数理解析研究所講究録
625巻 1987年 147-156

On Parallel Computation Time of Unification for Restricted Terms

Masaaki OHKUBO, Hiroto YASUURA and Shuzo YAJIMA
Faculty of Engineering, KYOTO University

1. Introduction

Unification[1] in first-order logic is one of the elementary operations in logic programming languages such as Prolog, mechanical theorem provers based on resolution, type inference systems, term rewriting systems and so on[2]. Informally, unification is defined as follows: Given two terms constructed of variables and function symbols, find, if it exists, the simplest assignment of an appropriate term to every variable which makes the two terms equal. In [3] and [4], Yasuura and Dwork's group showed that unification is log-DEPTH (log-SPACE) complete for PSIZE (PTIME). This fact suggests that parallel unification algorithm may not perform significantly faster than the best sequential algorithm[2][5]. In practical experiments, however, it was obtained that the number of variables or arity of each function symbol is not so large[6]. There remains a possibility that unifications appearing in actual execution of Prolog programs is quite easier than general cases discussed in [3] and [4].

In this paper, we discuss parallel computation time of unification for restricted terms. We place restrictions on appearance of variables, arity and nesting on function symbols of terms to be unified. We show that unification for quite strongly restricted terms has the same complexity of one for terms without any restrictions.

2. Preliminaries

2.1 Unifiability Decision Problem[1][3]

In this paper, we discuss unification in first-order logic. Let F be a set of function symbols and V be a set of variables. We assume that $F \cap V = \phi$. Each function symbol has a fixed arity, a nonnegative integer, and zero-arity function symbols are called constants. We use lower case letters $a, b, f, g, \ldots$ as function symbols, and upper letters $X, Y, X_1, Y_1, \ldots$ as variables.

Terms on $F \cup V$ are defined recursively as follows:

(1) a variable $X \in V$ or a constant $a \in F$ is a term.

(2) if $t_1, t_2, \ldots, t_k$ are terms and $f \in F$ is a k-arity function symbol ($k > 0$), then $f(t_1, t_2, \ldots, t_k)$ is a term.

Let T be the set of terms on $F \cup V$. A ***substitution*** $\sigma: X \to T$ is represented by a finite set of ordered pairs of terms and variables $\{(t_i, X_i) \mid t_i$ is a term, X_i is a variable and no two pairs have the same variables as the second element$\}$. Applying a substitution σ to a term t, we represent the resulting term by $\sigma(t)$. A substitution σ is called ***unifier*** for t_1 and t_2, if and only if $\sigma(t_1) = \sigma(t_2)$. We also say that t_1 and t_2 are ***unifiable*** when there is a unifier for them. A unifier σ is said to be the ***most general unifier*** (*MGU*) for t_1 and t_2, if and only if σ is a unifier for t_1 and t_2, and for every unifier θ of them there is a substitution λ such that $\theta = \sigma * \lambda$, where $*$ means the composition of substitutions. If two terms are unifiable, there is an *MGU* and it is unique up to variable renaming.

A term can be represented by a labeled directed acyclic graph $G=(N,E)$, called a ***term graph***, as the following manner:

(1) Every node $v \in N$ has a unique label in $F \cup V$. Every node labeled with a variable $X \in V$, called a ***variable node***, has outdegree 0, and no two nodes have the same label. Every node labeled with a k-arity function symbol $f \in F$, called a ***function node***, has k ($k \geqq 0$) outgoing edges each of which is labeled with $1,2,\ldots,k$, respectively.

(2) A node with a label t, a constant or a variable, represents a term t. A node v with label f, a k-arity function symbol ($k>0$), represents a term $f(t_1,t_2,\ldots,t_k)$ where t_i is a term represented by the node pointed by the i-th outgoing edge of v.

A term graph $G=(N,E)$ is encoded in $O(|E|\log|N|+|N|\log|N|)$ bits, where $|N|$ is the number of nodes and $|E|$ is the number of edges in G.

The ***unifiability decision problem*** (*UDP*) is defined as follows: For a given term graph G and two nodes v_1 and v_2 in G, decide whether or not t_1 and t_2 are unifiable, where t_1 and t_2 are terms represented by v_1 and v_2, respectively.

2.2 Parallel Computation Model and Complexity Classes

In this paper, for simplicity, we are concerned only with unifiability for given two terms, and discuss its time complexity in parallel computation. Clearly, UDP is not harder than the unification problem finding MGU.

We adopt combinational circuits as a model of parallel computation. We will be concerned with an implementation of a UDP computation circuit as a combinational circuit, and estimate parallel computation time for UDP by the depth of the circuit.

We represent the complexity classes of problems, which are computable by combinational circuits with polynomial size and with log depth, by *PSIZE* and *LOGDEPTH*, respectively[7][8]. A problem P is ***log-DEPTH reducible to*** a problem Q, if and only if there is an $O(\log n)$ depth combinational circuit C for computing P, and C is allowed to have oracle nodes for Q. An ***oracle node*** for Q computes Q, and the depth of it is defined as $\lceil \log r \rceil$, where r is the length of input for Q. A problem P is called ***log-DEPTH complete for*** a class A of problems, if and only if P is in A and every problem in A is *log-DEPTH* reducible to P. In other words, P is the most difficult problem in A with respect of parallel computation time. Formal definitions of these concepts are described in [7].

