$, then t_1, t_2 are said to be unifiable in-between. In addition to strings, useful representations for inputs are equations, forests and labeled *dags*. A dag, for terms in $\Sigma(\mathcal{X})$, is an acyclic graph $\mathcal{D}_{\mathcal{V},\mathcal{E}}$: $\mathcal{V}$ is a finite set of *vertices* with a symbol in Σ, associated with function $\mathcal{L}$, and $\mathcal{E}$ is a finite set of labeled *edges*. An edge is a triple (u, v, i): u, v are source and target verteces respectively, and i the label. The set of roots, i.e. vertices with indegree 0, is denoted $\mathcal{V}_0$. Strings and dags, for $\Sigma(\mathcal{X})$, are related to one another through the function $\mathcal{Q}$ defined as follows.

$$\mathcal{Q}(u) = \begin{cases} X & if \;\; \mathcal{L}(u) = X \in \mathcal{X} \\ f(t_1, ..., t_n) & if \;\; \mathcal{L}(u) = f \in \Sigma \wedge \mathcal{A}(f) = n \geq 0 \wedge \\ & \quad (\forall\, (u, v_i, i) \in \mathcal{E})\, t_i = \mathcal{Q}(v_i) = \mathcal{Q}(u)|_i \end{cases}$$

Then[3], $\vec{\mathcal{Q}}(\mathcal{D}_{\mathcal{V},\mathcal{E}})$=$\{\mathcal{Q}(r) | r \in \mathcal{V}_0\}$. A two-rooted dag $\mathcal{D}_{\mathcal{V},\mathcal{E}}$is a representation of $\{t_1, t_2\}$, where $t_1 = \mathcal{Q}(u)$, $t_2 = \mathcal{Q}(v)$ and $\mathcal{V}_0 = \{u, v\}$. Some useful measures are the *size*, which is $|\mathcal{V}| + |\mathcal{E}|$, the *depth*, which is the length, i.e. the number of edges of the longest path from a root to a leaf, and the *breadth*, which is the maximum outdegree of the vertices. Several different labeled dags can represent the same (set of) term(s). All such dags have the same depth, the same breadth but may differ from one another in the size, since some vertices may be collapsed together [BO]. In the paper we limit our considerations to *simple dags*, where only vertices labeled with the same variable, are collapsed. In studying unification, some useful relations on dag vertices are:

Definition 1 (unification closure, valid relations) . Let $\mathcal{D}_{\mathcal{V},\mathcal{E}}$ be a dag.
A homogeneous relation $\mathcal{H}$ is an equivalence on $\mathcal{V}$such that if $u, v \in \mathcal{H}$ then $\mathcal{L}(u) = \mathcal{L}(v)$.

[2](Idempotent) Substitutions are partially ordered and embedded into a semilattice [Ede85] having identity ϵ, i.e. $endo(\epsilon)(t) = t$ forall terms t, as the bottom and the over defined substitution $\top$, i.e. $endo(\top)(t) = null$, as the top [BO95]. Term *null* is a special, additional, constant term for which $mgu\{null, t\} = \top$, for all terms t.

[3]Not all dags are dags for terms: especially $\mathcal{Q}$ is not defined on dags with vertices having more than one outgoing edge, labeled in the same way. For a dag of this sort we cannot associate a term. However, the definition of $\mathcal{Q}$ also requires, that outgoing edges are labeled by sequential integers. This is not essential and could be relaxed requiring only a bijective function from labels to integer.

An acyclic relation $\mathcal{A}$ is an equivalence on $\mathcal{V}$ such that $\mathcal{D}_{\mathcal{V},\mathcal{E}}/\mathcal{A}$ is an acyclic graph [4].
The unification closure $\mathcal{UC}(\mathcal{R})$, is the finest equivalence relation containing $\mathcal{R}$ and such that for all vertices u, v for which $\{u,v\} \in \mathcal{UC}(\mathcal{R})$ holds then $\{u_i, v_i\} \in \mathcal{UC}(\mathcal{R})$ also holds for all pairs u_i, v_i of corresponding sons of u, v.
A valid relation is an equivalence that is both homogeneous and acyclic.

Section 6 introduces a unification algorithm (see Figure 1) which is based on the computation of the unification closure. The algorithms, defined in the literature, for the computation of mgu by sequential machines [Rob76, Hue76, Bax76, PW78, MM82, CB83, Muk83, Jaf84, MMR86, RP90, BO99] compute in a similar way but exploit different algebraic frameworks and use slightly different presentations of substitutions and hence, of the unifier.

Definition 2 (Unification problem) The unification problem $\mathcal{MGU}$ is the following decision problem:

$$\mathcal{MGU} = \{\{t_1, t_2\} \mid mgu\{t_1, t_2\} \neq \top\}$$

A solution for $\mathcal{MGU}$ is a procedure which answers $\{t_1, t_2\} \in \mathcal{MGU}$ for each input $\{t_1, t_2\}$ whose mgu is not the over defined $\top$ and $\{t_1, t_2\} \notin \mathcal{MGU}$ otherwise.

Theorem 1 *[PW78] Let u, v be a pair of vertices in $\mathcal{D}_{\mathcal{V},\mathcal{E}}$, and $\mathcal{R}$ be the equivalence relation containing only the pair $\{u, v\}$. Let $t_1 = \mathcal{Q}(u)$ and $t_2 = \mathcal{Q}(v)$. Then $\{t_1, t_2\} \in \mathcal{MGU}$ if and only if $\mathcal{UC}(\mathcal{R})$ is a valid relation.*

Proof: If $\mathcal{R}$ can be extended into a valid relation $\mathcal{UC}(\mathcal{R})$, let τ be the homomophism from $\mathcal{D}_{\mathcal{V},\mathcal{E}}$ onto $\mathcal{D}_{\mathcal{V},\mathcal{E}}/\mathcal{UC}(\mathcal{R})$ such that $\tau(u_i) = \tau(v_i)$ *iff* $\{u_i, v_i\} \in \mathcal{UC}(\mathcal{R})$. Then, ϕ such that $\phi(\mathcal{Q}(u_i)) = \mathcal{Q}(\tau(u_i))$, for all $u_i \in \mathcal{V}$ for which $\mathcal{Q}(u_i) \in \mathcal{X}$, is a unifier of $\{t_1, t_2\}$ and is not, over defined, $\top$. For the converse, we prove that if the most general unifier is not $\top$, then the unifier induces a relation on $\mathcal{V}$ which can be extended into a valid relation for $\mathcal{D}_{\mathcal{V},\mathcal{E}}$. $\square$

All the above mentioned sequential algorithms for mgu are also procedures for $\mathcal{MGU}$. However, in order to solve $\mathcal{MGU}$, step 5 of Algorithm 1 can be omitted, and if we are interested in *infinite unification* [Col82, Jaf84, KR89] step 4 must also be removed (see Section 6).

3 Unification on sets

Unification, as originally defined [Rob65], is also called *binary* unification since it applies to unordered pairs of terms. However, in some contexts, for instance *factoring* [CL74], unification applies to finite sets of terms. Unification on sets is denoted $\mathcal{MGU}_S$ and can be defined in the following straightforward way: $mgu(T_1 \cup T_2) = \phi \circ mgu(endo(\phi)(T_2))$ for any pair of nonsingleton sets T_1, T_2, where $\phi = mgu(T_1)$, $endo(\phi)(T) = \{endo(\phi)(t) | t \in T\}$ and $\circ$ is the usual function composition. Since $m-1$ repeated applications of

[4] $\mathcal{D}_{\mathcal{V},\mathcal{E}}/\mathcal{R}$ is the graph resulting from $\mathcal{D}_{\mathcal{V},\mathcal{E}}$ modulo the relation $\mathcal{R}$, see Section 8 for a definition.

binary unification provide a solution to the computation, when $T_1 \cup T_2$ has m terms, the resulting algorithm should require $m * \mathcal{F}(h)$ PRAM Time, where $\mathcal{F}(h)$ is the PRAM Time complexity of binary unification and h is the size bound of the terms in the set. We apply *LogSpace* reduction to show that unification on sets[5] has the same complexity of (binary) unification and can be computed with PRAM Time bound $\mathcal{F}(n)$, where n is the sum of the size of the terms in the set. A problem A is said to be *LogSpace* reducible to a problem B if there is a function f which transforms each input of A into an input of B and f is in *LogSpace* .

