ICOT Technical Report: TR-027

TR-027

On the Parallel Computational Complexity of Unification

by
Hiroto Yasuura
(Kyoto University, Japan)

October, 1983

ICOT
Mita Kokusai Bldg. 21F
4-28 Mita 1-Chome
Minato-ku Tokyo 108 Japan

(03) 456-3191～5
Telex ICOT J32964

Institute for New Generation Computer Technology

On the Parallel Computational Complexity of Unification

Hiroto YASUURA

(also availabl)

Yajima Lab. Research Report ER 83-01
October 12, 1983

Department of Information Science
Faculty of Engineering
Kyoto University
Kyoto 606,JAPAN

ABSTRACT

The computational complexity of unification in first-order logic under parallel computation scheme is discussed. The unification problem is described as a problem on directed acyclic graphs and related to the reachability problem on directed hypergraphs which is newly introduced here. We show that unification is the one of the hardest problems in the class of problems resolved by polynomial size circuits. Namely, unification is depth complete for PSIZE.

Key Words

Unification, Directed Hypergraph, Computational Complexity, Parallel Computation, Reachability Problem, log-depth Completeness, log-space Completeness, Combinational Logic Circuit, Logic Programming Languages

1. INTRODUCTION

The problem of unification in first-order logic is one of elementary operations in the area of theorem proving and logic programming languages such as PROLOG. The unification problem is defined as follows: Given two terms consisting of function symbols and variables, find, if it exists, a simple substitution which makes two terms equal.

Unification was first introduced by Robinson[1] as the basic operation of resolution and implemented in several resolution systems[2][3] and logic programming language PROLOG[4]. Since unification can be considered as a generalized operation of pattern matching, it was also applied to various areas where symbolic expressions are dealt with, such as in interpreters for equation languages, in knowledge base systems, in type checkers for programming languages with a complex type structure, and in the computation for term rewriting systems[5].

Two linear unification algorithms were proposed, in sequential computation, independently by Paterson-Wegman[6] and Martelli-Montanari[5]. For more efficient implementation of unification, one may try to design a parallel unification algorithm. However, no such efficient algorithm has been known and implemented.

In this paper, we are concerned with the complexity of unification in parallel computation. As the result, we show that it seems very hard to design a much more efficient unification algorithm using parallel computation than sequential one. In other words, unification contains essentially sequential

computation which might not accelerate by parallel computation scheme in the worst case.

For the model of parallel computation, we use combinational logic circuits constructed of fan-in restricted logic elements. This model seems to be one of the most stable models of parallel computation from computational complexity view point. Moreover, combinational logic circuits are the most natural model of implementation of parallel algorithms by hardware. The computation time is measured by the depth of circuits.

In the next section, we give some basic definitions and notations. We adopt a graph representation of terms for unification[6]. In section 3, we introduce a combinational logic circuits as a model of computation[7].

In order to discuss the relation between the complexity of logic circuits and unification, we introduce a new problem, called the reachability problem on directed hypergraphs. A directed hypergraph is a generalization of a directed graph and the reachability problem on the directed hypergraph is defined as an extension of one on the directed graph. We show a parallel algorithm for unification in which the unification problem is reduced to the reachability problem of a directed hypergraph. Definitions of directed hypergraphs and the reachability problem are also given in section 3. The parallel unification algorithm is presented in section 4, and its computation time is analyzed. In the section 5, we discuss about the lower bound of the parallel computation time of unification and show the unification problem is log-depth complete for a class of problems that can be solved by polynomial size circuits[7][8]. Namely, if we can

design circuits with depth $O(\log^k n)$ for unification, all problems in this class can be computed by circuits with depth $O(\log^k n)$ for any positive integer k. Therefore, we can say that unification is the hardest problem in this class which includes all problems having polynomial algorithms in sequential computation. It is also shown that unification is log-space complete for PTIME, by the similar discussion on the above parallel computation.

2.UNIFICATION

Let A_i (i=1,2,...) be a set of i-adic function symbols and A_0 be a set of constant symbols. $\bigcup_{i=0,1,..} A_i$ is denoted by A. Let X be a set of variables. We assume that $A \cap X = \emptyset$.

Terms on A and X are defined recursively as follows:

(1) Any a in A_0 and any x in X are terms.

(2) If t_1, t_2, ..., t_i are terms and f is a member of A_i, then $f(t_1,t_2,...,t_i)$ is also a term.

(3) All terms are generated by applying the above rules (1) and (2).

Let T be the set of terms on A and X. A substitution s:X->T is represented by a finite set of ordered pairs of terms and variables

{(t_i, x_i)| Every t_i is a term, every x_i is a variable and no two pairs have the same variable as the second element.}.

This representation means that $s(x_i)=t_i$, if (t_i, x_i) is in this set, and, otherwise, s(x)=x. Applying a substitution s to a term t, we simultaneously substitute all occurrences in t of every variable x, which is in a pair (t_i, x_i) of s, with the corresponding term t_i. We represent the resulting term by t_s. t_s is called an instance of t.

Let $s_1=\{(t_1,x_1), (t_2,x_2), ..., (t_n,x_n)\}$ and $s_2=\{(t_1,y_1),$

(t_2,y_2), ..., $(t_m,y_m)\}$ be two substitutions. The composition substitution of s_1 and s_2, denoted by s_1*s_2, is defined as follows:

(1) If (t,x) is included in s_1, then (t_{s_2},x) is in s_1*s_2.

(2) If (t,x) is included in s_2 and (t',x) is not included in s_1, then (t,x) is also in s_1*s_2.

