

Journal of computer and system sciences 31 (1986) 43-58  
© 1986 North-Holland Publishing Company. All rights reserved.

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

Leslie M. GOLDSCHLAGER

Basser Department of Computer Science, University of Sydney, New South Wales 2006, Australia

Ian PARBERRY

Department of Computer Science, The Pennsylvania State University, University Park, PA 16802,  
U.S.A.

Communicated by J.D. Ullman

Received April 1983

Revised March 1984, August 1985

**Abstract.** The effects of bases of two-input Boolean functions are characterized 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 networks of Boolean gates equal to general parallel computers. A class of problems called EP naturally arises 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 (NP). Tournament isomorphism is a member of EP.

### 1. Introduction

Complexity theory seeks to formalize our intuitive notions of computational difficulty. Whilst in many cases 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 [3, 11]). However, it is often possible to classify small classes of functions according to their relative complexity, as we shall do here for the two-input Boolean functions. It has also recently been shown [8] that our results hold equally well for Boolean functions with more than two inputs.

The motivation for our classification scheme comes from examining time-bounded parallel (equivalently, space-bounded sequential) computations involving the two-input Boolean functions. The main body of this paper is broken up into three sections. The first is on the space complexity of the circuit value problem over

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 realize parallel machines.

The *circuit value problem over basis B* ( $CVP_B$ , or more precisely,  $CVP_{B, \leq}$ ) is the problem of determining, for a given combinational circuit and its inputs, the value of the output. By a combinational circuit over basis  $B$  we mean a circuit without feedback loops, built using gates which realize functions drawn from a set  $B$ . Ladner [12] and Goldschlager [5] 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 [2, 7] 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 speedup 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 [2]. 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 configuration 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 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 recognized by polynomial time bounded extended Turing machines over the bases whose circuit value problem can be computed in  $\log^2$  space. The first three are the familiar classes P, NP, and Co-NP. The fourth is a less familiar class which we shall call EP. A language in EP is the set of strings for which there is an even parity (or equivalently, odd parity, since we will find that EP is closed under complementation) number of solutions to a given problem, just as NP is the set of strings for which there is at least one solution.

Another previously studied model of parallel machine architecture is the conglomerate of [7]. These are communication networks of synchronous finite-state machines. We restrict these machines to bases  $B$  of two-input Boolean functions and show

that they are as powerful as parallel machines iff the circuit value problem over  $B$  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, [1, 9]). Let P be the class of languages recognizable in polynomial time by a deterministic Turing machine.

**Definition.** A language  $A$  is *log space transformable* to  $B$  (written  $A \leq_{\text{log}} B$ ) 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  $A \in P$ ,  $A \leq_{\text{log}} B$ .

**Lemma 2.1.** (i) If  $B$  is log space complete for P,  $B \leq_{\text{log}} A$  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  $O(\log^k n)$  space for some constant  $k \geq 1$ , then every  $A \in P$  can be recognized in  $O(\log^k n)$  space.

**Definition.**  $B_n = \{f : \{0, 1\}^n \rightarrow \{0, 1\}\}$  is the set of  $n$ -input Boolean functions. We will denote the elements of  $B_2$  by 0, 1,  $x$ ,  $y$ ,  $\neg x$ ,  $\neg y$ ,  $\leftrightarrow$ ,  $\oplus$ ,  $\wedge$ ,  $\vee$ ,  $\uparrow$ ,  $\downarrow$ ,  $\rightarrow$ ,  $\neq$ ,  $\leftarrow$ ,  $\leftarrow'$  for 0, 1, left identity, right identity, left negation, right negation, equivalence, exclusive-or, and, or, nand, nor, implies, not implies, is implied by, is not implied by, respectively.

**Definition.** A *circuit* over basis  $B \subseteq B_2$  is a sequence  $C = (g_1, \dots, g_n)$ , where each  $g_i$  is either a variable  $x_1, x_2, \dots$  (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*),  $i > j, k$ . An *input assignment* is an assignment of values  $v(x_i) \in \{0, 1\}$  to the variables  $x_i$  of C. The *value* of a circuit C at gate  $g_n$ ,  $v(C, g_n)$ , is given by

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

The *value* of a circuit C is defined to be  $v(C) = v(C, g_n)$ . The *circuit value problem*  $\text{CVP}_B = \{C \mid v(C) = 1\}$ .

