View metadata, citation and similar papers at core.ac.uk

brought to you by CORE

provided by Warwick Research Archives Portal Repository

Original citation:
Parberry, I. D. and Goldschlager, L. M. (1983) On the construction of parallel computers from various bases of Boolean functions. University of Warwick. Department of Computer Science. (Department of Computer Science Research Report). (Unpublished) CS-RR-048

Permanent WRAP url:
http://wrap.warwick.ac.uk/60755

Copyright and reuse:
The Warwick Research Archive Portal (WRAP) makes this work by researchers of the University of Warwick available open access under the following conditions. Copyright © and all moral rights to the version of the paper presented here belong to the individual author(s) and/or other copyright owners. To the extent reasonable and practicable the material made available in WRAP has been checked for eligibility before being made available.

Copies of full items can be used for personal research or study, educational, or not-for-profit purposes without prior permission or charge. Provided that the authors, title and full bibliographic details are credited, a hyperlink and/or URL is given for the original metadata page and the content is not changed in any way.

A note on versions:
The version presented in WRAP is the published version or, version of record, and may be cited as it appears here.For more information, please contact the WRAP Team at:
publications@warwick.ac.uk

http://wrap.warwick.ac.uk/

The University of Warwick

THEORY OF COMPUTATION

REPORT NO. 48

ON THE CONSTRUCTION OF PARALLEL COMPUTERS FROM VARIOUS BASES OF BOOLEAN FUNCTIONS

Leslie M. Goldschlager

Ian Parberry

Issued jointly as Technical Report No. 206 by :

Basser Department of Computer Science
University of Sydney
N.S.W. 2006
Australia

Department of Computer Science
University of Warwick
Coventry CV4 7AL
England

March 1983

ON THE CONSTRUCTION OF PARALLEL COMPUTERS FROM VARIOUS BASES OF BOOLEAN FUNCTIONS

Leslie M. Goldschlager
Ian Parberry

Basser Department of Computer Science
University of Sydney
N.S.W. 2006 Australia

Department of Computer Science
University of Warwick
Coventry CV4 7AL England

November 1982

Abstract

The effects of bases of two-input boolean functions are characterised in terms of their impact on some questions in parallel computation. It is found that a certain set of bases (called the P-complete set) which are not necessarily complete in the classical sense, apparently makes the circuit value problem difficult, and renders extended Turing machines and conglomerates equal to general parallel computers. A class of problems called EP arises naturally from this study, relating to the parity of the number of solutions to a problem, in contrast to previously defined classes concerning the count of the number of solutions (#P) or the existence of solutions to a problem (NP). Tournament isomorphism is a member of EP.

1. Introduction

Complexity theory seeks to formalize our intuitive notions of computational difficulty. Whilst we are intuitively sure that certain functions are more difficult to compute than others, very rarely can we actually prove it (the classical example is that of NP-complete problems, see [Cook 71], [Karp 72]). However, it is possible to classify small classes of functions according to their relative complexity. This we shall do for the two-input boolean functions.

The motivation for our classification scheme comes from examining time-bounded parallel (space bounded sequential) computations involving the two-input boolean functions. The main body of this paper is comprised of three sections. The first is on the space complexity of the circuit value problem over the two-input bases, the second the computing power of time-bounded extended Turing machines over two-input bases, and the third the ability of two-input bases to realise parallel machines.

The circuit value problem over basis B (CVP, or more precisely CVP_B) is the problem of determining, for a given combinational circuit (a circuit without feedback loops) over basis B and its inputs, the value of its output. By a circuit over basis B we mean a circuit built using gates which realize functions drawn from a boolean basis B. [Lad 75] and [Gol 77] have shown that the circuit value problem over complete bases and the monotone circuit value problem respectively are log space complete for P. This means that circuit

value problems over these bases are in a sense among the most difficult in P. For if they can be computed in $O(\log^k n)$ space then so can every member of P.

The parallel computation thesis [CKS 81], [Gol 82] states that time on any "reasonable" model of parallel computation is polynomially related to space on a deterministic Turing machine. Thus the circuit value problems over complete and monotone bases are unlikely to have an exponential speed-up on a parallel computer. We classify the two input boolean functions according to the effect which their presence in a basis has upon the complexity of the circuit value problem over that basis. We find that for the two input bases B, either CVP_B is log space complete for P or it can be computed in $O(\log^2 n)$ space.

Among the "reasonable" models of parallel machine architecture is the alternating Turing machine of [CKS 81]. This differs from the standard nondeterministic Turing machine only in the manner of defining acceptance. The states of an alternating Turing machine may be labelled AND (universal), OR (existential), NOT (negating), accept or reject. This labelling is extended to configurations in the obvious way. A configuration is deemed to be accepting if it has an accept state, or if it is universal and all successor configurations are accepting, or if it is existential and some successor configuration is accepting, or if it is negating and its successor is not an accepting configuration. We generalize this by allowing the states to be labelled with a larger range of functions, in particular the two input boolean functions. We prove that these extended Turing machines over two input boolean bases B are as powerful as parallel machines iff the circuit value problem over basis B is log space complete for P.