[Example 1] For a term $t=f(x_1,g(x_2))$, instances of t under substitutions $s_1=\{(g(x_2),x_1)\ (x_1,x_2)\}$ and $s_2=\{(h(a),x_1)\}$ are

$t_{s_1}=f(g(x_2),g(x_1))$

and

$t_{s_2}=f(h(a),g(x_2))$,

respectively. The composition substitution of them $s_1*s_2=\{(g(x_2),x_1),\ (h(a),x_2)\}$. Thus

$t_{s_1*s_2}=f(g(x_2),g(h(a)))$.

A substitution s is called a unifier for t_1 and t_2, if and only if $t_{1s}=t_{2s}$. We also say that t_1 and t_2 are unifiable when there is a unifier for them. More general, we can define a unifier for a set of terms $\{t_1, t_2, ..., t_n\}$ as a substitution s which makes $t_{1s}=t_{2s}=...=t_{ns}$.

A unifiers for a set $\{t_1, t_2, ..., t_n\}$ is said to be a most general unifier (MGU), if and only if for every unifier s' of the set there is a substitution s" such that s'=s*s".

[Example 2] For terms

$t_1=f(x_1, g(x_2), h(a))$

and

$t_2=f(g(x_3), g(h(x_3)), x_4)$,

a substitution $s=\{(g(x_3),x_1),\ (h(x_3),x_2),\ (h(a),x_4)\}$ is the most general unifier for t_1 and t_2.

A term t can be represented by an acyclic directed graph G=(V,E), called a term graph, as the following manner:

(1) Each vertex v in V has a label p in $A \cup X$. No two vertices have a same label in X, and the outdegree of them are 0. A vertex having a label in A_i ($i \geq 0$) has i outgoing edges each of which is labeled by a distinct positive integer j in {1, 2, ..., i}.

(2) A vertex with label p in $A_0 \cup X$ represents term p (p is a constant or a variable).

(3) A vertex v with label f in A_i ($i \geq 1$) represents term $f(t_1, t_2, \ldots, t_i)$ where t_j is a term represented by the vertex pointed by the j-th outgoing edge of v.

We sometimes call vertices with labels in X variable vertices, and ones with labels in A function vertices.

A term graph G=(V,E) is encoded in the following set of tuples.

$\{(v,p,v_1,v_2,\ldots,v_i) \mid$ v is a vertex in V, p is the label of v, and (v,v_j) $(1 \leq j \leq i)$ in E has the label j$\}$

We consider a binary coding of the tuple sequence \$G. Let n be the number of vertices in V and m be the number of edges in E. Since each vertex v and its label p can be coded by $\lceil \log_2 n \rceil$ bits, respectively, the length of the binary coding is $O(m \log n + n \log n)$ bits.

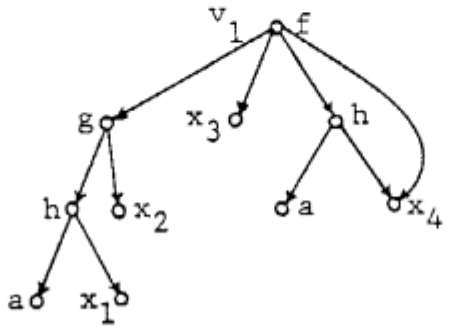


Fig.1 A Term Graph

[Example 3] In Fig.1, an example of term graphs is illustrated. The term represented by the root vertex v_1 is
$f(g(h(a,x_1),x_2), x_3, h(a,x_4), x_4)$.

Now we will define the unification problem formally.

[Definition 1] The unification problem (UP) is defined as follows: For a given coding $G of a term graph G and two vertices v_1 and v_2 in G, find, if it exists, a most general unifier s for terms t_1 and t_2 which are represented by v_1 and v_2 respectively.

[Definition 2] The unifiability decision problem (UDP) is defined as follows: For a given coding $G of a term graph G and two vertices v_1 and v_2 in G, decide whether or not t_1 and t_2 are unifiable.

3. COMPUTATION MODEL AND HYPERGRAPHS

3.1 Parallel Computation Model

In this paper, we adopt combinational logic circuits as a model of parallel computation. Combinational logic circuits are the most fundamental components of digital systems and many researches have been carried out on the complexity theory of logic functions realized by combinational circuits[7]. There is two measures to evaluate the complexity of circuits, size and depth, which correspond to the number of resources and the computation time spent in the computation respectively. In the following section we will mainly consider the depth of a circuit that computes UDP and UP.

A basis is a finite set of logic functions. A combinational logic circuit C over a basis B is a labeled acyclic directed graph. Vertices with indegree 0 are input vertices each of which is labeled with an element in $\{x_1, x_2, \ldots, x_n, 0, 1\}$. Output vertices each of which is labeled with an output variable have indegree 1. Other vertices are computation ones each of which is labeled with a logic function in B. Indegree of a vertex labeled with an i-variable function f is just i.

Reversing the direction of all edges of a combinational circuit C over a basis B, we can obtain a term graph G_C on $\{x_1, x_2, \ldots, x_n, 0, 1\} \cup B \cup \{0\}$, where 0 means an identical function assigned to each output vertex. The output function of each input or computation vertex in C is represented by the term corresponding the vertex in G_C. The output function of an output vertex v is represented by $\hat{0}(t)$ which is the same function

with t, where t is the label of vertex v' such that the edge (v',v) is in C (i.e., v is adjacent to v').

For a function $f:\{0,1\}^n \to \{0,1\}^m$, we say a combinational logic circuit C computes f, if there exists a labeling of input vertices and a set of output vertices $V=\{v_1, v_2, \ldots, v_m\}$ in C such that f equals to the set of logic functions represented by terms in V.

Levels of computation vertices in a circuit C are defined in the following way: the level of each input vertex is 0; the level of a computation vertex v is one greater than the maximum of the levels of the vertices to which v is adjacent.