Theorem 2 $\mathcal{MGU}_S \approx_{log} \mathcal{MGU}$.

Proof: $\mathcal{MGU}$ is a special case of unification on sets, hence its reducibility to unification on sets is immediate. For the converse, we show how any n-rooted dag for $\mathcal{MGU}_S$ can be reduced in *LogSpace* into an equivalent input for $\mathcal{MGU}$. Let $\mathcal{D}_{\mathcal{V},\mathcal{E}}$ be the n-rooted $(n > 0)$ simple dag representing input $I = \{t_1, \dots, t_n\}$, and $\{u_1, \dots, u_n\}$ be its roots. We construct the 2-rooted simple dag $\mathcal{D}_{\mathcal{V}',\mathcal{E}'}$ such that:

$\mathcal{V}' = \mathcal{V} \cup \{u_0, v_0, v_x\}$

$\mathcal{E}' = \mathcal{E} \cup \{(u_0, u_i, i), (v_0, v_x, i) | i \in [1, n]\}$

where, $\mathcal{L}(u_0) = \mathcal{L}(v_0) = f$ and $\mathcal{L}(v_x) = X$, f being a n-constructor and X a variable symbol, not occurring in I, for terms in $\Sigma(\mathcal{X})$[6]. This construction can be made in constant space and $O(n)$ RAM Time since, for each of the n roots, we must maintain space only for: the selected root u_i and the three vertices u_0, v_0, v_x. Finally, let ϕ be a unifier of I, t be the term $t = endo(\phi)(t_1) = \dots = endo(\phi)(t_n)$, and ϑ be the substitution which maps identically except for $\vartheta(X) = t$. Then, $\phi \circ \vartheta$ is a unifier of $\{\mathcal{Q}(u_0), \mathcal{Q}(v_0)\}$. □

Theorem 2 shows that extending unification to sets has no relevance from the complexity viewpoint. The above proof provides a simple way to extend algorithms for ordinary unification to the computation, with PRAM Time bound $\mathcal{F}(n)$, of unification on sets. Complexity $\mathcal{F}(n)$ is in fact linear with n, since unification is unlikely in $\mathcal{NC}$. $\mathcal{NC}$ [Joh90] is a basic complexity class for PRAM. It consists of all those decision problems that are computable on PRAM that has a polylogaritmic bound, $O(\log^{O(1)} n)$, on the running time and a polynomial bound, $O(n^{O(1)})$, on the used processors, where n is the input size. We have the following inclusion hierarchy [Joh90]: *LogSpace* $\subseteq \mathcal{NC} \subseteq \mathcal{P}$. Functions depth, breadth, $\mathcal{Q}$ and the reduction in Theorem 2, are all in *LogSpace* , hence they are in $\mathcal{NC}$, and fast parallel programs can be easily constructed for them [Dek81, QD84, CJC82]. The same holds for each elementary step (i.e. 1, 2.a-c, 3, 4, 5) of the Algorithm 1, but not for the loop in step 2, see section 6. Let B be a problem in $\mathcal{NC}$, if A is *LogSpace* reducible to B, then A too is in $\mathcal{NC}$. The same obviously holds for B in $\mathcal{P}$. A $\mathcal{P}$-complete problem is such that all problems in $\mathcal{P}$ are *LogSpace* reducible to it. The *Monotone Circuit Value*, $\mathcal{MCV}$ [Gol77], is in the class of $\mathcal{P}$-Complete problems.

Theorem 3 *[DKM84, Yas84] $\mathcal{MGU}$ is in $\mathcal{P}$-Complete.*

[5]Unification on sets is less critical than *set unification* which is the extension of the binary unification studied in [KP86] and proved NP-complete by reducing in *LogSpace* , unification, with commutative-associative-idempotend constructors, to it

[6]Without loss of generality, we can assume $\mathcal{X}$ and Σ be both denumerable infinite sets and, for each nonnegative i, Σ contains at least one constructor f such that $\mathcal{A}(f) = i$

Proof: $\mathcal{MCV}$ is *LogSpace* reducible to the $\mathcal{MGU}$ problem. □

In this section we have shown how unification on sets can be reduced, in *LogSpace*, to binary unification and hence solved using procedures for binary unification. Moreover, in [Sie90], unification is defined on sets of sets of terms. It too is only a syntactic extension and can be easily reduced to binary unification using a reduction strongly similar to the one introduced in proof of Theorem 2. We leave it to the readers. So, in the next sections, we will limit our consideration to binary unification. As a final remark, we recall that, by introducing suitable algebraic frameworks for substitutions[7], [Ede85, Pal90] new algorithms have been found for unification on sets which are as complex as the one discussed here.

4 The classes $\mathcal{MGU}$*(k-breadth)*

We show that the general $\mathcal{MGU}$ problem can be reduced to each of the sub-problems obtained by restricting input breadth to be superiorly bound by a constant $k \geq 2$.

Definition 3 ($\mathcal{MGU}$*(k-breadth)*) For each natural $k(>0)$, $\mathcal{MGU}$*(k-breadth)* is the unification problem whose inputs have breadth no greater than k.

We introduce a family of functions τ_k, indexed by the natural $k \geq 2$, which map $\mathcal{MGU}$ inputs into inputs of $\mathcal{MGU}$*(k-breadth)*.

Definition 4 (τ_k) Let $\{t_1, t_2\}$ be an input of $\mathcal{MGU}$. Let $\{C_i\}$ be a family of 2-arity constructors of Σ, such that $C_i = C_j$ only if $i = j$, and τ_k is the function defined recursively as follows:

- $\tau_k(a) = a$ for each variable or constant
- $\tau_k(f(s_1, \ldots, s_m)) = f(\tau_k(s_1), \ldots, \tau_k(s_m))$ for $m \leq k$
- $\tau_k(f(s_1, \ldots, s_m)) = C_1(\iota(f), C_2(\tau_k(s_1), (\ldots, C_m(\tau_k(s_{m-1}), \tau_k(s_m)) \ldots)))$ otherwise

where ι is an injective mapping from constructors into constants, not occurring in the terms t_1, t_2.

Lemma 1 *For each k, function τ_k is in LogSpace*

Proof: By induction on the structure of the term we can prove that τ_k can be computed in constant space and $O(n)$ RAM Time, n being the size of the terms. It is straightforwardly when τ_k applies to either a variable or a constant. In other cases, τ_k can be computed separately, on the subdags of each s_i. Hence, when vertex v has outdegree $m > k$, the computation requires the introduction of a new vertex w_v, labeled C_1, and a new vertex w_f, labeled $\iota(\mathcal{L}(v))$. Moreover, for i ranging in $[1..m-1]$, a new vertex w_i, labeled C_{i+1}, is associated with the i_{th} son v_i of v. Vertex v is marked removed. When

[7] In [Pal90] composition of substitutions is made associative and also commutative, so that unification on set can be now expressed by: $mgu(\{t\} \cup T_1 \cup T_2) = mgu(\{t\} \cup T_1) \uparrow mgu(\{t\} \cup T_2)$, being $\uparrow$ the new composition operator

all vertices have been considered and possibly marked, the edges $(w_v, w_f, 1)$, $(w_v, w_1, 2)$ and, for i ranging in $[1..m-1]$, the pairs of edges $(w_i, \overline{v}_i, 1)$, $(w_i, w_{i+1}, 2)$ are introduced, $\overline{v}_i$ being vertex v_i if v_i is not marked, otherwise it is the new vertex w_{v_i}. The removing of all marked vertices completes the construction. $\square$

Lemma 2 *Let $I = \{t_1, t_2\}$ be input of $\mathcal{MGU}$. Then,*[8]

$$\tau_k(I) \text{ is input of } \mathcal{MGU}\text{(k-breadth)} \tag{1}$$

$$I \in \mathcal{MGU} \text{ iff } \tau_k(I) \in \mathcal{MGU}\text{(k-breadth)} \tag{2}$$