Furthermore, there are four language classes recognised by polynomial time bounded extended Turing machines over the bases whose CVP can be computed in $\log^2$ space. The first three are the familiar classes P, NP and CO -NP. The fourth is a previously undiscovered class which we shall call EP. Problems in the latter can be thought of (at the abstract problem level) as the parity of the number of solutions to problems in NP, in the same way that problems in CO -NP can be thought of as the complements of problems in NP.

Another previously studied model of parallel machine architectures are the conglomerates of [Gol 82]. These are essentially communication networks of synchronous finite state machines. We restrict these machines to bases of two input boolean functions. These restricted machines over basis B are as powerful as parallel machines iff the circuit value problem over basis B is log space complete for P.

2. The Circuit Value Problem

We shall use the standard definitions of space and time on a Turing machine (see for example [AHU 74], [JL 76]). Let P be the class of languages recognizable in time polynomial in the length of the input by a deterministic Turing machine.

Definitions A language A is log space transformable to B (written $A \leq_{\log} B$) if there exists a function f computable in log space such that for all w, $w \in A$ iff $f(w) \in B$. A language B is log space complete for P if $B \in P$ and for all $A \in P$, $A \leq_{\log} B$.

Lemma 1 (i) If B is log space complete for P, $B \leq_{\log} A$ and $A \in P$ then A is log space complete for P. (ii) If B is log space complete for P and is recognizable in space $O(\log^k n)$ for some constant $k \geq 1$ then every $A \in P$ can be recognized in space $O(\log^k n)$.

Definition $B_n = \{f:\{0,1\}^n \to \{0,1\}\}$ is the set of n-input boolean functions.

Definitions A circuit over basis $B \subseteq B_2$ is a sequence $C = \langle g_1,\ldots,g_n\rangle$ where each g_i is either a variable $x_1, x_2, \ldots$ (in which case it is called an input) or $f(j,k)$ for some function $f \in B$ (in which case it is called a gate), $i < j \leq k$. The value of a circuit C at gate g_i, $v(C,g_i)$ is given by

$$v(C,x_j) = x_j$$

$$v(C,f(j,k)) = f(v(C,g_j), v(C,g_k)).$$

The value of a circuit C is defined to be $v(C) = v(c,g_1)$. The circuit value problem

$$CVP_B = \{C \mid v(C) = 1\}.$$

Lemma 2 ([Lad 15]) If B is a complete basis then CVP_B is log space complete for P.

Lemma 3 ([Gol 77]) If B contains {&,V} then CVP_B is log space complete for P.

Lemma 4 If B contains $\{\to\}$, $\{\not\to\}$, $\{\leftarrow\}$ or $\{\not\leftarrow\}$ then CVP_B is log space complete for P.

Proof: $\{\to,\neg\}$ is complete, and hence by lemma 2, $CVP_{\{\to,\neg\}}$ is log space complete for P. Furthermore $CVP_{\{\to,\neg\}} \leq_{\log} CVP_B$ where $B = \{\to\}$, $\{\not\to\}$, $\{\leftarrow\}$ or $\{\not\leftarrow\}$ since $\neg x$ can be replaced by $x \to 0$, $1 \not\to x$, $0 \leftarrow x$ or $x \not\leftarrow 1$ respectively and $x \to y$ can be replaced by $x \to y$, $1 \not\to (x \not\to y)$, $y \leftarrow x$ or $(y \not\leftarrow x) \not\leftarrow 1$ respectively. □

Lemma 5 If B contains $\{\&, \leftrightarrow\}$, $\{V,\leftrightarrow\}$, $\{\&,\oplus\}$ or $\{V,\oplus\}$ then CVP_B is log space complete for P.

Proof: $CVP_{\{\&,V\}} \leq_{\log} CVP_B$ where $B = \{\&,\leftrightarrow\}$, $\{V,\leftrightarrow\}$, $\{\&,\oplus\}$ or $\{V,\oplus\}$ since

$$\begin{aligned} aVb &= (a \leftrightarrow b) \leftrightarrow (a\&b) \\ a\&b &= (a \leftrightarrow b) \leftrightarrow (aVb) \\ aVb &= (a \oplus b) \oplus (a\&b) \\ \text{and } a\&b &= (a \oplus b) \oplus (aVb) \end{aligned}$$

respectively. □

Definition Let $C = \langle g_1,\ldots,g_n\rangle$ be a circuit. Define a path of length u from g_i to g_j as follows. There is a path of length 1 from g_i to g_j if there exists $k \leq n$ such that $g_j = f(g_i,g_k)$ or $g_j = f(g_k,g_i)$. A path of length $u > 1$ from g_i to g_j is a path of length u-1 from g_i to g_l and a path of length 1 from g_l to g_j.