The size of a combinational circuit C, denoted size(C), is the number of computation vertices in C. The depth of C, denoted depth(C), is the maximum level of the computation vertices in C. When we assume that the delay of computation only depends on the delay in each computation vertex (gate) and they have same value, depth(C) is linearly proportional to the delay of computation on C.

The combinational (or circuit) complexity of a function f relative to a basis B, denoted $C_B(f)$, is the smallest size of a circuit over B that computes f. The delay complexity of f relative to B, denoted $D_B(f)$, is the smallest depth of a circuit over B that computes f.

3.2 Directed Hypergraphs and Reachability Problem

In the algorithm for UP and UDP in the next section, we will transform UP and UDP to the reachability problem on directed

hypergraphs which is introduced in this paper. Here we define the directed hypergraphs and its reachability problem.

[Definition 3] A directed hypergraph H is denoted (V,E), where V is a set of vertices and E is a set of directed hyperedges. A hyperedge e is an ordered pair of a set of vertices V_e in V and a vertex v_e V-V_e, denoted (V_e,v_e). The number of elements in V_e is called the rank of hyperedge e. The rank of the directed hypergraph H is defined by the maximum rank of all hyperedges in E.

A directed hypergraph with rank 1 is just a directed graph. In this paper, we sometimes distinguish hyperedges with rank 1, called simply edges, from hyperedges with rank more than one.

An incidence matrix of a directed hypergraph H=(V,E) is an n*m matrix (a_{ij}), where n=|V| and m=|E|. Each entry a_{ij} represents a relation between the vertex v_i and the hyperedge e_j=(V_{e_j},v_{e_j}). If v_i=v_{e_j} then a_{ij}=2, if v_i is included in V_{e_j} then a_{ij}=1, and otherwise a_{ij}=0. We can encode (a_{ij}) into binary with O(mn) bits.

Now, we will define the reachability problem on directed hypergraphs. The reachability problem is a generalization of the reachability problem on directed graphs which has been examined in detail in the theory of computational complexity[7][8].

[Definition 4] In a directed hypergraph H=(V,E), v in V is said to be reachable from a subset S of V if and only if v is the

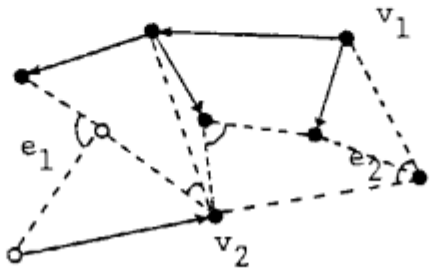


Fig.2 A Directed Hypergraph

member of S or there exists a hyperedge (V_e, v) such that all vertices in V_e is reachable from S.

[Definition 5] The reachability problem of directed hypergraphs (DHGAP) is defined as follows: For a given incidence matrix $H of a directed hypergraph H and a subset of vertices S, obtain all vertices in H which are reachable from S.

[Example 4] Fig.2 shows a hypergraph H. Edges are illustrated by arrows and hyperedges with rank more than one are drawn by dotted lines. The rank of hyperedges e_1 and e_2 is 2 and 3, respectively. The rank of the hypergraph H is 3. In H, all vertices reachable from $\{v_1, v_2\}$ is shown as black nodes.

4. A PARALLEL UNIFICATION ALGORITHM

4.1 A Unification Algorithm

First we show an algorithm for UDP.

[Algorithm 1] UNIFY

Input A binary coding \$G of a term graph $G=(V,E)$ on $X \cup A$, and vertices 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 '1'. Otherwise it is '0'.

Method

Step 1. Generate a directed hypergraph $H=(V',E')$ from G as follows:

(1) For every pair of vertices v_i and v_j in V, there is a vertex v_{ij} in V' where $v_{ij}=v_{ji}$.

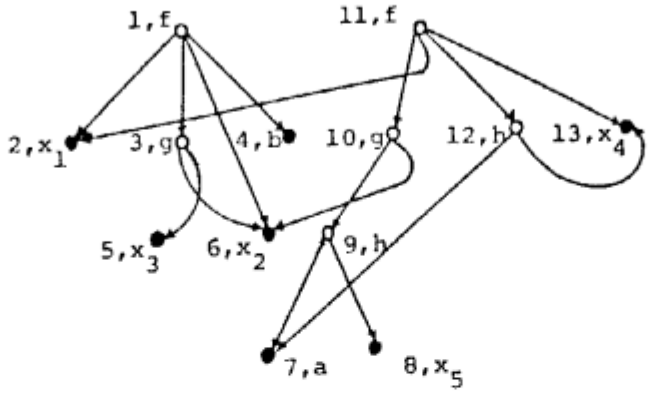
(2) If v_i and v_j have the same label in A and the h-th outgoing edges of them points to v_q and v_r respectively, an edge (v_{ij}, v_{qr}) is in E'.

(3) If the label of v_q is a variable in X, a hyperedge $(\{v_{iq},v_{jq}\}, v_{ij})$ is in E'.

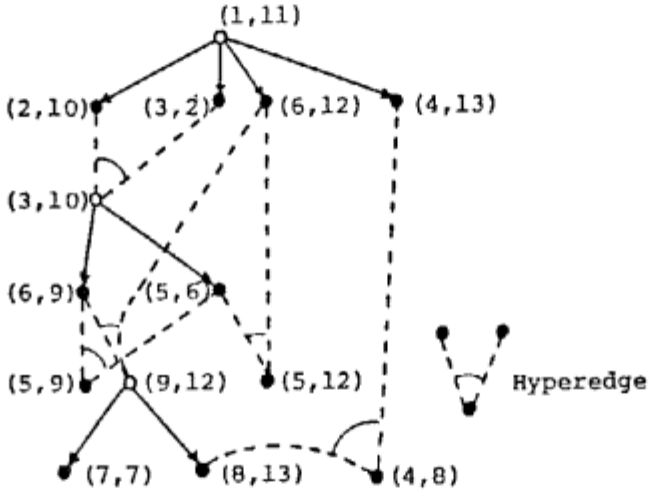
Step 2. Compute the reachability problem of H from $\{v_{12}\}$ in parallel.