Proof: (1) straightforwardly follows from the construction of the terms.
(2) Let $\mathcal{D}_{\mathcal{V},\mathcal{E}}$ and $\mathcal{D}_{\mathcal{V}',\mathcal{E}'}$ be the simple dags representing I and $\tau_k(I)$ respectively. The two dags differ in relation of the subdags representing subterms of form $f(s_1, ..., s_n)$ for which $\mathcal{A}(f) > k$, if any. Hence, a dag homomorhism ρ exists from $\mathcal{D}_{\mathcal{V}',\mathcal{E}'}$ to $\mathcal{D}_{\mathcal{V},\mathcal{E}}$: ρ maps vertices identically except for the ones labeled either $\iota(f)$ or C_i, of each $\tau_k(f(s_1, ..., s_n))$. Such vertices are mapped together onto the vertex which is the root of the corresponding $f(s_1, ..., s_n)$. By Theorem 1, $I \in \mathcal{MGU}$ implies the existence of a valid relation $\mathcal{UC}(\mathcal{R})$ for $\mathcal{D}_{\mathcal{V},\mathcal{E}}$. Then, $\mathcal{UC}(\mathcal{R}_w)$ such that: $\{u, v\} \in \mathcal{UC}(\mathcal{R}_w)$ *iff*

either $\{u, v\} \in \mathcal{UC}(\mathcal{R})$ *and* ρ maps identically on both u and v,

or $\{\rho(u), \rho(v)\} \in \mathcal{UC}(\mathcal{R})$, $\rho(u) \neq u$, $\rho(v) \neq v$, *and* $\mathcal{L}(u) = \mathcal{L}(v)$,

is a valid relation for $\mathcal{D}_{\mathcal{V}',\mathcal{E}'}$. The converse is straightforward using similar arguments. $\square$

Theorem 4 *For each natural $k \geq 2$, $\mathcal{MGU} \approx_{log} \mathcal{MGU}$(k-breadth).*

Proof: $\mathcal{MGU}$*(k-breadth)* is included in $\mathcal{MGU}$ and $\mathcal{MGU}$ is *LogSpace* reducible to $\mathcal{MGU}$*(k-breadth)* by Lemma 1 and by Lemma 2 $\square$

This reducibility result says that without loss of generality, we can always restrict unification to terms having constructors of arity not greater than k for $k \geq 2$. For $k = 2$, the space of terms is reduced to the one used in [Koz77, DSR80] in investigating complexity of the congruence closure problem, CC. [KR89] used $\mathcal{MGU}$*(2-breadth)* in investigating the relations about CC and $\mathcal{MGU}$. For $k = 1$, we have monadic unification, $\mathcal{MGU}$*(mon)*. In [AM85], $\mathcal{MGU}$*(mon)* has been proved in $\mathcal{NC}$ and a very fast procedure, on PRAM machines, has been defined. Hence, $k = 1$ becomes the superior bound for the problems in $\{\mathcal{MGU}$*(k-breadth)*$\}$, which are in $\mathcal{NC}$, and a bound for the breadth of inputs on which unification can be computed in polylog PRAM Time.

5 The classes $\mathcal{MGU}$*(k-depth)*

As far as function τ_k applies, by reducing the breadth, the input depth increases. Since depth can be considered as the relevant bound for the PRAM Time required to traverse input [HJ87], the previous result on the complexity of $\mathcal{MGU}$*(k-breadth)* should not be surprising. However, we show that a similar result holds also when the input depth is superiorly bound.

[8] τ_k is straightly extended to inputs by putting $\tau_k(\{t_1, t_2\}) = \{\tau_k(t_1), \tau_k(t_2)\}$.

Definition 5 ($\mathcal{MGU}$*(k-depth)*) For each natural $k(>0)$, $\mathcal{MGU}$*(k-depth)* is the unification problem whose inputs have depth no greater than k.

Let t be a term and, for $k>1$, $I_k(t)=\{p_1,...,p_n\}$ be the set of occurrences of all non-variable subterms of t whose height is a multiple of $(k-1)$. Occurrences are partially ordered: $p<q$ *iff* p is a prefix of q. Given a set I of occurrences, we call maximal of p any occurrence q in I such that $p<q$ and if both $p<r$ and $r<q$ hold then $r=q$. Operator $-$ is prefix dropping, which is always defined when $p<q$, and it is such that if $q-p=r$ then $q=p.r$. In particular, for each q, $q-\Lambda=q$.

Let $X_1,...,X_n$ be n distinct variables not occurring in t and f_{μ_k} be the injective function which associates the i-th variable X_i with the i-th occurrence p_i.

Definition 6 (μ_k **and** $\vec{\mu}_k$) For $p\in\{\Lambda\}\cup I_k(t)$, $\mu_k(t|_p)$ is the term resulting from $t|_p$ replacing each subterm at the occurrence p_j-p with $f_{\mu_k}(p_j)$, for each p_j which is a maximal of p. Function $\vec{\mu}_k$ extends μ_k to inputs of $\mathcal{MGU}$ in the following way. Let $\{t_1,t_2\}$ be an input. Let $I_k(t_1)=\{p_{1_1},...,p_{1_{n_1}}\}$ and $I_k(t_2)=\{p_{2_1},...,p_{2_{n_2}}\}$. Let $X_1,...,X_{n_1}$ and $Y_1,...,Y_{n_2}$ be n_1+n_2 variables occurring neither in t_1 nor in t_2, and $\mathcal{C}$ be a $(n+m+1)$-arity constructor. Then, $\vec{\mu}_k(\{t_1,t_2\})=\{t'_1,t'_2\}$ where:

$$\begin{aligned} t'_1 &= \mathcal{C}(\mu_k(t_1),\mu_k(t_1|_{p_{1_1}}),...,\mu_k(t_1|_{p_{1_{n_1}}}),Y_1,...,Y_{n_2}) \\ t'_2 &= \mathcal{C}(\mu_k(t_2),X_1,...,X_{n_1},\mu_k(t_2|_{p_{2_1}}),...,\mu_k(t_2|_{p_{2_{n_2}}})) \end{aligned}$$

Lemma 3 *For each k, function* $\vec{\mu}_k$ *is in LogSpace*

Proof: $\vec{\mu}_k$ is computed in constant space, provided each vertex has its associated height. In fact, let $\mathcal{D}_{\mathcal{V},\mathcal{E}}$ be the dag for input $\{t_1,t_2\}$. Then, with each vertex $v\in\mathcal{V}$ whose height is a multiple of $k-1$ (and not labeled by a variable), we associate a new vertex x_v, labeled by a new variable, and a distinctive index i_v. Let $\mathcal{V}'$be the set of vertices which contains all vertices in $\mathcal{V}$, two new vertices r_1 and r_2, both labeled by constructor $\mathcal{C}$, and all the vertices x_v, introduced as above. Let $\mathcal{E}'$ be the set of edges which contains all edges in $\mathcal{E}$ but each edge (u,v,i) whose target v has an associated x_v. Each of them is replaced, in $\mathcal{E}'$, by edge (u,x_v,i), and by the two additional edges: (r_1,v,i_v) and (r_2,x_v,i_v) if v is a descendent of the first root of $\mathcal{D}_{\mathcal{V},\mathcal{E}}$, (r_2,v,i_v) and (r_1,x_v,i_v) otherwise. Then, $\mathcal{D}_{\mathcal{V}',\mathcal{E}'}$ is the dag for $\vec{\mu}_k(\{t_1,t_2\})$. Height is in *LogSpace* [Bor77] and can be computed in polylog PRAM Time using the doubling technique [J92] □

Lemma 4 *Let* $\{t_1,t_2\}$ *be an input of* $\mathcal{MGU}$*. Then,*

$$\vec{\mu}_k(\{t_1,t_2\}) \textit{ is input of } \mathcal{MGU}\textit{(k-depth)} \quad (1)$$

$$\{t_1,t_2\}\in\mathcal{MGU} \text{ iff } \vec{\mu}_k(\{t_1,t_2\})\in\mathcal{MGU}\textit{(k-depth)} \quad (2)$$