DLOGSPACE (*NLOGSPACE*) is the class of problems computable by an $O(\log n)$ tape bounded deterministic (nondeterministic) Turing machine.

3. Parallel Computation Time of Unification for General Terms[3]

In this section, we introduce parallel complexity of unification without restrictions. It has been derived in [3] by showing the relations between UDP and the directed hypergraph accessibility problem.

A ***directed hypergraph*** $H=(N_h, E_h)$, where N_h is a set of nodes and E_h is a set of directed hyperedges, is a generalization of a directed graph. A ***hyperedge*** e is an ordered pair of a pair of nodes $\{w_i, w_j\}$ in N_h and a node w_k in $N_h-\{w_i, w_j\}$, denoted $(\{w_i, w_j\}, w_e)$. We sometimes call the edge e a ***normal edge*** when $w_i=w_j$. In a directed hypergraph $H=(N_h, E_h)$, w in N_h is said to be ***accessible*** from a subset S of N_h, if and only if w is a member of S or there exists a hyperedge $(\{w_i, w_j\}, w)$ such that both w_i and w_j are accessible from S.

An ***incidence matrix*** of a directed hypergraph $H=(N_h, E_h)$ is an $|N_h|\times|E_h|$ matrix(h_{ij}). Each entry h_{ij} represents a relation between the node w_i and the hyperedge $e_j=(\{w_{j_1}, w_{j_2}\}, w_k)$. If $w_i=w_k$ then $h_{ij}=2$, if w_i is w_{j_1} or w_{j_2} then $h_{ij}=1$, and otherwise $h_{ij}=0$. We can encode (h_{ij}) into binary with $O(|N_h||E_h|)$ bits.

The ***directed hypergraph accessibility problem*** (*DHGAP*) is defined as follows: For a given incidence matrix of a hypergraph $H=(N_h, E_h)$, a subset of nodes S and a node w in H, determine whether w is accessible from S.

UDP can be reduced to DHGAP by the following algorithm.

Algorithm UNIFY[3]

input: A binary coding of a term graph $G=(V,E)$, nodes v_1 and v_2 in V which represent terms t_1 and t_2 respectively.

output: If t_1 and t_2 are unifiable, the output is 'YES', otherwise 'NO'.

step 1 Construct a hypergraph $H=(N_h, E_h)$ as follows: For every pair of nodes v_i and v_j in N, generate a node w_{ij} in N_h where $w_{ij}=w_{ji}$. If v_i and v_j have the same label in F and the h-th outgoing edges of them point to v_k and v_l respectively, generate a normal edge (w_{ij}, w_{kl}) in E_h. If the label of v_k is a variable in V, generate a hyperedge $(\{w_{ik}, w_{jk}\}, w_{ij})$ in E_h for every v_i and v_j.

step 2 Compute the accessibility problem of H from w_{12} to every node in E_h in parallel.

step 3 If there exists a node w_{ij} in N_h such that it is accessible from w_{12}, and v_i and v_j have different labels in F, then output 'NO', otherwise output 'YES'.□

UDP is log-DEPTH reducible to DHGAP by the algorithm *UNIFY*. Conversely, DHGAP for any acyclic directed hypergraph is log-DEPTH reducible to UDP. UDP is log-DEPTH complete for PSIZE since DHGAP is log-DEPTH complete for PSIZE[3]. It is also shown that UDP can be computed by a combinational circuit with depth $O(\log^2 n + m\log m)$, where n is the number of nodes in a given term graph G and m the number of variable nodes in G.

4. Parallel Computation Time of Unification for Restricted Terms

4.1 Restrictions on Variables

From the discussion of Section 3, it is clear that if m is small enough, more precisely $m \leq \log^2 n/\log\log n$, UDP is computable by an $O(\log^2 n)$ depth combinational circuit. In this subsection, we show the parallel complexity of UDP under some restrictions on variables.

The ***directed acyclic graph accessibility problem*** (*AGAP*) is defined as follows: For a given adjacency matrix of a directed acyclic graph $G_A = (N_A, E_A)$, and two nodes u_1 and u_2 in G_A, determine whether u_2 is accessible from u_1. We represent an AGAP by

(G_A, u_1, u_2). From the discussion of [9], we can easily show the following lemma.

Lemma 4.1.1 AGAP is log-DEPTH complete for NLOGSPACE.□

At first, we show the parallel complexity of UDP with no variables. We call the problem ***labeled directed acyclic graph matching*** (***DAG matching***). If two terms are given in the form of a string of symbols, it is a trivial operation on strings, but it is not quite so trivial an operation when terms are represented by term graphs. Since only normal edges appear in Algorithm *UNIFY* for *DAG* matching, *DAG* matching and *AGAP* are log-DEPTH reducible each other.