Step 3. If there exists a vertex v_{ij} in V' such that it is reachable from v_{12} and v_1 and v_2 have different labels in A, output '0' and stop.

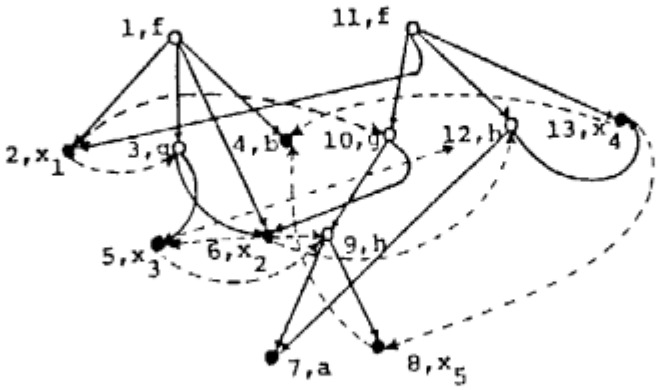
Step 4. For all v_{ij}'s which are reachable from v_{12} and v_i or v_j



(a) A Term Graph G



(b) Reachable Part of Hypergraph H



(c) The Resulting Graph G'

Fig.3 Execution of UNIFY

has a label in X, add edges into G by the following way. We call resulting graph G'.

(1) If the label of v_i (v_j) is in X and the label of v_j (v_i) is in A, then add edge (v_i,v_j) ((v_j,v_i)) into G.

(2) If both v_i and v_j have distinct labels in X, add edge (v_i,v_j) into G, where v_i is assumed to be assigned the smaller integer than v_j in the coding.

Step 5. Examine whether the graph G' is acyclic in parallel. If G' is acyclic output '1', otherwise '0'.

[Example 5] Fig.3 shows an execution of UNIFY for term

$f(x_1,g(x_2,x_3),x_2,b)$

and

$f(g(h(a,x_5),x_2),x_1,h(a,x_4),x_4)$.

(a) is a term graph representing these two terms. In (b), a part of the hypergraph H reachable from vertex (1,11) is illustrated. The hyperedge ({(2,10),(3,2)}, (3,10)) means that in the term graph G vertex 3 and 10 should also be unifiable because vertex 2 (representing variable x_1) must be unifiable both 3 and 10. In H, each black node represents vertex where at least one element of the label corresponds to a variable vertex in G. (c) is the final graph G'. Dotted lines express edges added in Step 4.

[Theorem 1] Algorithm UNIFY computes UDP correctly.

(Proof) For a given G and (v_1,v_2), t_1 and t_2 are unifiable if and only if vertices in G are partitioned by an equivalence relation satisfying the following conditions [6]:

(1) if two function vertices are equivalent then their

corresponding sons are equivalent in pairs;

(2) each equivalence class does not contain the vertices with distinct symbols in A;

(3) a reduced graph by the equivalent relation is acyclic.

We define an equivalence relation from the hypergraph H in Step 1 and Step 2. Namely, if v_{ij} in H is reachable from v_{12}, v_i is equivalent to v_j in G. Algorithm UNIFY stops at Step 3, if and only if the relation does not satisfy the condition (1) and (2). The condition (3) is checked by the acyclic test in Step 5.

Q.E.D.

It is easy to obtain a most general unifier from graph G' which is constructed in algorithm UNIFY. Thus we have an algorithm for UP by trivial modification of UNIFY.

[Algorithm 2] UNIFY-2

Input A binary coding \$G of a term graph G=(V,E) on $X^U A$, and vertices v_1 and v_2 in V which represent terms t_1 and t_2, respectively.

Output A most general unifier for t_1 and t_2. The mgu is represented by a term graph and a set of pairs $\{(v_i, x_i)\}$ where v_i is a vertex in the graph and x_i is a variable in X.

Method

Step 1. By algorithm UNIFY, generate a directed graph G' in Step 4 of UNIFY.

Step 2. For all pairs (v_i, v_j)'s in G', if there is a path from v_i to v_j in which all vertices except v_i and v_j have labels in X, connect v_i with v_j directly.

Step 3. For every vertex v with a variable label x, if there is an outgoing edge from v, make a pair (v', x) as follows:

(1) If all outgoing edges from v are pointing only variable vertices, make pairs (v', x)'s for every v' pointed by one of these edges.

(2) If there are function vertices pointed by outgoing edges, select a function vertex v' in them and generate a pair (v', x).

Step 4. Delete all edges from variable vertices.

Step 5. Delete all vertices that are not reachable from vertices in the pairs obtained in Step 3.

[Example 6] From the graph G' in Fig.3 (c), we obtain the most general unifier $\{(g(h(a,b),h(a,b)),x_1), (h(a,b),x_2), (h(a,b),x_3), (b,x_4), (b,x_5)\}$.

4.2 Depth Complexity of Algorithm UNIFY

In this subsection, we analyze the depth of a combinational logic circuit which computes UDP according to algorithm UNIFY.

Let n be the number of vertices in a given term graph G, n' be the number of vertices with labels of variables, and m be the number of edges in G.

[Lemma 1] The reachability problem of a hypergraph H in Step 2 can be computed by a combinational circuit with depth $O(\log^2 n + n' \log n')$.