Proof: (1) straightforwardly follows from the construction of the terms.
(2) Let $\mathcal{D}_{\mathcal{V},\mathcal{E}}$ and $\mathcal{D}_{\mathcal{V}',\mathcal{E}'}$ be the simple dags representing $\{t_1,t_2\}$ and $\vec{\mu}_k(\{t_1,t_2\})$ respectively. The two dags differ in terms of the roots of $\mathcal{D}_{\mathcal{V}',\mathcal{E}'}$ and in terms of the roots of the subdags representing the subterms $t_j|_{p_{j_i}}$, for $j\in[1..2]$ and $i\in[1..n_j]$, if any,

and the corresponding new variables $f_{\mu_k}(p_{j_i})$. Let $\mathcal{D}_{\mathcal{V}'',\mathcal{E}''}$ be the dag which results in removing the roots from $\mathcal{D}_{\mathcal{V}',\mathcal{E}'}$. Then, a dag homomorhism ρ exists from $\mathcal{D}_{\mathcal{V}'',\mathcal{E}''}$ to $\mathcal{D}_{\mathcal{V},\mathcal{E}}$: it maps identically except for the root of each subterm $\mu_k(t_j|_{p_{j_i}})$ and the vertex labeled, correspondingly, by variable $f_{\mu_k}(p_{j_i})$. For $j \in [1..2]$ and $i \in [1..n_j]$, ρ maps the vertices of each of the pairs onto the vertex of $\mathcal{D}_{\mathcal{V},\mathcal{E}}$ which is the root of $t_j|_{p_{j_i}}$. By Theorem 1, $\{t_1, t_2\} \in \mathcal{MGU}$ implies the existence of a valid relation $\mathcal{UC}(\mathcal{R})$ for $\mathcal{D}_{\mathcal{V},\mathcal{E}}$. Thus, the relation U such that: $\{u, v\} \in U \mathit{iff} \{\rho(u), \rho(v)\} \in U$ is a valid relation for $\mathcal{D}_{\mathcal{V}'',\mathcal{E}''}$. Finally, by construction of $\mathcal{D}_{\mathcal{V}'',\mathcal{E}''}$, U is trivially extended to a valid relation $\mathcal{UC}(\mathcal{R}_d)$ on $\mathcal{D}_{\mathcal{V}',\mathcal{E}'}$, starting from relation $\mathcal{R}_d$ which contains the roots of $\mathcal{D}_{\mathcal{V}',\mathcal{E}'}$. The converse is straightforward using similar arguments. □

Theorem 5 *For each $k \geq 2$, $\mathcal{MGU} \approx_{log} \mathcal{MGU}$(k-depth).*

Proof: $\mathcal{MGU}$*(k-depth)* is included in $\mathcal{MGU}$ and $\mathcal{MGU}$ is *LogSpace* reducible to $\mathcal{MGU}$-*(k-depth)* by Lemma 3 and by Lemma 4 □

This reducibility result says that, without loss of generality, we can always restrict the unification problem to inputs which have a depth bound by k, for arbitrary $k \geq 2$. For $k = 1$ we have $\mathcal{MGU}$*(1-depth)* which is in $\mathcal{NC}$.

Theorem 6 *[BO] $\mathcal{MGU}$(1-depth) is in $\mathcal{NC}$.*

Proof: Since only roots have outdegrees different from 0, only one step of propagation from roots to sons is needed and then, one step of transitive closure on the sons (see Section 6) is enough to compute $\mathcal{UC}(\mathcal{R})$. □

Hence, $k = 1$ is the superior bound for the problems in $\{\mathcal{MGU}$*(k-depth)*$\}$ which are in $\mathcal{NC}$ and is a bound for the depth of inputs on which unification can be computed in polylog PRAM Time.

6 Algorithms for parallel unification

Figure 1 shows an algorithm for unification. Algorithm 1 is based on Theorem 1 and uses the algebraic structures introduced in Definition 1. The abstract nature of the structures involved in it, makes Algorithm 1 adequate for PRAM machines and easily comparable with other algorithms which have been proposed for parallel unification [DKM84, Yas84, HJ87, Rob85, VS86, Bar90, MMR86].

In particular, step 2 computes the unification closure, $\mathcal{UC}(\mathcal{R})$, of the input dag, $\mathcal{D}_{\mathcal{V},\mathcal{E}}$, starting from a relation $\mathcal{R}$ on $\mathcal{V}$: from [DKM84, KR89] we know that finitely many (at most $|\mathcal{V}| + |\mathcal{E}|$) alternations of one step of (downward) propagation and one step of (transitive) closure is enough to compute $\mathcal{UC}(\mathcal{R})$. In Algorithm 1, propagation is acquainted in step 2.b' by the binary relation, $\bigtriangledown_{\mathcal{R},\mathcal{D}_{\mathcal{V},\mathcal{E}}}$. Such a relation is defined on the unordered pairs of $\mathcal{V}^2$, and is such that: $<\{u, v\}, \{u_s, v_s\}> \in \bigtriangledown_{\mathcal{R},\mathcal{D}_{\mathcal{V},\mathcal{E}}}$ *iff* $\{u, v\} \in \mathcal{R}$ and $\{u_s, v_s\}$ is either $\{u, v\}$ or a pair of corresponding sons of $\{u, v\}$. The construction of $\bigtriangledown_{\mathcal{R},\mathcal{D}_{\mathcal{V},\mathcal{E}}}$ is trivially in $\mathcal{NC}$. Transitive closure, denoted $\uplus$, is in $\mathcal{NC}$ too [J92], and

Program Computation of $\mathcal{UC}(\mathcal{R})$ (Unification)

input dag: $\mathcal{D}_{\mathcal{V},\mathcal{E}}$; relation: $\mathcal{R} = \{\{u_0, v_0\}\}$
output relation $\mathcal{UC}(\mathcal{R})$: P

begin
1. set $S = \mathcal{R}$
2. **repeat**
   a. set $P = S$
   b'. *propagation*: set $S = \{\{u_s, v_s\} | <\{u, v\}, \{u_s, v_s\}> \in \bigtriangledown_{P,\mathcal{D}_{\mathcal{V},\mathcal{E}}}\}$
   c. *closure*: set $S = \biguplus(S)$
   until $P = S$
3. test classes of $\mathcal{UC}(\mathcal{R})$ for homogeneity
4. test the graph $\mathcal{D}_{\mathcal{V},\mathcal{E}}/\mathcal{UC}(\mathcal{R})$ for acyclicity
5. construct the substitution

end.

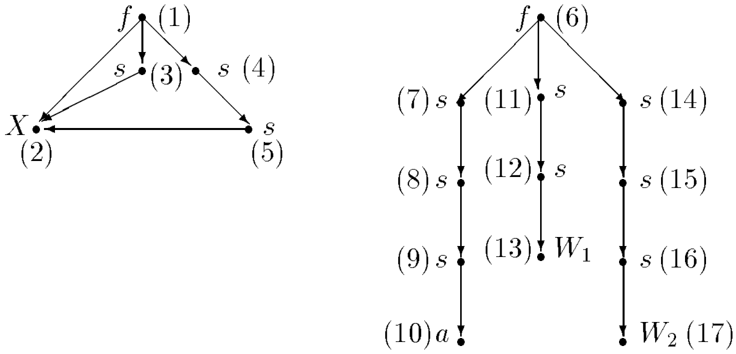
Figure 1: Algorithm 1

2. b *propagation* :set $S = \{\{u_s, v_s\} | <\{u, v\}, \{u_s, v_s\}> \in \biguplus(\bigtriangledown_{P,\mathcal{D}_{\mathcal{V},\mathcal{E}}})\}$

Figure 2: Algorithm 2: The propagation closure

it is applied in step 2.b. Next steps 3 and 4 test $\mathcal{UC}(\mathcal{R})$, as constructed in the loop of step 2, for a valid relation. In particular, step 3 checks whether a unifier exists at least for rational terms [KR89] (i.e. the occur-check must be omitted [Col82]). At the same time step 4 checks whether such a unifier involves only terms in $\Sigma(\mathcal{X})$. Both steps are in $\mathcal{NC}$, the first one, in particular, can be computed in constant space. Eventually, the last step computes the unifier, if there is one: It consists in the construction of a dag morphism from $\mathcal{D}_{\mathcal{V},\mathcal{E}}$ into $\mathcal{D}_{\mathcal{V},\mathcal{E}}/\mathcal{UC}(\mathcal{R})$ which can be computed in $\mathcal{NC}$[BO99].