Definitions Let $C = \langle g_1,\ldots,g_n\rangle$ be a circuit. Define the function $odd_u(g_i,g_j)$ to be true iff there are an odd number of paths of length u in C from g_i to g_j. Further define $odd(g_i,g_j) = \bigoplus_{u=1}^{n} odd_u(g_i,g_j)$ to be true iff there are an odd number of paths (of any length) from g_i to g_j.

Lemma 6 Let $C = \langle g_1,\ldots,g_n\rangle$ be a circuit over the basis $\{\oplus\}$. For $j = 1,\ldots,n$ the value of the circuit at gate g_j is given by

$$v(g_j) = \bigoplus_{\substack{\text{inputs}\\ g_i}} [odd(g_i,g_j)\&v(g_i)]$$

Proof By induction on j, noting that & distributes over $\oplus$ (i.e. $a\&(b\oplus c) = (a\&b)\oplus(a\&c)$) □

Lemma 7 Let $C = \langle g_1,\ldots,g_n\rangle$ be a circuit over $\{\oplus\}$. If $u > d \geq 1$ then

$$odd_u(g_i,g_j) = \bigoplus_{k=1}^{n} [odd_d(g_i,g_k)\ \&\ odd_{u-d}(g_k,g_j)]$$

Proof By induction on u. □

Consider the following procedure:

```
boolean procedure path (i,j,k)
   comment returns true iff there exists an odd number of paths
   from g_i to g_j of length k.
   if k = 1
      then ∃ odd no. of connections from g_i to g_j
      else ⊕_{ℓ=1}^{n} [path(i,1,⌈k/2⌉) & path(1,j,⌊k/2⌋)]
```

Lemma 8 $path(i,j,u) = odd_u(g_i,g_j)$

Proof By induction on u, using lemma 7 with $d = \lceil u/2\rceil$. □

Lemma 9 $CVP_{\{\oplus\}}$ can be solved by a deterministic Turing machine in $O(\log^2 n)$ space.

Proof Let $C = \langle g_1,\ldots,g_n\rangle$ be a circuit over $\{\oplus\}$. Consider the program which computes

$$\bigoplus_{\substack{\text{inputs}\\ g_i}} \bigoplus_{u=1}^{n} [path(i,n,u)\&v(g_i)]$$

This uses space $O(\log^2 n)$ and

$$\bigoplus_{\substack{\text{inputs}\\ g_i}} \bigoplus_{u=1}^{n} [path(i,n,u)\&v(g_i)] = \bigoplus_{\substack{\text{inputs}\\ g_i}} I(\bigoplus_{u=1}^{n} odd_u(g_i,g_n)\&v(g_i)) \quad \text{lemma 8}$$

$$= v(n) \quad \text{lemma 6} \quad \square$$

Lemma 10 $CVP_{\{\oplus\}}$, $CVP_{\{\leftrightarrow\}}$, $CVP_{\{\oplus,\leftrightarrow\}}$ and $CVP_{\{\oplus,\leftrightarrow,\neg\}}$ are all log space equivalent.

Proof Uses double rail logic as in the proof of lemma 3, with the identity $a \oplus b = \neg(a \leftrightarrow b) = (\neg a) \leftrightarrow b$

Lemma 11 $CVP_{\{\&\}}$ and $CVP_{\{\vee\}}$ can be solved by a deterministic Turing machine in $O(\log^2 n)$ space.

Proof A simplified version of the proof of lemma 9 will suffice, since a circuit built from OR gates is true precisely when there exists a path to the output from a true input, and a circuit built from AND gates is false precisely when there exists a path to the output from a false input. $\square$

Definitions [McC 81] A function $f(x,y)$ is monotone if for all $x_1 \leq x_2$ and $y_1 \leq y_2$ $f(x_1,y_1) \leq f(x_2,y_2)$. $f(x,y)$ is linear if it can be expressed in the form

$$a_0 \oplus (a_1\&x) \oplus (a_2\&y)$$

where $a_0, a_1, a_2 \in \{0, 1\}$.

The two-input boolean functions fall into four classes induced by the properties of linearity and monotonicity (see table 1). The functions which are both linear and monotone we shall call "trivial", those which are linear only "easy", those which are monotone only "moderate" and those which are neither linear nor monotone "hard". If the gates in basis B are all easy or trivial, then CVP_B is easy (i.e., can be solved in $\log^2$ space). If B contains at most one moderate gate (and the rest trivial) then CVP_B is easy. If B contains two moderate gates, or a moderate and an easy gate, or a hard gate, then CVP_B is hard. This is summed up by the following theorem, which follows from the earlier lemmas.