(Proof) First, we compute the reachability problem of each node, only considering edges in H. This computation performed by a circuit with depth $O(\log^2 n)$ [7]. We connect every pair of vertices v and v' by a directed edge (v,v') when v' is reachable from v only along edges. After this operation, we consider with reachability along hyperedges. From a subset S of vertices, we can obtain another subset S' such that for each vertex v in S' there exists a hyperedge $(\{v_i, v_j\}, v)$ where v_i and v_j are in S. It is clear that it requires only a constant depth circuit for this one step traverse of hyperedges. Since a hyperedge $(\{v_{iq}, v_{jq}\}, v_{ij})$ means v_i, v_j and v_q in G are contained in the same equivalence class and v_q must be a variable vertex, at most $[\log_2 n']$ hyperedges are consecutive in the computation of Step 2. Thus we construct each equivalence class by at most $\lceil \log_2 n' \rceil$ steps of traversing hyperedges. There are at most n' equivalence classes containing variable vertices, because no variable vertex is contained in more than one equivalence class. Then we can compute the reachability problem on H by n' times alternating execution of $\lceil \log_2 n' \rceil$ steps hyperedge traversing and one step edge traversing. So we can construct a combinational logic circuit with depth $O(\log^2 n + n' \log n')$.

Q.E.D.

[Lemma 2] We can construct a combinational logic circuit with depth $O(\log^2 n)$ which decides whether a graph G' in Step 5 is acyclic.

(Proof) Using $O(\log^2 n)$ depth circuit for reachability problem of directed graphs [7], we can construct a circuit with depth $O(\log^2 n)$ for acyclic test.

Q.E.D.

From Lemma 1 and Lemma 2, we directly obtain the following theorem.

[Theorem 2] Algorithm UNIFY is implemented by a combinational logic circuit with depth $O(\log^2 n + n' \log n')$.

Since Step 2-5 in UNIFY-2 requires a circuit with depth $O(\log n')$, we have the following corollary.

[Corollary] Algorithm UNIFY-2 is implemented by a combinational logic circuit with depth $O(\log^2 n + n' \log n')$.

5. CONSIDERATIONS ON THE LOWER BOUND

In this section, we are concerned with the lower bound of the depth of circuits computing UDP and UP. It is shown that UDP is the hardest problem in problems can be solved by polynomial size circuits in the sense of depth complexity. Namely, it seems very difficult to design an efficient parallel algorithm for UDP.

Before discussing the lower bound of the depth complexity of UDP, we introduce several concepts and notations. First, we define complexity classes of decision problems. Let P be a problem on alphabet {0,1}, P_n denotes a subproblem of P with length n, i.e.,

$P_n = P \cap \{0,1\}^n$.

Thus P_n can be considered as an n-variable logic function. We define complexity classes related with size and depth of logic circuits.

PSIZE={P| For each P, there exists a polynomial p(n) such that $C(P_n)<p(n)$}

$LOG^kDEPTH=\{P|$ For each P, $D(P_n)=O(\log^k n)\}$

Since the size of a circuit for UDP constructed in the previous section according to algorithm UNIFY is bounded by a polynomial of the input length, UDP is in PSIZE.

[Definition 6] A problem P is said to be log-depth complete for PSIZE if and only if P satisfies the following two properties:

(1) P is in PSIZE.

(2) For every problem Q in PSIZE, there exists a circuit with depth O(log n) which transforms Q_n to $\{P_1, P_2, \ldots, P_m\}$ where m is bounded by a polynomial of n.

It is directly concluded that if a problem P is log-depth complete for PSIZE and P is in LOG^kDEPTH for a positive integer k then PSIZE is included in LOG^kDEPTH. However, we have known no evidence to suggest that there is some k such that PSIZE is in LOG^kDEPTH. In other words, P is the hardest problem in PSIZE concerned with the delay complexity.

Here we claim that UDP is one of such problems.

[Theorem 3] UDP is log-depth complete for PSIZE.

Before proving Theorem 3, we prepare three lemmas.

[Lemma 3] For any combinational logic circuit C over a basis {2-AND, 2-NAND} with a single output vertex and for any input vector $(x_1, x_2, \ldots, x_n)$ for C, there is a hypergraph H with rank 2, a subset S of vertices in H and a vertex v_1 such that v_1 is reachable from S if and only if C outputs 1 for the input vector. (Proof) We construct a hypergraph H as follows. For each vertex u in C, place two vertices u_0 and u_1 in H. If u is a computation vertex with label AND and there are edges (u',u) and (u",u) in C, make hyperedges $(\{u_0', u_0''\}, u_0)$, $(\{u_0', u_1''\}, u_0)$, $(\{u_1', u_0''\}, u_0)$, and $(\{u_1', u_1''\}, u_1)$ in H. If label of u is NAND, make hyperedges