Theorem 5 proves that input depth is not in itself responsible for the negative results in [DKM84, Yas84]. Moreover it suggests solutions for those parallel unification algorithms whose cost is also conditioned by the input depth. Such algorithms compute in linear PRAM Time with depth, also for those inputs for which solutions can be found in polylog. This is the case for the algorithms in [HJ87, Rob85, VS86, Bar90, MMR86], and all those for *equation solving* described in [Cla91] and those adopted in [Sha89] where unification is used to control the interaction among concurrent parallel computations. Even though such algorithms are very different from one another, they have the same drawback and share the structure of Algorithm 1. In particular, none has mechanisms which support the computation of the *propagation closure* in polylog PRAM Time. This requires in fact, the replacement of step 2.b', which computes only one step of downward *propagation*, with step 2.b in figure 2, which iterates propagation for as long as possible. For a better understanding of the problem consider the following example.



Initially:	$P =$	$\mathcal{R} =$	$\{\{1,6\}\}$
1^{st} iteration of step 2		$P^{(1)} =$	$\mathcal{R} \cup \{\{2,7\},\{3,11\},\{4,14\}\}$
2^{nd} iteration of step 2		$P^{(2)} =$	$P^{(1)} \cup \{\{2,12\},\{5,15\},\{7,12\}\}$
3^{rd} iteration of step 2		$P^{(3)} =$	$P^{(2)} \cup \{\{8,13\},\{2,16\},\{13,17\},$ $\{7,16\},\{12,16\}\}$
4^{th} iteration of step 2		$P^{(4)} =$	$P^{(3)} \cup \{\{8,17\},\{13,17\}\}$

The classes of the partition $\mathcal{UC}(\mathcal{R})$ are:
$\{[1,6],[2,7,12,16],[8,13,17],[5,15],[3,11],[4,14],[9],[10]\}$

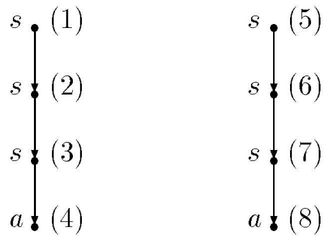
Figure 3: Computation of $\{f(X,s(X),s(s(X))), f(s(s(s(a))),s(s(W_1)),s(s(s(W_2))))\}$ with Algorithm 1

Example 1

$$\{f(X, sX, s^2X, ..., s^{n-1}X), f(s^n a, s^2W_1, s^3W_2, ..., s^nW_{n-1})\}$$

is an input scheme. For n ranging on naturals greater than 1, it is a different input for $\mathcal{MGU}$. X is a variable occurring n times in the first term, $W_1, ..., W_{n-1}$ are $n-1$ different variables occurring once in the second term and $s^i a$ stands for terms constructed by i applications of constructor s to term a. The input size has bound $O(n^2)$. Using Algorithm 1, the unification closure is computed with PRAM Time bound $O(n \log^2 n)$. Figure 3 shows the computation for $n = 3$. It is left to the readers to show that using Algorithm 2, the computation has PRAM Time bound $O(\log^k n)$, where $k < 2$ [CT95].

Algorithm 2 is due to [DKM84]. It has the same behavior as the one introduced in [Yas84], which uses hypergraph reachability for computing unification. As far as we know, it is the *best* algorithm currently proposed for parallel unification: in particular, it computes in polylog on all the classes of $\mathcal{MGU}$ inputs which are known to be solvable in $\mathcal{NC}$ [KR89].



Initially:	$P = \mathcal{R} =$	$\{\{1,5\}\}$
1^{st} iteration of step 2	$P^{(1)} =$	$\mathcal{R} \cup \{\{2,6\},\{3,7\},\{4,8\}\}$

Figure 4: Computation of $\{s(s(s(a))), s(s(s(a)))\}$ with Algorithm 2

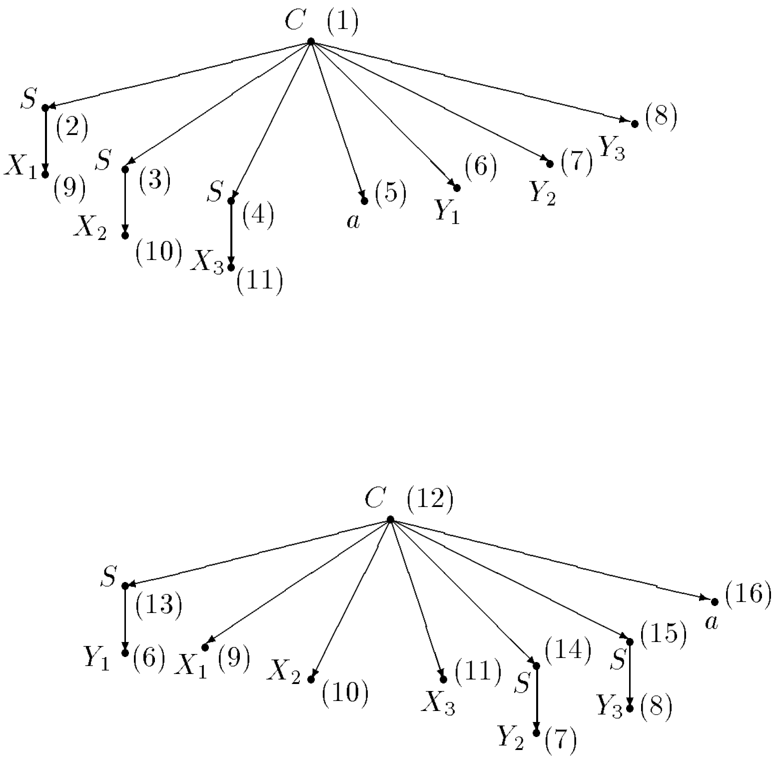
7 Unification with flat terms

Depth and breadth reductions have an obvious complementary effect on the input size: in reducing the depth we obtain a corresponding increase in the breadth of the resulting input and, conversely, in reducing the breadth. The effects that the two reductions have on the complexity of the computation appear less obvious, i.e. when we use the reduced input instead of the original one. In the previous sections we have shown that there are no advantages, for $\mathcal{MGU}$, in constraining the bound of input depth. In this section we show that depth not only does not affect the complexity of the unification, but on the contrary, it may be even better to deal with inputs of unbound depth rather than unbound breadth. This is proved by showing that Algorithm 2 computes with the original input better than with the input resulting from depth reduction. In fact, the depth reduction shift the burden of traversing the input dag from the propagation closure to the transitive closure step.

Theorem 7 *Let I be the input for $\mathcal{MGU}$ and $I_d = \vec{\mu}_k(I)$ be the reduced input for some $k > 1$. Let n be the maximum size of the path of corresponding descendants that the various propagation closure steps traverse in the computation of Algorithm 2 with input I. Then, the computation with input I_d requires additional $\lfloor n/(k-1) \rfloor$ alternate steps of one propagation closure and one transitive closure.*

Proof: The proof of Lemma 2 introduced the dag homomorphism ρ from I_d to I. According to which, for all pairs $\{u,v\}$ of vertices in I if $\{u,v\} \in \mathcal{UC}(\mathcal{R})$ then $\{u,v\} \in \mathcal{UC}(\mathcal{R}_d)$, where $\mathcal{UC}(\mathcal{R})$, $\mathcal{UC}(\mathcal{R}_d)$ are the valid relations for I and I_d respectively. By assuming n, let $\{u_n, v_n\}$ be the pair reached at the end of a path of size n. By construction of $\vec{\mu}_k(I)$, if $n > k$ then, in corresponding with vertices in I which are multiples of $k-1$, I_d has occurrences of new variables. Hence, I_d has $\lceil n/(k-1) \rceil$ occurrences of such variables, each requiring one transitive closure step to be traversed. Since they must be traversed in sequence, Algorithm 2 with input I_d requires $(n/(k-1))$ iterations of the main loop. □

Since k is a constant, theorem 7 says that the complexity of Algorithm 2 may become linear with n as long as the input depth decreases. The theorem also says that this worsening does not always happen since it depends on the effective need to



Initially:	$P = \mathcal{R} =$	$\{\{1,12\}\}$
1^{st} iteration of step 2	$P^{(1)} =$	$\mathcal{R} \cup \{\{2,13\},\{3,9\},\{4,10\},\{5,11\}$ $\{6,14\},\{7,15\},\{8,16\},\{6,9\},\{9,14\},$ $\{3,6\},\{3,14\}\}$
2^{st} iteration of step 2	$P^{(2)} =$	$P^{(1)} \cup \{\{7,10\},\{4,7\},\{4,15\},\{10,15\}\}$
3^{st} iteration of step 2	$P^{(3)} =$	$P^{(2)} \cup \{\{8,11\},\{5,8\},\{5,16\},\{11,16\}\}$

Figure 5: computation of $\vec{\mu}_k(s(s(s(a))), s(s(s(a))))$ with Algorithm 2

traverse deeper subterms. However, the following example shows that such a situation may actually occur.