Theorem 4.1.1 DAG matching is log-DEPTH complete for Co-NLOGSPACE.□

We will show the complexity is equal to one of DAG matching even if variables appear only in one term. We call the problem ***pattern matching***. For pattern matching, any path from w_{12} to any node in *UNIFY* includes only one hyperedge. Thus we only need to solve *AGAP* twice.

Theorem 4.1.2 Pattern matching is log-DEPTH complete for Co-NLOGSPACE.□

4.2 Restriction on Function Symbols

In this subsection, we show the parallel complexity of UDP under restriction on outdegree of function nodes, *i.e.*, restriction on arity of function symbols. In the following discussion, the maximum outdegree of function nodes in G is denoted by q.

It is well known that *AGAP* for a graph in which outdegree of each node is not greater than 1 is log-DEPTH complete for DLOGSPACE[7]. When q=1, no node accessible from w_{12} in the hypergraph of *UNIFY* has two or more outgoing edges. Then we have the following Lemma.

Lemma 4.2.1 If q=1, UDP is log-DEPTH complete for DLOGSPACE.□

For any acyclic directed graph H, in which outdegree of each node is at most 2, *DHGAP* for H is log-DEPTH complete for *PSIZE* by the similar discussion in [3]. We can easily show that *DHGAP* for H is log-DEPTH reducible to *UDP* with q=2 by the same way in [3]. Therefore, *UDP* with q=2 is log-DEPTH complete for *PSIZE*.

Theorem 4.2.1 For any $q \geq 2$, UDP for a term graph with the maximum outdegree q of function nodes is log-DEPTH complete for PSIZE. If q=1, UDP is log-DEPTH complete for DLOGSPACE.□

4.3 Restriction on Depth of Term Graphs

In this subsection, we show the parallel complexity of *UDP* under a restriction on the depth of term graphs. The depth of each node v_i and a term graph G are defined as follows: For a given term graph G and two nodes v_1 and v_2 in *UDP*, ***depth of*** v_i is the length, of the longest path from v_1 or v_2 to v_i. If there is no path from v_1 (v_2) to v_i, the length of the path is defined as 0. The ***depth of*** G, denoted d, is defined as the maximum depth of nodes.

Assume that $d=1$. ***For a given term graph*** G, we can construct an undirected graph G' such that nodes are constant or variable nodes in G and there is an edge (v_i, v_j) if v_i and v_j are h-th sons of each root node in *G*. It is easy to show that negation of *UDP* is equal to accessibility problem on *G'* from a constant node to different constant nodes. Accessibility problem for undirected graphs is *NLOGSPACE* but has not shown to be log-DEPTH complete.

Lemma 4.3.1 If d=1, UDP is in Co-NLOGSPACE.□

Introducing new variables, we can easily transform a term graph with large depth into a term graph with depth 2. Thus we have the following theorem.

Theorem 4.3.1 For any $d \geqq 2$, UDP for a term graph with the maximum depth d is log-DEPTH complete for PSIZE. If d=1, UDP is in Co-NLOGSPACE and there is $O(\log^2 n)$ depth circuit for it..□

5. Conclusion

We have shown the parallel computational complexity of unification under several restrictions on terms. The results obtained in this paper suggest that it is difficult to design a parallel unification algorithm in time $O(\log^k n)$ except for DAG matching and pattern matching, even if outdegree of function nodes and the depth of a term graph are restricted to at most 2.

By the similar discussion in section 4, we can also show that we can change "log-DEPTH complete for" with "NC^1 complete for[8]" in the theorems proved in this paper, if we only replace "for PSIZE" with "for PTIME" in them.

References

[1] Robinson,J.A. : A Machine-Oriented Logic Based on the Resolu-

tion Principle, *J. ACM* 12, 1 (1965), 23-41.

[2] Martelli,A. and Montanari,U. : An Efficient Unification Algorithm, ACM TOPLAS 4, 2 (Apr. 1982) 258-282.

[3] Yasuura,H. : On Parallel Computational Complexity of Unification, *FGCS* (1984), 235-243.

[4] Dwork,C., Kanellakis,P.C. and Mitchell,J.C. : On the Sequential Nature of Unification, *J. Logic Programming* (1984) 35-50.

[5] Paterson,M.S. and Wegman,M.N. : Linear Unification, *JCSS* 16 (1978), 158-167.

[6] Onai,R., Shimizu,H., Masuda,K. and Aso,M. : Analysis of Sequential Prolog Programs, *J. Logic Programming* (1986), 119-141.

[7] Yasuura,H. : Complexity Theory of Logic Circuits, *J. IPSJ* 26, 6 (June, 1986), 575-582 (in Japanese).

[8] Cook,S.A. : The Classification of Problems Which Have Fast Parallel Algorithms, *LNCS* 158 (*Proc. of Int. FCT Conf.*) (1983), 78-93.

[9] Borodin,A. : On Relating Time and Space to Size and Depth, *SIAM J. Comput.* 6, 4 (1977) 733-744.