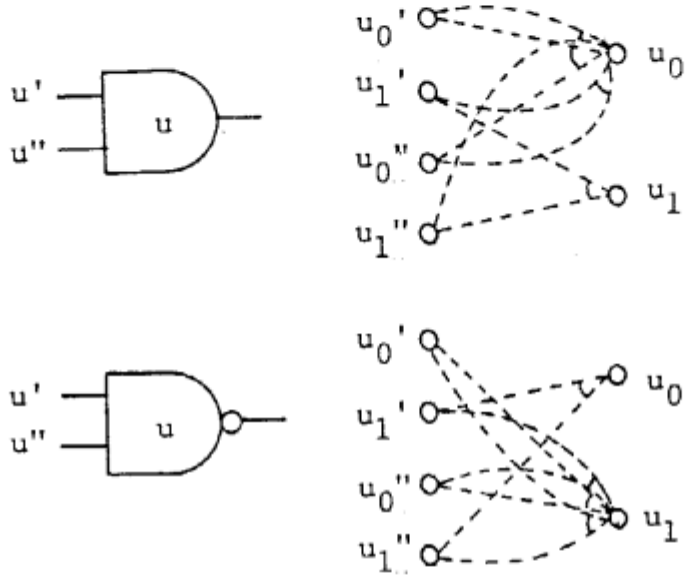


Fig.4 Conversion from Circuit to Hypergraph

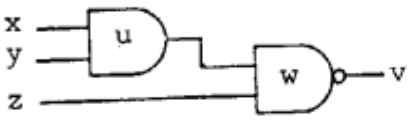
$(\{u_0', u_0''\}, u_1)$, $(\{u_0', u_1''\}, u_1)$, $(\{u_1', u_0''\}, u_1)$, and $(\{u_1', u_1''\}, u_0)$ in H (See Fig.4). Suppose that v is the output node of C and is adjacent to u. Generate a edge (u_1, v_1) in H. Let w be an input vertex with label x_i. When $x_i=1$ let w_1 be in S, and when $x_i=0$ w_0 in S. It is clear that v_1 is reachable from S if and only if C outputs 1 for a given input.

Q.E.D.

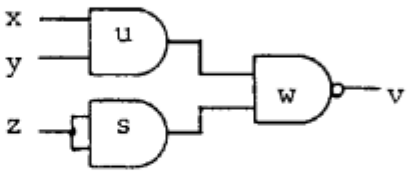
Note that the number of vertices and hyperedges (including edges) are bounded by 2size(C) and 4size(C), respectively, in the above construction.

A hypergraph H with rank 2 is said to be synchronous if and only if (1) vertices are partitioned into d levels; (2) for all edges (u,v) and all hyperedges ({u,w},v) if v is in level i then u and w should be in level i-1; (3) each vertex in odd levels has only rank 1 outgoing edges and each vertex in even levels is a source of at most one hyperedge with rank 2; (4) indegree of each vertex in even levels is just one; and (5) indegree of each vertex in odd levels except the first one is positive. We call vertices in odd levels "AND-nodes", and ones in even levels "OR-nodes".

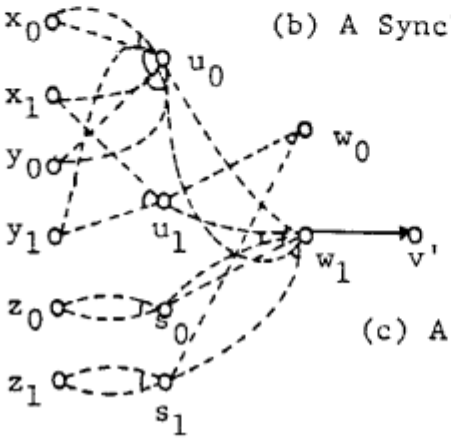
[Lemma 4] For any combinational logic circuit C in Lemma 3, we can construct a synchronous hypergraph H satisfying the condition of Lemma 3. Moreover the number of vertices and hyperedges (including edges) in H is $O(\text{size}(C)^2)$ and all vertices in S are in level 1.



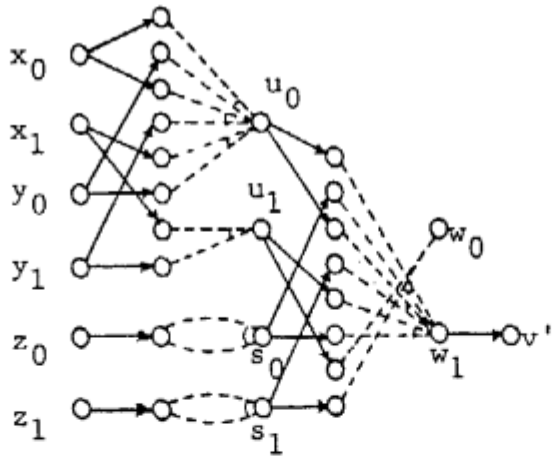
(a) A Circuit C



(b) A Synchronous Circuit C'



(c) A Hypergraph H



(d) A Synchronous Hypergraph H'

Fig.5 Transformation from Circuit
to a Synchronous Hypergraph

(Proof) It is easy to convert C to a synchronous circuit C' in which all outgoing edges from vertices in level i is pointing vertices in level i+1. The size of C' is clearly bounded by $size(C)^2$. We generate a hypergraph H from C' by the straight way mentioned in the proof of Lemma 3. It is easy to transform H to satisfy the conditions of synchronous hypergraph. Thus we obtain a synchronous hypergraph H with $O(size(C)^2)$ vertices and hyperedges.

Q.E.D.