Theorem 12 CVP_B is log space complete for P if either:

(i) B contains a gate which is not linear and a gate which is not monotone; or

(ii) $\{\&,\vee\} \subseteq B$

and is solvable in $O(\log^2 n)$ space otherwise.

function	name	linear	monotone	class
0	false	1	1	trivial
1	true	1	1	
x	left identity	1	1	
y	right identity	1	1	
$\neg x$	left negation	1	0	easy
$\neg y$	right negation	1	0	
$\leftrightarrow$	equivalence	1	0	
+	exclusive or	1	0	
&	and	0	1	moderate
V	or	0	1	
$\barwedge$	nand	0	0	hard
$\downarrow$	nor	0	0	
$\rightarrow$	implies	0	0	
$\not\rightarrow$	not implies	0	0	
$\leftarrow$	is implied by	0	0	
$\not\leftarrow$	is not implied by	0	0	

Table 1

Complexity classes of functions in B_2. An entry of 1 under property p of gate g indicates that g has property p (where p is monotonicity or linearity).

3. Extended Turing Machines

The definition of alternating Turing machine [CKS 81] can be generalized to allow the labelling of nonfinal states with any reasonable function.

Definition An extended Turing machine (ETM) is a nine-tuple $M = (D, B, k, Q, \Sigma, \Gamma, \delta, q_0, g)$ where

D is a problem domain ($0, 1 \in D$; $\perp \notin D$).
$B = \{f_1, \ldots, f_n\}$ is a finite set (basis) of fixed-arity functions f_i of arity $a_i \geq 0$, $f_i : D^{a_i} \to D$ $1 \leq i \leq n$.
k is the number of work tapes.
Q is a finite set of states.
Σ is a finite input alphabet ($\not\in\Sigma$ is an endmarker).
Γ is a finite work-tape alphabet ($\#\in\Gamma$ is the blank symbol).
$\delta \subseteq (Q \times \Gamma^k \times (\Sigma \cup \{\not\in\})) \times (Q \times (\Gamma - \{\#\})^k \times \{\text{left, right}\}^{k+1})$ is the next-move relation.
$q_0 \in Q$ is the initial state.
$g : Q \to B \cup D$

Definitions A configuration of an ETM $M = (D, B, k, Q, \Sigma, \Gamma, \delta, q_0, g_1)$ is an element of $C_M = Q \times \Sigma^* \times ((\Gamma - \{\#\})^*)^k \times N^{k+1}$. If α and β are configurations of M we say that β is a successor of α (written $\alpha \vdash \beta$) if β follows from α in one step according to the transition function δ. The initial configuration of M on input x is $\sigma_m(x) = (q_0, x, \underbrace{\lambda, \ldots, \lambda}_{k}, \underbrace{0, \ldots, 0}_{k+1})$

where λ denotes the empty string.

The semantics of an extended Turing machine are analogous to those of an alternating Turing machine. We insist that the transition function δ is such that, for all states $q \in Q$, every configuration containing q has exactly arity (g(q)) successors, where elements of the domain D are interpreted as functions of arity zero.

For $f:D^a \to D$ where $0 \in D, \perp \notin D$ we define the monotone extension $\hat{f}:(D\cup\{\perp\})^a \to D\cup\{\perp\}$ of f as follows:

If $\underset{\sim}{x} \in D^a$ then $\hat{f}(\underset{\sim}{x}) = f(\underset{\sim}{x})$ and for $1 \leq m \leq a$, if $\underset{\sim}{x} \in D^{m-1}$ and $\underset{\sim}{y} \in (D\cup\{\perp\})^{a-m}$

$$\hat{f}(\underset{\sim}{x},\perp,\underset{\sim}{y}) = \begin{cases} \hat{f}(\underset{\sim}{x},0,\underset{\sim}{y}) \text{ if for all } d \in D\ \hat{f}(\underset{\sim}{x},0,\underset{\sim}{y}) = \hat{f}(\underset{\sim}{x},d,\underset{\sim}{y}) \\ \perp \text{ otherwise} \end{cases}.$$

For example, the monotone extensions of some functions in B_2 are shown in table 2.

A labelling of configurations is a map

$$l : C_M \to D\cup\{\perp\}.$$

Let τ be the operator mapping labellings to labellings defined as follows. Let $M = (D, B, Q, \Sigma, \Gamma, \delta, q_0, g)$ and α be a configuration of M with state q. Assume a total ordering on the elements of δ, so that we can order those β such that $\alpha \vdash \beta$. Then

$$\tau(l)(\alpha) = \begin{cases} g(q) \text{ if } g(q) \in D \\ \hat{f}(l(\beta_1),\ldots,l(\beta_a)) \text{ if } g(q) = f \text{ and } \alpha \vdash \beta_i,\ 1 \leq i \leq a. \end{cases}$$

If we define the relation "$\leq$" by $\perp \leq d$ for all $d \in D$ then τ has a least fixed point l^* with respect to $\leq$.