**Lemma 2.2** (Ladner [12]). *If B is a complete basis, then  $\text{CVP}_B$  is log space complete for P.*

**Lemma 2.3** (Goldschlag [5]). *If  $\{\wedge, \vee\} \subseteq B$ , then  $\text{CVP}_B$  is log space complete for P.*

**Lemma 2.4.** *If  $B \cap \{\rightarrow, \neq, \leftarrow, \leftarrow'\} \neq \emptyset$ , then  $\text{CVP}_B$  is log space complete for P.*

**Proof.**  $\{\rightarrow, \neg\}$  is complete, and hence, by Lemma 2.2,  $\text{CVP}_{\{\rightarrow, \neg\}}$  is log space complete for P. Furthermore,  $\text{CVP}_{\{\rightarrow, \neg\}} \leq_{\text{log}} \text{CVP}_B$ , where  $B = \{\rightarrow\}, \{\neq\}, \{\leftarrow\}$ , or  $\{\neq\}$  since  $\neg x$  can be replaced by  $x \rightarrow 0$ ,  $1 \neq x$ ,  $0 \leftarrow x$ , or  $x \leftarrow 1$ , respectively, and  $x \rightarrow y$  can be replaced by  $x \rightarrow y$ ,  $1 \neq (x \neq y)$ ,  $y \leftarrow x$ , or  $(y \neq x) \neq 1$ , respectively.  $\square$

**Lemma 2.5.** If  $B$  contains  $\{\wedge, \leftrightarrow\}$ ,  $\{\vee, \leftrightarrow\}$ ,  $\{\wedge, \oplus\}$ , or  $\{\vee, \oplus\}$ , then  $\text{CVP}_B$  is log space complete for  $P$ .

**Proof.**  $\text{CVP}_{\{\wedge, \vee\}} \leq_{\log} \text{CVP}_B$ , where  $B = \{\wedge, \leftrightarrow\}$ ,  $\{\vee, \leftrightarrow\}$ ,  $\{\wedge, \oplus\}$ , or  $\{\vee, \oplus\}$  since

$$\begin{aligned} a \vee b &= (a \leftrightarrow b) \leftrightarrow (a \wedge b), & a \vee b &= (a \oplus b) \oplus (a \wedge b), \\ a \wedge b &= (a \leftrightarrow b) \leftrightarrow (a \vee b), & a \wedge b &= (a \oplus b) \oplus (a \vee b), \end{aligned}$$

respectively.  $\square$

**Definition.** Let  $C = \langle g_1, \dots, g_n \rangle$  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 a  $k \leq n$  such that  $g_j = f(g_k, g_i)$  or  $g_i = f(g_k, g_j)$ . A path of length  $u > 1$  from  $g_i$  to  $g_j$  is a path of length  $u - 1$  from  $g_i$  to  $g_k$  and a path of length 1 from  $g_k$  to  $g_j$ .

**Definition.** Let  $C = \langle g_1, \dots, g_n \rangle$  be a circuit. Define the function  $\text{odd}_u(g_i, g_j)$  to be true iff there is an odd number of paths of length  $u$  from  $g_i$  to  $g_j$  in  $C$ . Further, define  $\text{odd}(g_i, g_j)$  to be true iff there is an odd number of paths (of any length) from  $g_i$  to  $g_j$ . Thus,

$$\text{odd}(g_i, g_j) = \bigoplus_{u=1}^n \text{odd}_u(g_i, g_j).$$

**Lemma 2.6.** Let  $C = \langle g_1, \dots, g_n \rangle$  be a circuit over basis  $\{\oplus\}$ . For  $1 \leq j \leq n$  the value of the circuit at gate  $g_j$  is given by

$$v(g_j) = \bigoplus_{\text{inputs } g_i} (\text{odd}(g_i, g_j) \wedge v(g_i)).$$

**Proof.** The proof follows by induction on  $j$ , noting that " $\wedge$ " distributes over " $\oplus$ " (i.e.,  $a \wedge (b \oplus c) = (a \wedge b) \oplus (a \wedge c)$ ).  $\square$

**Lemma 2.7.** Let  $C = \langle g_1, \dots, g_n \rangle$  be a circuit over  $\{\oplus\}$ . If  $u > d \geq 1$ , then

$$\text{odd}_u(g_i, g_j) = \bigoplus_{k=1}^d (\text{odd}_d(g_i, g_k) \wedge \text{odd}_{u-d}(g_k, g_j)).$$

**Proof.** The proof follows by induction on  $u$ .  $\square$

Consider the following procedure.