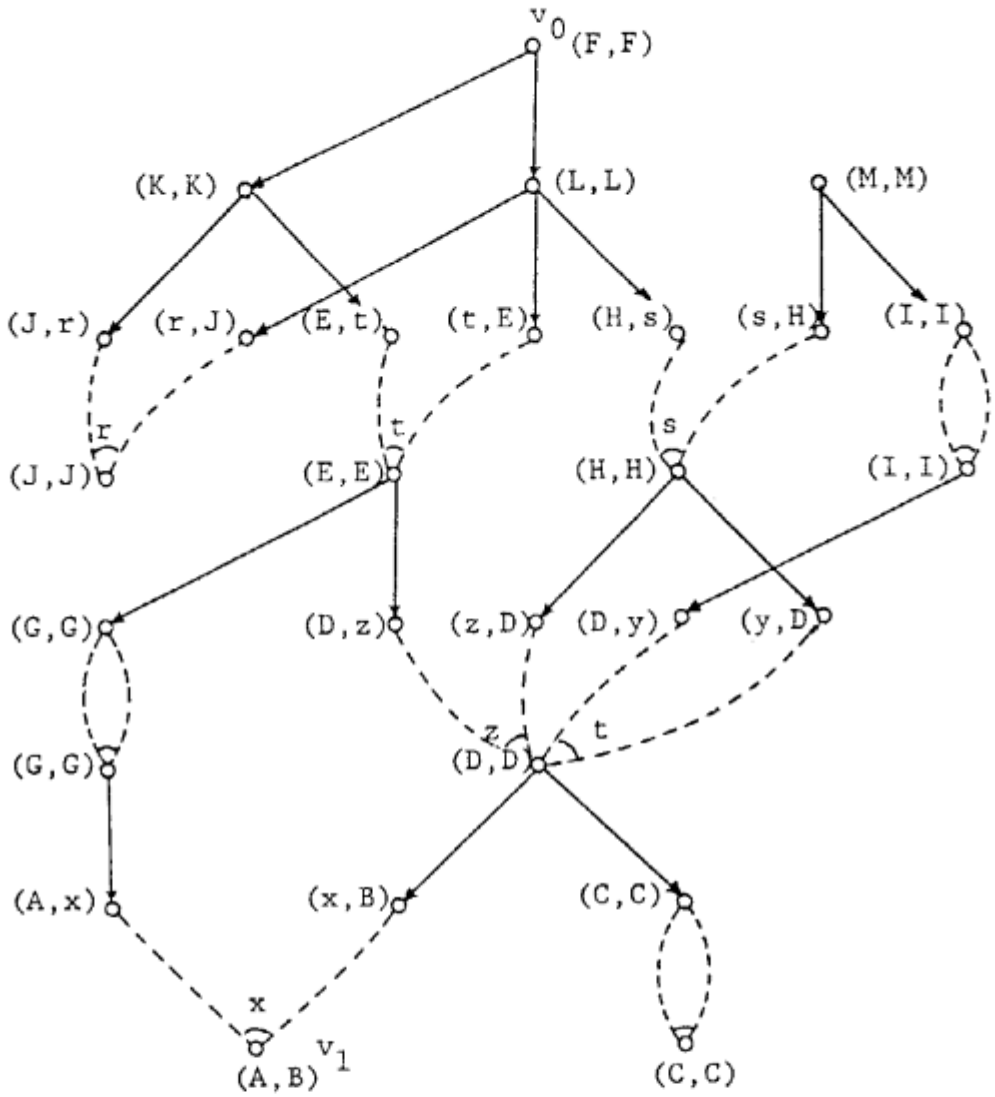
[Example 7] In Fig.5, we show a transformation process from a combinational circuit C to a synchronous hypergraph H'. First, C is reconstructed as a synchronous circuit C' in (b). By conversion in Fig.4, we have a hypergraph H in (c) which satisfies conditions (1) and (2) of synchronous hypergraphs. Separating each level into two levels, we can construct a synchronous hypergraph H' in (d).

[Lemma 5] For a synchronous hypergraph H, a subset S of vertices in level 1, and an AND-node v_1, we can find a term graph G such that UDP for G is false if and only if v_1 is reachable from S on H.

(Proof) First we put a label for each vertex in H as follows:

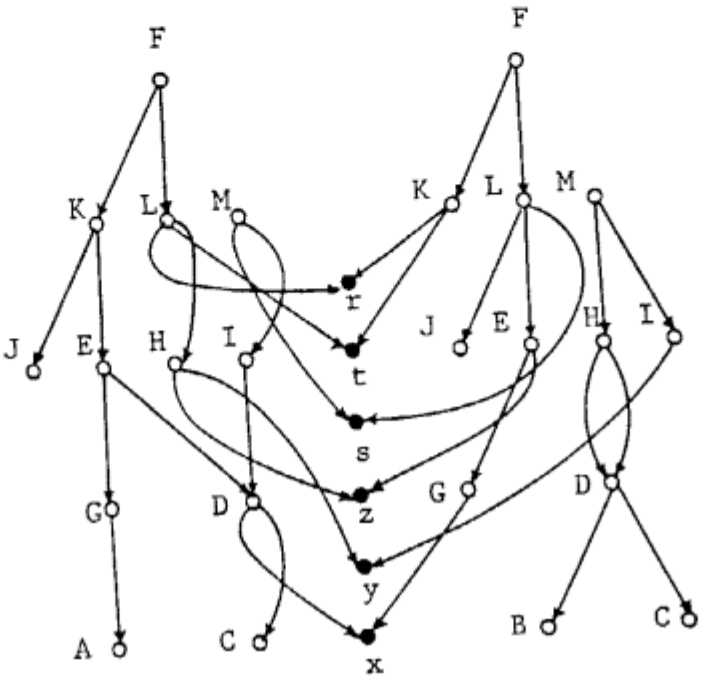
(1) For each hyperedge ({u,v},w) in H, if $u \neq v$, assign a distinct variables in X.

(2) For every AND-node v in H except v_1, assign a distinct function or constant symbol f in A_i, where outdegree of v is i. Let the label of v be (f,f). Put the label (a,b) for v_1, where a and b are distinct function or constant symbol in A.



(a) Labeling of Hypergraph H'

Fig.6 Transformation from the Reachability Problem on a Synchronous Hypergraph to the Unification Problem



(b) A Term Graph G

Fig.6 (Cont.)

(3) For every AND-node v with label (f,f) and a hyperedge ({u,w},v) assigned to x, put the label (f,f) for u if u=w, or put the label (f,x) for u and (x,f) for w if u≠w.

Make a new vertex v_0 with label (f,f) where f is different from all other function symbols used above operations. Connect v_0 with all vertices in S by directed edges from v_0. We call resulting hypergraph H'. It is trivial that v_1 is reachable from S on H if and only if v_1 is reachable from v_0 on H'. Considering H' as the hypergraph in Step 1 of algorithm UNIFY, we can construct a term graph G such that UDP for G and the reachability from v_0 to v_1 are equivalent. The construction procedure of G from H' is as follows:

(1) Generate vertices corresponding each variable in H'.