Definitions An ETM M accepts x iff $l^*(\sigma_m(x)) = 1$, M rejects x iff $l^*(\sigma_m(x)) = 0$, M halts on x iff M accepts or rejects x, and the language accepted by M, $L(M) = \{x \in \Sigma^* \mid M \text{ accepts } x\}$.

Theorem 13 The extended Turing machines with computable bases accept precisely the r.e. sets.

Note that extended Turing machines with domain the natural numbers and basis {+} are the counting Turing machines of [Val 79]; and if we choose the domain to be the boolean set {0,1}, ETM's with basis {&,V,¬} are alternating Turing machines, those with basis {V} are nondeterministic Turing machines, and those with basis {&} are co-nondeterministic Turing machines. Since our interest lies with the two input boolean functions, we will henceforth restrict ourselves to extended Turing machines with D = {0,1} and $B \subseteq B_2$.

Definition $\text{PTIME}_B = \bigcup_{k>0} \text{TIME}_B(n^k)$

Definitions $\text{AP} = \text{PTIME}_{\{\&,V,\neg\}}$

$\text{NP} = \text{PTIME}_{\{V\}}$

$\text{co-NP} = \text{PTIME}_{\{\&\}}$

&	1	⊥	0
1	1	⊥	0
⊥	⊥	⊥	0
0	0	0	0

V	1	⊥	0
1	1	1	1
⊥	1	⊥	⊥
0	1	⊥	0

→	1	⊥	0
1	1	⊥	0
⊥	1	⊥	⊥
0	1	1	1

⊕	1	⊥	0
1	0	⊥	1
⊥	⊥	⊥	⊥
0	1	⊥	0

Table 2

Extensions of some functions in B_2 to domain $\{0, \perp, 1\}$.

Definition A basis B is called P-complete iff CVP_B is log space complete for P.

Theorem 14 For all P-complete bases B, $B' \subseteq B_2$

$$TIME_B(T(n)) \subseteq TIME_{B'}(dT(n))$$
$$SPACE_B(S(n)) \subseteq SPACE_{B'}(S(n))$$

for some constant d.

Proof: Theorem 2.5 of [CKS 81] proves the result for $B = \{\&, V, \neg\}$ and $B' = \{\&, V\}$. The technique used is similar to the one used to show that the monotone circuit value problem is log space complete for P (lemma 3). De Morgans laws are used to push the negations down to the final states in the same manner that they are used to push the negations back to the inputs in the monotone circuit value problem. A similar modification to the proofs of the P-completeness of all such B gives the required results. □

Thus extended Turing machines over the P-complete two input boolean bases are just as powerful, within a constant factor, as alternating Turing machines. [CKS 81] have shown that alternating Turing machines are as powerful, to within a polynomial, as any parallel machine. Theorem 14 implies that the complexity results on alternating Turing machines (notably theorems 3.1-3.4 and corollaries 3.5-3.6 of [CKS 81]) apply equally well to extended Turing machines over P-complete bases.

Theorem 15 $TIME_{\{\oplus\}}(T(n)) = TIME_{\{\leftrightarrow\}}(T(n)) = TIME_{\{\oplus,\leftrightarrow\}}(T(n)) = TIME_{\{\oplus,\leftrightarrow,\neg\}}(T(n))$

Proof A simple modification to the proof of lemma 10 suffices to give this result. □

Definition $ETIME(T(n)) = TIME_{\{\oplus\}}(T(n))$

$$EP = PTIME_{\{\oplus\}}$$

At this stage we have four interesting classes of languages accepted by polynomial time-bounded extended Turing machines. The most powerful class is that recognized by machines over a P-complete basis, exemplified by alternating Turing machines. In the light of theorem 15, we see that the remaining languages fall into the three classes accepted by polynomial time bounded extended Turing machines over the bases {&}, {V} and $\{\oplus\}$. Machines over the first two bases are nondeterministic and co-nondeterministic Turing machines respectively. Languages in the corresponding polynomial-time bounded classes NP and co-NP are well-studied (see for example [AHU 74] [G&J 79]).

The last class is EP, the class of languages accepted in polynomial time by extended Turing machines over basis $\{\oplus\}$ (E for Equivalence or Exclusive-or). The classical open problems regarding the relationships between P, NP and co-NP can be extended to include EP. For example, one might wonder whether or not NP $\cap$ co-NP $\cap$ EP = P? (See figure 1.) As with the question P $\neq$ NP? there are complete problems for the question P $\neq$ EP?

Definitions A language A is (many-one) reducible to B (written $A \leq_p B$) if there exists a function f computable in polynomial time such that for all w, $w \in A$ iff $f(w) \in B$. A language B is said to be EP-complete if $B \in EP$ and for all $A \in EP$, $A \leq_p B$.