Example 2

$$I = \{s^n a, s^n a)\}$$

Figure 4 shows the computation of Algorithm 2, with input I for $n = 3$. While Figure 5 shows the computation of the algorithm with the input $\vec{\mu}_k(I)$ when the depth is bound by $k = 2$. In the first case one step of propagation closure is all the algorithm has to do to compute unification, while in the second case it must alternate three steps of propagation closure and transitive closure.

The situation is quite different in the case of breadth reductions. The following theorem shows that given an input I, any breadth reduction, I_w, is computed by Algorithm 2 with the same complexity as with I.

Theorem 8 *Let I be an input for $\mathcal{MGU}$ and $I_w = \tau_k(I)$ be the reduced input, for $k > 1$. Hence, the computation of Algorithm 2 with input I and with input I_w requires exactly the same number of alternate of propagation closure and transitive closure steps.*

Proof: The two inputs may differ only for terms of form $f(s_1, \dots, s_n)$ which, for $n > k$, are replaced by $C_1(\iota(f), C_2(\tau_k(s_1), (\dots, C_n(\tau_k(s_{n-1}), \tau_k(s_n)) \dots)))$. But subterms $s_1, \dots, s_n$ in I, and $\tau_k(s_1), \dots, \tau_k(s_n)$ in I_w, are all traversed in exactly one step of propagation closure. Thus, the proof can be completed using induction on the structure of the two terms. □

Hence, we note that the complementarity of depth and breadth reductions does not hold when we consider the effects on the computation. In particular, Theorem 8 says that we can reduce as much as we like, the breadth of the input and, correspondingly increase its depth, with no effect on the parallelization of the computation process. This contrasts with what happens in reducing input depth, as Theorem 7 proves, and with the notion that complexity of parallel unification is measured by the depth of the term graphs [HJ87].

8 A new algorithm

We have looked at the existence of flat inputs. This is a class of inputs of bound depth, on which the best known algorithms do not compute in polylog PRAM Time even though such inputs are obtained, in *LogSpace*, from those on which algorithms compute in polylog. Unfortunately, our reductions are not invertible so we cannot go back. In this section we present Algorithm 3 which improves the performance of known unification algorithms and also computes in polylog PRAM Time on flat inputs. The novelty of the algorithm is in the way the propagation closure is computed. In Algorithm 2, propagation is always applied to the input dag. In this way, the propagation closure cannot take into account the equivalence that the partially computed unification closure P induces on the sons of vertices in P. In the proposed algorithm the

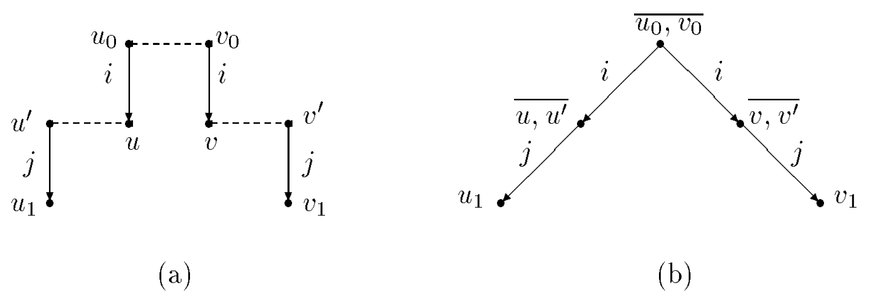


Figure 6: $\mathcal{D}$ and $\mathcal{D}/\mathcal{P}$

computation of the propagation closure is applied on the dag modulo P, i.e. the graph which is obtained contracting vertices which belong to the same class of the partition of P. This means that if vertex u, with outdegree 0 (possibly a variable with indegree $m > 1$), is in relation P with a node u', with outdegree $\neq 0$, then the propagation closure from vertex u reaches the sons of u', like its own sons. This can be seen in Figure 6 where eight vertices of a dag $\mathcal{D}$, are connected by four edges, labeled either i or j. In the figure, the equivalence relation P is graphically represented by dashed lines and contains the pairs $\{\{u_0, v_0\}, \{u, u'\}, \{v, v'\}\}$. Thus, we can see that the propagation closure on $\mathcal{D}$ computes $\{\{u_0, v_0\}, \{u, u'\}, \{v, v'\}, \{u, v\}\}$, while on $\mathcal{D}/\mathcal{R}$ computes $\{\{u_0, v_0\}, \{u, u'\}, \{v, v'\}, \{u, v\}, \{u_1, v_1\}\}$, Figure 6 b. Hence, in the first case, besides P, we only have $\{u, v\}$, while in the second case we have $\{u_1, v_1\}$ in addition.

Algorithm 3, Figure 7, is obtained replacing step 2.b, in Figure 2, with the following:

$$\mathit{set}\ S = \{\{u_s, v_s\} | <\{u, v\}, \{u_s, v_s\}> \in \biguplus(\bigtriangledown_{P, \mathcal{D}_{\mathcal{V},\mathcal{E}}/P})\}$$

Given a dag $\mathcal{D}_{\mathcal{V},\mathcal{E}}$ and an equivalence relation $\mathcal{R}$ on $\mathcal{V}$, $\mathcal{D}_{\mathcal{V},\mathcal{E}}/\mathcal{R}$ is the graph whose set of vertices, $\mathcal{V}'$, and set of edges, $\mathcal{E}'$, are such that:

$$\begin{array}{lll} u/\mathcal{R} \in \mathcal{V}' & \mathit{iff} & u \in \mathcal{V} \\ (\forall u \neq v \in \mathcal{V})\ u/\mathcal{R} \neq v/\mathcal{R} & \mathit{iff} & \{u, v\} \notin \mathcal{R} \\ (u/\mathcal{R}, v/\mathcal{R}, d) \in \mathcal{E}' & \mathit{iff} & (u, v, d) \in \mathcal{E} \end{array}$$

where $u/\mathcal{R}$ is a vertex of the class of $\mathcal{R}$ which u belongs to and denotes the representative of such a class. The computation of $\mathcal{D}_{\mathcal{V},\mathcal{E}}/\mathcal{R}$ is trivially in *LogSpace*. Furthermore, it results in a directed graph which may contain (1) cycles, (2) vertices with more than one outgoing edge labeled in the same way. Cycles do not affect the propagation closure, which can be computed in polylog PRAM Time using, for instance, algorithms for graph reachability [Bor77]. Regarding (2), Figure 6.b shows vertex $\overline{u_0, v_0}$ which has two edges with the same label. Acyclic graphs of this sort become an attractive representation when unification is expressed through sets of equations [MM82]. In this case, vertices, with more than one outgoing with the same label, represent (multi-)equations [JK91], i.e. terms that have to be matched against. Algorithm 3 allows us to deal with such graphs as input. The reason for this is that the relation triangle $\bigtriangledown$ is also defined for $< \{u, v\}, \{u_s, v_s\} >$ where $u = v$.