(2) For every AND-node v in H' with label (f,f), generate vertices v' and v" in G with label f.

(3) Suppose there is a edge (v,u) in H'. If the label of u is (g,g) then make edges (v',u') and (v",u") in G. If the label of u is (g,x) then generate edges (v',u') and (v",w) where w is the vertex with label x. Similarly, if the label is (x,g), then generate (v',w) and (v",u").

From the construction of H' and G, it is clear that UDP for $(G,v_0',v_0")$ fails if and only if v_1 is reachable from v_0 on H'.

Q.E.D.

[Example 8] In Fig.6, an example of the construction from a synchronous hypergraph to a term graph G is illustrated. For a given hypergraph H, we first put labels. On each hyperedge distinct variable is assigned. The label of v_1 is (A,B) but other

AND-nodes have labels (f,f). Labeling of each OR-node is decided by the rule (3) in above proof. Adding a new vertex v_0 edges according to S, we get a hypergraph H' in (a). Next we construct a term graph G in (b). In this example, the reachability problem is reduced to UDP for

F(K(J,E(G(A),D(x,C))),L(r,t,H(z,y))

and

F(K(r,t),L(J,E(G(x),z),s)).

Note that the number of vertices and edges in G are linearly proportional to the number of vertices and hyperedges in H, respectively.

Now, we return to the proof of Theorem 3.

(Proof of Theorem 3) For any problem P in PSIZE and every positive integer n, there are a polynomial p(n) and a circuit C_n such that C_n computes a subproblem P_n of P and size(C_n) is not greater than p(n). From C_n, we can construct a simple hypergraph H by Lemma 4. From Lemma 5, for a given input vector for C_n, we obtain term graph G such that UDP for G is false if and only if C_n outputs 1 for the input vector. According to the construction of H and G in the above discussion, it is easy to make a circuit transforming P_n to a UDP with constant depth. Indeed it is enough the circuit only generates edges of G in the first level from the input to P_n. As shown in the proof of Lemma 4 and 5, these adding edges in G clearly corresponds to the input for P_n. Thus the depth of the circuit is a constant independent of n. Moreover, it has been already shown in the above consideration that the length

of UDP also bounded a polynomial of n. Thus we conclude that UDP is log-depth complete for PSIZE.

Q.E.D.

By the similal discussion of this section, we can also show that unification is log-space complete for PTIME [9]. Namely, if we have an algorithm on a Turing machine for unification which uses O(log n) cells on tapes, we conclude that all problems that have polynomial time algorithms should be computable in O(log n) space complexity.

[Theorem 4] UDP is log-space complete for PTIME.
(Proof) The proof is quite similar to the above proof. It is known that the Circuit Value Problem (CVP) is log-space complete for PTIME[10]. The CVP is defined as follows: for a given combinational circuit C and an input vector for C, decide whether C outputs 1 for the input. We transform CVP to UP by the same manner in the proof of Theorem 3. We only have to consider the space consumption in the transformation from CVP to UDP. We can easily verify that the space complexity of the transformation is O(log n).

Q.E.D.

6. CONCLUSION

The unification problem is related with the reachability problem on directed hypergraph. In algorithm UNIFY, the unification problem is reduced to the reachability problem on a hypergraph. Contrary, some kind of the reachability problem can be reduced to the unification problem as shown in section 5. From the consideration on the relation between PSIZE and $LOG^{k}DEPTH$, it seems hard to design efficient algorithms for these two problems.

However, we may be able to implement a hardware which can compute UDP or UP effectively, because many practical terms for unification inherently include parallelism that may compute efficiently. It is important to examine the properties of terms which appear in practical situation for designing a good parallel unification algorithm.

Acknowledgment

The author expresses sincere appreciation to Professor Shuzo YAJIMA of Kyoto University for his suggestions and critisisms. He also thanks Associate Professor Yahiko KAMBAYASHI, Dr. Hiromi HIRAISHI and Mr. Masatoshi YOSHIKAWA in Kyoto University to their comments and discussions on directed hypergraphs. Thanks are also due to Mr. Yuuji MATSUMOTO in ETL, Dr.Kenichi HAGIHARA in Oosaka University Mr.Toshiaki KUROKAWA in ICOT and all the members of WG.5 in ICOT for their discussions. This paper is based on the result of activities of working groups for the Fifth Generation Computer System Project.

References

[1] Robinson,J.A., "A machine-oriented logic based on the resolution principle", JACM vol.12, no.1, 1965.

[2] Nilson,N.J., "Problem solving Method in artificial intelligence", McGraw-Hill, New York, 1971.

[3] Chang,C.L. and Lee,R.C.T., "Symbolic logic and mechanical theorem proving", Academic Press, New York, 1973.

[4] Kowalski,R., "Logic for problem solving", North Holland, New York, 1979.

[5] Martelli,A. and Montanari,U., "An efficient unification algorithm", ACM Trans. on Prog. Lang. and Systems, vol.4, no.2, 1982.

[6] Paterson,M.S. and Wegman,M.N., "Linear Unification", JCSS vol.16, 1978.

[7] Yasuura, H., "Studies on complexity theory of logic circuits and hardware algorithms for VLSI systems", Doctorial Dissertation of Kyoto Univ. 1982.

[8] Borodin,A., "On relating time and space to size and depth", SIAM J.Computing, vol.6, no.4, 1977.

[9] Jones, N.D. and Laaser, W.T., "Complete problems for deteministic polynomial time", Theoretical Computer Science, vol.3, no.1, 1976.

[10] Ladner, R.E., "The circuit value problem is log-space complete for P", SIGACT news, vol.7, no.1, 1975.