Definitions Let $X = \{x_1,\ldots,x_m\}$ be a set of boolean variables. A truth assignment for x is a function $t : X \to \{0,1\}$. A literal over x is either a variable x or its complement $\overline{x}$. The value of a literal l under truth assignment t is given by

$$v(l,t) = \begin{cases} t(x) & \text{if } l = x \\ \neg t(x) & \text{if } l = \overline{x} \end{cases}$$

A clause over x is a set of literals over x. The value of a clause C under truth assignment t is $v(c,t) = \bigvee_{l \in c} v(l,c)$. A boolean formula over x is a set of clauses over x. The value of a formula f under truth assignment t is $v(f,t) = \bigwedge_{c \in f} v(c,t)$. The satisfiability problem SAT = {boolean formulae $f \mid \bigvee_t v(f,t)$}. Further define

parity-SAT = {boolean formulae $f \mid \bigoplus_t v(f,t)$}.

Theorem 16 parity-SAT is EP-complete.

Proof Clearly parity-SAT $\in$ EP. We follow the proof of Cook's theorem (see for example [AHU 74]). Given an extended Turing machine M with $L(M) \in EP$, we can encode it as a boolean formula, as if it were a nondeterministic Turing machine. Without loss of generality, assume that M has only exclusive-or states. Then M accepts input x iff there are an odd number of accepting computation paths of x iff there are an odd number of satisfying assignments. to the boolean formula of M. □

Similarly, determining the parity of the number of solutions to NP-complete problems is EP-complete provided the reduction from SAT is solution-preserving. The generalized Ladner's theorem [KS 80] tells us that (provided P $\neq$ EP) there are problems in EP which are neither in P nor EP-complete. A candidate is tournament isomorphism, which is not known to be in P (the best known algorithm is the $n^{c \cdot \log n}$ time algorithm of [Luk 80]). Tournament isomorphism is in EP since the automorphism group of a tournament has odd order (hence the number of isomorphism between two tournaments is either zero or odd).

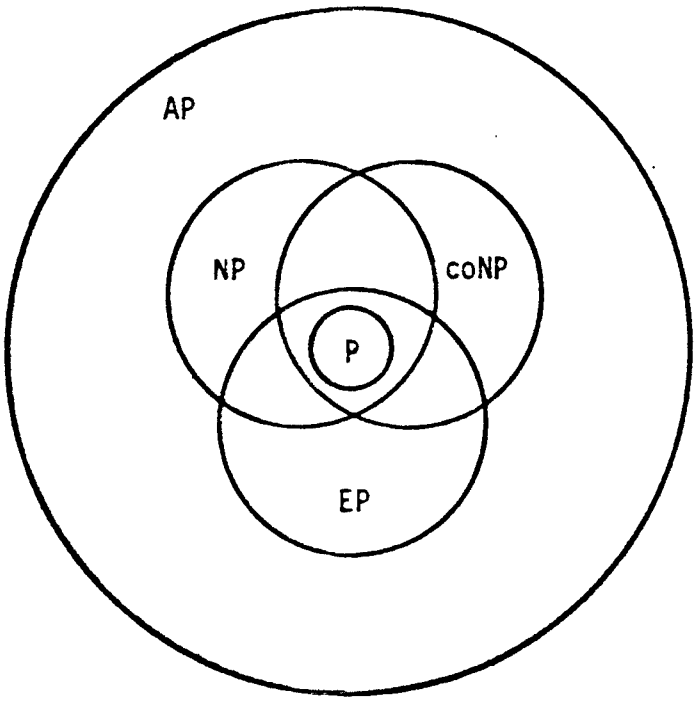

Figure 1 - The Class EP

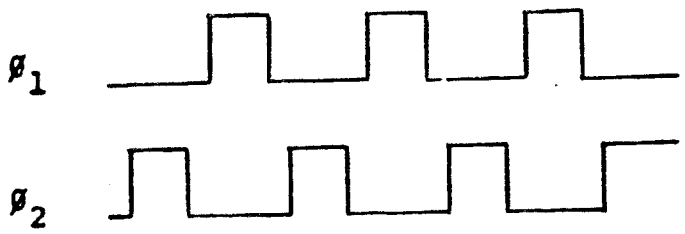

Figure 2 - Two-phase Non-overlapping Clock

4. Conglomerates

Conglomerates consisting of synchronous finite state machines communicating via an interconnection network are another model of parallel machines. If the pattern of the interconnections is computable in polynomial space (or polynomial parallel time) then the resulting class of conglomerates are as powerful, to within a polynomial, as any parallel machine [Gol 82].

Definitions Conglomerates, and the notion of computation, acceptance and execution time are defined as in [Gol 82].

Definition A conglomerate over basis B consists of gates of B communicating via an interconnection network. Note that the network may be cyclic, unlike the combinational circuits in CVP_B. There is also available a regular "two phase non-overlapping clock" [MC 81] (see figure 2) whose period is no greater than a finite number of gate delays. Thus the computations of the gates can be made synchronous, and execution time is defined to be the number of clock pulses. The input convention is that the input bits $b_1, b_2, \ldots, b_n$ are represented at time zero by having the inputs to gate i equal to b_i and the inputs to gate i' equal to $\neg b_i$. Acceptance and the complexity of the interconnection network are defined analogously to conglomerates.