Theorem 9 *(1) Algorithm 3 is correct and complete*

Program Computation of $\mathcal{UC}(\mathcal{R})$ (Unification)

input dag: $\mathcal{D}_{\mathcal{V},\mathcal{E}}$; relation: $\mathcal{R} = \{\{u_0, v_0\}\}$
output relation $\mathcal{UC}(\mathcal{R})$: P

begin
1. set $S = \mathcal{R}$
2. **repeat**
   a. set $P = S$
   b. *propagation*: set $S = \{\{u_s, v_s\} | <\{u, v\}, \{u_s, v_s\}> \in \biguplus(\bigtriangledown_{P, \mathcal{D}_{\mathcal{V}',\mathcal{E}'}/P})\}$
   c. *closure*: set $S = \biguplus(S)$
   until $P = S$
3. test classes of $\mathcal{UC}(\mathcal{R})$ for homogeneity
4. test the graph $\mathcal{D}_{\mathcal{V},\mathcal{E}}/\mathcal{UC}(\mathcal{R})$ for acyclicity
5. construct the substitution

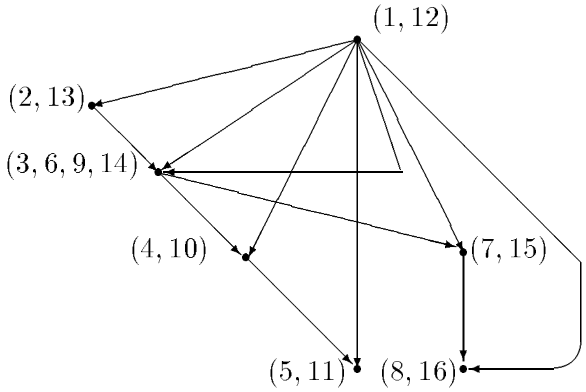
end.

Figure 7: Algorithm 3

(2) Let I be an input for $\mathcal{MGU}$, B_I be the iterations of steps 2.a-c, which Algorithm 2 needs to compute with I, N_I be the iterations of step 2.a-c that Algorithm 3 needs. Then, $N_I \leq B_I$.

(3) Let I_d be a flat input. Then, Algorithm 3 computes in polylog PRAM Time with the size of I_d.

Proof: (1) Correctness follows from Theorem 1. In fact, after step 2, P is the unification closure by construction. To prove completeness, we must only prove termination. At each iteration, step 2.b computes a new set S if and only if a new pair of corresponding sons can be reached. Since it requires the traversal of a new pair of edges and the set of edges of $\mathcal{D}_{\mathcal{V},\mathcal{E}}/P$ is bound by $|\mathcal{E}|$ then, the number of iterations of step 2.b is bound by $|\mathcal{E}|$ too. (2) The two algorithms differ in the computation of S, by the respective step 2.b . Moreover, if S is unchanged, i.e. $S = P$, then iterations stop. Let S_B and S_N be the sets computed by the respective steps 2.b of the algorithms. It is easy to prove that triangle $\bigtriangledown$ is such that: for each P, Q and graph $\mathcal{D}$, $\bigtriangledown_{P,\mathcal{D}} \subseteq \bigtriangledown_{Q,\mathcal{D}}$, if $P \subseteq Q$, and also, that $(\bigtriangledown_{P,\mathcal{D}})/P \subseteq \bigtriangledown_{P,\mathcal{D}/P}$. Then, at each iteration of step 2.b, we have: $S_B \subseteq S_N \subseteq \mathcal{UC}(\mathcal{R})$. (3) Since I_d is a flat input, by definition, input I exists such that: $I_d = \vec{\mu}_k(I)$, and Algorithm 2 computes with I in polylog. Let $\{u_0, v_0\}$ be the roots of I and, $\{u_0^d, v_0^d\}$ be the roots of I_d. Let ρ be the homomorphism from I_d to I, we introduced in the proof of Lemma 4, and u_0^ρ, v_0^ρ be the vertices of I_d such that $\rho(u_0^\rho) = u_0$ and $\rho(v_0^\rho) = v_0$. By definition of $\vec{\mu}$, after the first iteration of step 2.b with I_d and with $\mathcal{R}$=$\{u_0^d, v_0^d\}$, set $S = S_1 \cup S_2$, where S_1=$\{\{u_0^d, v_0^d\}, \{u_0^\rho, v_0^\rho\}\}$. While, set S_2 is $\{\{u, v\} | \mathcal{Q}(u) = f_{\mu_k}(p_{j_i}), \mathcal{Q}(v) = \mu_k(\vec{\mathcal{Q}}(I)|_{p_{j_i}})$ by construction of $\vec{\mu}_k(I)$ (see Definition 6). Let I' be I removing its roots, and I'_d be I_d removing vertices in the pairs of S_1. Then, I' and I'_d/S_2 are isomorphic. Since pairs in S_1 have incoming edges only from the



Initially: $P = \mathcal{R} = \{\{1, 12\}\}$

1^{st} iteration of step 2 $P^{(1)} = \mathcal{R} \cup \{\{2, 13\}, \{3, 9\}, \{4, 10\}, \{5, 11\}$
$\{6, 14\}, \{7, 15\}, \{8, 16\}, \{6, 9\}, \{9, 14\},$
$\{3, 6\}, \{3, 14\}\}$

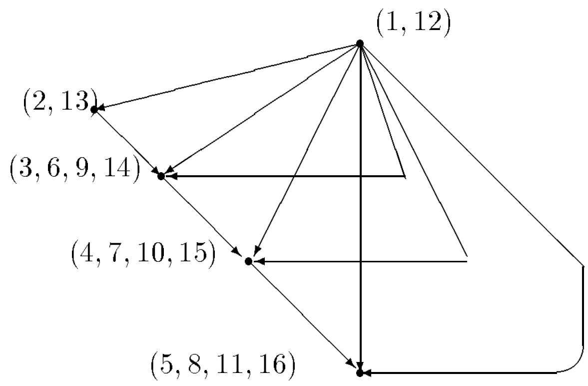
Figure 8: The graph $\mathcal{D}/P^{(1)}$ computed by Algorithm 3

roots of I_d (and the roots have been already traversed), after first traversal, Algorithm 3 computes with I' and with I'_d/S_2 in the same way. $\square$

We conclude the section showing, in Figure 8-9, the graphs obtained after, respectively, first and second iteration of step 2 of Algorithm 3 with input
$\vec{\mu}_k(s(s(s(a))), s(s(s(a))))$.

9 Conclusions

We have studied the effects of the main bounds, *depth* and *breadth*, on the inputs of $\mathcal{MGU}$. We have introduced two families of subproblems of $\mathcal{MGU}$ and proved that, contrary to what happens in other $\mathcal{P}$-Complete problems, such as $\mathcal{MCV}$, input depth does not influence the complexity of $\mathcal{MGU}$. As a matter of fact, $\mathcal{MGU}$*(k-depth)* has been proved in $\mathcal{P}$-Complete for any $k > 1$. This does not contradict the pathological nature of the *worst inputs* [Yas84, Rob92] and the experimental results collected in [HJ87] seem to confirm that the worst inputs are rarely met in real applications. But it definitely rejects “the assumption that the complexity of parallel unification is precisely measured by the depth of the term graphs” [HJ87]. Nevertheless, we remarked that most of the algorithms proposed for parallel unification do not include mechanisms to adequately deal with dag traversing, i.e. *downward propagation*. So that input depth may be a bound for the running time of such algorithms. This problem is overcome in the algorithms of [Yas84, DKM84, KR89] where *propagation closure*, possibly based on the computation of graph reachability, is introduced. We have also shown that $\mathcal{MGU}$*(k-breadth)* is in $\mathcal{P}$-Complete for any $k > 1$. But input breadth may actually be



2^{st} iteration of step 2 $\quad P^{(2)} = P^{(1)} \cup \{\{7,10\},\{4,7\},\{4,15\},\{10,15\}\}$
$\{\{8,11\},\{5,8\},\{5,16\},\{11,16\}\}$

Figure 9: The graph $\mathcal{D}/P^{(2)}$ computed by Algorithm 3

critical for the time taken to compute unification. Subsequently, we introduced the class of *flat inputs* which have bound depth. The computation of Algorithm 2 with flat inputs requires linear PRAM Time, but is obtained, in *LogSpace* , by the reductions of inputs, of unbound depth, which are all computable in polylog PRAM Time. The existence of this class does not necessarily mean that the classes $\{\mathcal{MGU}$*(k-breadth)*$\}$ contain fewer worst inputs than the class $\{\mathcal{MGU}$*(k-depth)*$\}$. However, we did not find any subclass similar to flat inputs in the class $\{\mathcal{MGU}$*(k-breadth)*$\}$. Finally, we have defined a variant of the best known algorithm which extends the classes of the inputs computable in polylog with the class of *flat inputs*.