Definition $\text{CONG-TIME}_B(T(n))$ ($\text{CONGLOMERATE-TIME}(T(n))$) denotes the class of languages accepted by conglomerates over basis B (conglomerates) in time $T(n)$ with the interconnection network computable in polynomial parallel time. Since conglomerates over a finite basis are a special case of general conglomerates, it is obvious that $\text{CONG-TIME}_B(T(n)) \subseteq \text{CONGLOMERATE-TIME}(T(n))$. We shall now investigate the converse of this result.

Theorem 17: For all complete bases B,

$$\text{CONGLOMERATE-TIME}(T(n)) \subseteq \text{CONG-TIME}_B(dT(n))$$

for some constant d.

Proof Each finite state machine in the conglomerate can be replaced by an equivalent combinational circuit over basis B, and a finite number of memory elements. These memory elements can be clocked by the regular clock pulses and their outputs fed back into the inputs of the combinational circuit in order to simulate the finite state machines in the standard way. If B is complete, the memory elements may be constructed using a cyclic network of gates of B as in the normal "flip-flop" circuits. E.g., if B = {&,V,¬} the flip-flop could be as shown in figure 3.

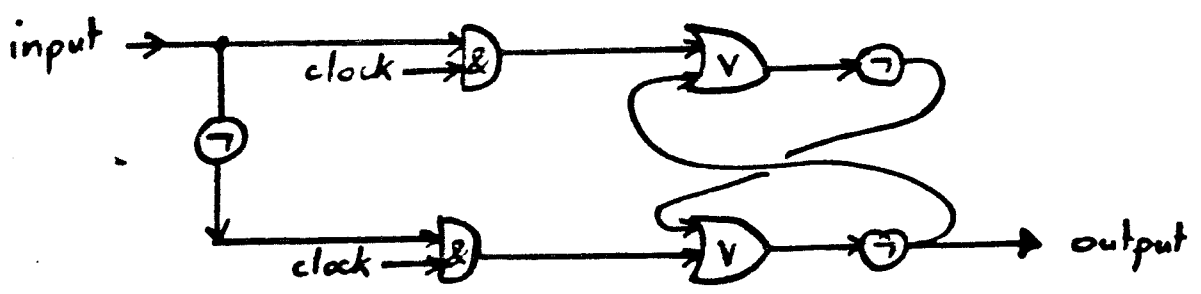

Figure 3 - Standard Flip-flop

Note that consecutive flip-flops will need to be clocked with opposite phases (ϕ_1 and ϕ_2) of the clock in order to prevent race conditions. Because the conglomerate consists of finite state machines, only a finite number of gates from B are required to simulate each one and so the execution time will be increased by at most a constant factor d. Similarly, the complexity of the interconnection pattern will not be increased by more than a constant factor.

Theorem 18: For all P-complete bases B,

$$\text{CONGLOMERATE-TIME}(T(n)) \subseteq \text{CONG-TIME}_B(dT(n))$$

for some constant d.

Proof: Consider the case when B = {&,V}. Again using the "double rail logic" idea in lemma 3, it is straightforward to replace the combinational circuits in the proof of theorem 17 with gates from B by pushing the negations back to the inputs. Thus each edge of the interconnection network will be simulated by two edges of the {&,V} network, one carrying the negation of the other. Now the memory elements can be simulated as in figure 4.

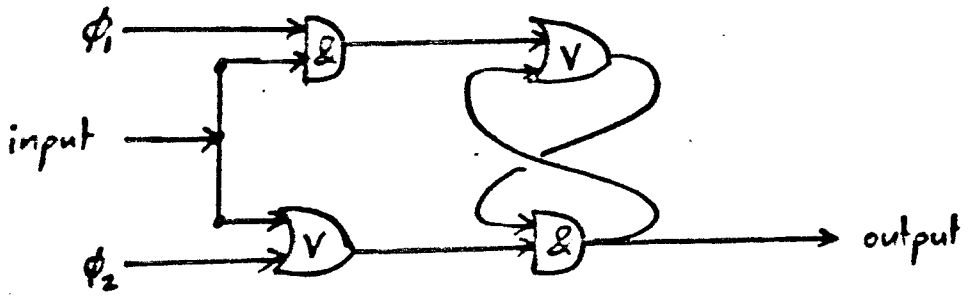

Figure 4 - "Monotone Memory"

Thus the theorem holds when B = {&,V}, neither the execution time nor the complexity of the interconnection network increasing by more than a constant factor. Now the theorem follows for all other P-complete B using the techniques of lemmas 4 and 5. □

Theorem 19: If some basis B is not P-complete, then conglomerates over basis B cannot in general simulate conglomerates (or any other general purpose parallel computer).

Proof: Assume to the contrary that some basis B which is not P-complete can be used to simulate an arbitrary conglomerate. Then in particular, it can simulate the conglomerate which computes the NAND function NAND (b_1,b_2) = $f(b_1, \neg b_1, b_2, \neg b_2)$. Thus a (possibly cyclic) network of gates from B can simulate the NAND function in some particular time t. Now such a network can be "unrolled" into a combinational circuit with depth at most dt for some constant d [Sav 72]. Note also that any clock signals coming into a gate in the unrolled circuit can be set to a constant value representing the value of that clock signal at the particular time in the computation which the depth of that gate represents. Thus there is a fixed combinational circuit over basis B which computes the NAND function from the values of two inputs and their negations. Hence $\text{CVP}_{\{\&\}} \leq \text{CVP}_B$, contradiction. □

Loosely speaking, we can summarise this section by saying that a particular basis $B \subseteq B_2$ can be used to build general purpose parallel machines iff B is P-complete.

5. Conclusions

We have examined bases of two input boolean functions, and defined the notion of a basis being P-complete. With reference to table 1, a basis is P-complete if it contains at least one "hard" function, or two "moderate", or a "moderate" and an "easy". The remaining bases of two input boolean functions are not believed to be P-complete (unless P = SPACE($\log^k n$) for some constant k).

If a basis is P-complete, then the circuit value problem over that basis is probably inherently sequential, and extended Turing machines and conglomerates over that basis are powerful parallel machines. The remaining bases are not suitable for building general purpose parallel machines, and the circuit value problem over them can be solved quickly on a parallel machine.

However the bases which do not appear to be P-complete can be further classified into four groups according to their apparent effect on the computational power of extended Turing machines. These four groups are exemplified by {V}, {&}, the one-input functions, and $\{\oplus\}$, corresponding to non-deterministic, co-nondeterministic, deterministic and the new class of "parity" computations.

6. Further work

How do planar circuits behave over different bases? For example, it appears that {&,V} is not a powerful computational basis for planar circuits [Gol 80].

It would also be nice to know more about the class EP. Is this identical to a previously studied class? What is the relationship between EP and P, NP and co-NP? Is there a "natural" problem which is EP-complete?

Acknowledgements

We would like to thank Michael Hickey for his contribution to the "monotone memory" elements of section 4.

References

[Lad 75] R.E. Ladner, "The circuit value problem is log space complete for P", SIGACT News 7(1) (1975) 18-20.

[Gol 77] L.M. Goldschlager, "The monotone and planar circuit value problems are log space complete for P", SIGACT News 1(2) (1977) 25-29.

[Gol 80] L.M. Goldschlager, "A space efficient algorithm for the monotone planar circuit value problem", Inf. Proc. Letters 10(1) (1980).

[McC 81] W.F. McColl, "Planar crossovers", IEEE Trans. Computers, C-30 (3) (1981).

[CKS 81] A.C. Chandra, D.C. Kozen and L.J. Stockmeyer, "Alternation", JACM 28 (1) (1981).

[Val 79] L.G. Valiant, "The complexity of enumeration and reliability problems", Siam J. Comput., 8(3) (1979).

[Luk 80] E.M. Luks, "Isomorphism of graphs of bounded valence can be tested in polynomial time", Proc. 21st Annual IEEE Symp. On Foundations of Comp. Sci., (1980).

[Cook 71] S.A. Cook, "The Complexity of theorem proving procedures", Proc. 3rd ACM Symp. on Theory of Computing (1971) 151-158.

[Karp 72] R.M. Karp, "Reducibility among combinatorial problems", in R.E. Miller and J.W. Thatcher, Eds, "Complexity of Computer Computations", Plenum Press, New York, (1972).

[AHU 74] A.V. Aho, J.E. Hopcroft and J.D. Ullman, "The design and analysis of com puter algorithms", Addison-Wesley, Reading, MA, (1974).

[GJ 79] M.R. Garey and D.S. Johnson, "Computers and intractability: a guide to the theory of NP-completeness", W.H. Freeman, San Francisco (1979).

[KS 80] T. Kamimura and G. Slutzki, "Some results on pseudopolynomial algorithms", Tech. Report TR-80-6, Comp. Sci. Dept, University of Kansas (Nov. 1980).

[Gol 82] L.M. Goldschlager, "A universal interconnection pattern for parallel computers", JACM, Vol. 29, No. 4, 1982.

[Sav 72] J.E. Savage, "Computational work and time on finite machines", JACM, Vol. 19, No. 4, pp. 600-674, 1972.

[JL 76] N.D. Jones and W.T. Laaser, "Complete Problems for Deterministic Polynomial Time", TCS 3(1) 1976, pp 105-117.

[MC 81] C. Mead and L. Conway, An Introduction to VLSI Systems, Addison-Wesley, 1981.