References

[AM85] I.E. Auger and Krishnamoorthy M.S. A parallel algorithm for the monadic unification problem. *BIT*, 25:302–306, 1985.

[Bar90] J. Barklund. *Parallel unification.* PhD thesis, Upsala University, 1990.

[Bau93] A. Baudet. Combining unification algorithms. *Journal of Symbolic Computation*, 16(6):597–626, 1993.

[Bax76] S. Baxter. *The complexity of unification.* PhD thesis, University of Waterloo, 1976.

[BM99] R. Bejor and F. Mania. Solving combinatorial problems with regular local search algorithms. In *Proc. Of 6th LPAR'99*, volume 1705 of *LNCS*, pages 33–43. Springer Verlag, 1999.

[BO] M. Bellia and M.E. Occhiuto. Parallel unification: problems and perspectives. In preparation.

[BO95] M. Bellia and M.E. Occhiuto. Suprema of open and closed formulas and their application to resolution. *Information and Computation*, 117(1):136–150, 1995.

[BO99] M. Bellia and M.E. Occhiuto. Combinatorial lazy unification. *Journal of Symbolic Computation*, 27:185–206, 1999.

[Bod99] L. Bodea. Knowledge modeling and reusability in $e_x claim$. In *Proc. Of 60th IJCAI'99*, pages 606–611, 1999.

[Bor77] A. Borodin. On relating time and space to size and depth. *SIAM J. Computing*, 6(4):733–744, 1977.

[BS98] F. Baader and K.U. Schulz. Unification in the union of disjoint equational theories: Combining decision procedures. *Journal of Symbolic Computation*, 21(2):211–244, 1998.

[CB83] J. Corbin and M. Bidoit. A rehabilitation of Robinson's unification algorithm. In R.E.A.Mason, editor, *Information Processing 83*, pages 909–914. Elsevier Science Publisher B.V (North-Holland), 1983.

[CJC82] F. Chin, Lam J., and I. Chen. Efficient parallel algorithms for some graph problems. *CACM*, 25:659–665, 1982.

[CL74] C. Chang and R.C. Lee. *Symbolic Logic and Mechanical Theorem Proving.* Academic Press, N.Y., 1974.

[Cla91] K.L. Clark. Logic-programming schemes and their implementations. In J. L Lassez and G. Plotkin, editors, *Essays in Honor of Alan Robinson, Computational Logic*, pages 487–541. MIT Press, 1991.

[Col82] A. Colmerauer. Prolog and infinite trees. In K.L. Clark and SA. Tarnlund, editors, *Logic Programming*, pages 231–251. Academic Press, 1982.

[CT95] K.W. Chong and Lam T.W. Finding connected components in $O(\log n \log\log n)$ Time on the EREW PRAM. *J. of Algorithms*, 18:378–402, 1995.

[Dek81] A. Dekel. Parallel matrix and graph algorithms. *SIAM Computing Journal*, 10(4):657–675, 1981.

[DKM84] C. Dwork, P.C. Kanellakis, and J.C. Mitchell. On the sequential nature of unification. *J. Logic Programming*, 1:35–50, 1984.

[DKS86] C. Dwork, P.C. Kanellakis, and L. Stockemayer. Parallel algorithms for term matching. In *Proc. Of the 8^{th} Int. Conf. On Automated Deduction*, 1986.

[DSR80] P.J. Downey, R. Sethi, and Tarjan R.E. Variations on the common subexpression problem. *J. ACM*, 27(6):758–771, 1980.

[Ede85] E. Eder. Properties of substitutions and unification. *Journal of Symbolic Computation*, 1:31–46, 1985.

[Gol77] L.M. Goldschlager. The monotone and planar circuit value problems are logspace complete for p. *SIGACT News*, 9(2):25–29, 1977.

[HJ87] J. Harland and J. Jaffar. On parallel unification for prolog. *New Generation Computing*, 5:259–279, 1987.

[Hue76] G. Huet. *Resolution d'equations dans les language d'ordre 1,2,...,w.* PhD thesis, Univ. de Paris VII, 1976.

[IS99] K. Inoue and C. Sakama. Abducing priorities to derive intended conclusions. In *Proc. Of 60th IJCAI'99*, pages 44–49, 1999.

[J92́] J. Jájá. *An introduction to parallel algorithms.* Addison Wesley Publishing Company, 1992.

[Jaf84] J. Jaffar. Efficient unification over infinite terms. *New Generation Computing*, 2:207–219, 1984.

[JJ98] P. Jansson and J. Jeuring. Polytypic unification. *Functional Programming*, 8:527–536, 1998.

[JK91] J-P. Jouannaud and C. Kirchner. Solving equations in abstract algebras: A rule-based survey of unification. In J.L. Lassez and G. Plotkin, editors, *Computational Logic: Essay in Honor of Alan Robinson, The MIT Press*, pages 257–321. MIT Press, 1991.

[Joh90] D.S. Johnson. A catalog of complexity classes. In J.V. Leeuwen, editor, *Algorithms and Complexity*, HandBook of Theoretical Computer Science, pages 67–162. Elsevier Science Publisher, 1990.

[Kir90] C. Kirchner. *Unification.* Academic press, 1990.

[Kni89] K. Knight. Unification: A multidisciplinary survey. *ACM Computing Survey*, 21(1), 1989.

[Koz77] D.C. Kozen. Complexity of finitely presented algebras. In *Proc.* 9^{th} *ACM STOC*, pages 164–177, 1977.

[KP86] D. Kapur and Narendram P. Np-completeness of the set unification and matching problems. In *8th CADE*, volume 230 of *LNCS*, pages 489–495. Springer Verlag, 1986.

[KR89] P.C. Kanellakis and P.S. Reves. On relationship of congruence closure and unification. *J. of Symbolic Computation*, 7(4):427–44, 1989.

[MM82] A. Martelli and U. Montanari. An efficient unification algorithm. *ACM Trans. on Progr. Lang. and Sys*, 4(2):259–282, 1982.

[MMR86] A. Martelli, C. Moiso, and G.F. Rossi. An algorithm for unification in equational theories. In *Symposium on Logic Programming*, pages 259–282, 1986.

[MO84] T. (ed.) Moto-Oka. *Fifth generation computer systems.* North-Holland, 1984.

[Muk83] K. Mukai. A unification algorithm for infinite tree. In *Proc. IJCAI'83*, pages 547–549, 1983.

[Pal90] C. Palamidessi. Alegebraic properties of idempotent substitutions. In *Proc. Of 60th ICALP'90*, volume 443 of *LNCS*, pages 386–399. Springer Verlag, 1990.

[PW78] M.S. Paterson and M.N. Wegman. Linear unification. *J. Comput. Syst. Sci*, 16:158–167, 1978.

[QD84] M.J. Quinn and N. Deo. Parallel graph algorithms. *ACM Computing Surveys*, 16(3):319–348, 1984.

[Rob65] J.A. Robinson. A machine-oriented logic based on the resolution principle. *J.ACM*, 12:23–41, 1965.

[Rob76] J.A. Robinson. Fast unification. In *Theorem Proving Workshop*, 1976.

[Rob79] J.A. Robinson. *Logic: Form and Function. The mechanization of deductive reasoning.* Edinburgh Univ.Press, 1979.

[Rob85] P. Robinson. The SUM: an AI coprocessor. *Byte*, pages 169–180, June 1985.

[Rob92] J.A. Robinson. Logic and logic programming. *Comm.ACM*, 35(3):40–65, 1992.

[RP90] P. Ruzicka and I. Privara. An almost linear Robinson unification algorithm. *Acta Informatica*, 27(1):232–245, 1990.

[Sha89] E. Shapirp. The family of concurrent logic programming languages. *Computing Surveys*, 21(3):413–510, 1989.

[Sie90] J.H. Siekmann. Unification theory. In Kirchner C., editor, *Unification*, pages 1–44. Academic Press, NewYork, 1990.

[VS86] S. Vitter and R.A. Simon. New classes in parallel complexity: A study of unification and other complete problems. *IEEE Trans. Comput*, 35(5), 1986.

[Yas84] H. Yasuura. On parallel computational complexity of unification. In *Proc. Of the Intern. Conf. On the Fifth Generation Computer Systems (ICOT)*, 1984.