**Boolean procedure**  $\text{path}(i, j, k)$

comment returns true iff there exists an odd number of paths from  $g_i$  to  $g_j$  of length  $k$ .

if  $k = 1$  then  $\exists$  an odd number of connections from  $g_i$  to  $g_j$

else  $\bigoplus_{l=1}^n (\text{path}(i, l, \lfloor \frac{k}{2} \rfloor) \wedge \text{path}(l, j, \lfloor \frac{k}{2} \rfloor))$ .

**Lemma 2.8.**  $\text{path}(i, j, u) = \text{odd}_u(g_i, g_j)$ .

**Proof.** The proof follows by induction on  $u$ , using Lemma 2.7 with  $d = \lceil \frac{1}{2}u \rceil$ .  $\square$

**Lemma 2.9.**  $CVP_{\{\oplus\}}$  can be solved by a deterministic Turing machine in  $O(\log^2 n)$  space.

**Proof.** Let  $C = (g_1, \dots, g_n)$  be a circuit over  $\{\oplus\}$ . Consider the program which computes

$$\bigoplus_{\text{inputs } g_i} \bigoplus_{u=1}^n (\text{path}(i, n, u) \wedge v(g_i)).$$

This uses  $O(\log^2 n)$  space (since the depth of recursion is  $O(\log n)$ ), and

$$\begin{aligned} & \bigoplus_{\text{inputs } g_i} \bigoplus_{u=1}^n (\text{path}(i, n, u) \wedge v(g_i)) \\ &= \bigoplus_{\text{inputs } g_i} \left( \left( \bigoplus_{u=1}^n \text{odd}_u(g_n, g_n) \right) \wedge v(g_i) \right) \quad \text{by Lemma 2.8} \\ &= v(g_n) \quad \text{by Lemma 2.6} \end{aligned}$$

as required.  $\square$

**Lemma 2.10.**  $CVP_{\{\oplus\}}, CVP_{\{\leftrightarrow\}}, CVP_{\{\oplus, \leftrightarrow\}}, \text{ and } CVP_{\{\oplus, \leftrightarrow, \neg\}}$  are all log space equivalent.

**Proof.** To prove the lemma, use the identities  $a \oplus b = \neg(a \leftrightarrow b) = (\neg a) \leftrightarrow b$ .  $\square$

**Lemma 2.11.**  $CVP_{\{s\}}$  and  $CVP_{\{v\}}$  can be solved by a deterministic Turing machine in  $O(\log^2 n)$  space.

**Proof.** A simplified version of the proof of Lemma 2.9 will suffice, since a circuit built from OR gates is true precisely when there exists a path from the output to 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.  $\square$

**Definition.** A function  $f(x, y)$  is *monotone* if, for all  $x_1 \leq x_2$  and  $y_1 \leq y_2$ ,  $f(x_1, y_1) \leq f(x_2, y_2)$ . Function  $f(x, y)$  is *linear* if it can be expressed in the form

$$a_0 \oplus (a_1 \wedge x) \oplus (a_2 \wedge y),$$

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). We call the functions which are both linear and monotone '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, the  $CVP_B$  is hard. This is summed up by the following theorem, which follows from the above lemmas.

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).

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	
$\oplus$	Exclusive-or	1	0	
$\wedge$	And	0	1	Moderate
$\vee$	Or	0	1	
$\uparrow$	Nand	0	0	Hard
$\downarrow$	Nor	0	0	
$\rightarrow$	Implies	0	0	
$\not\rightarrow$	Not implies	0	0	
$\leftarrow$	Implied by	0	0	
$\not\leftarrow$	Not implied by	0	0	

**Theorem 2.12.** CVP<sub>B</sub> is log space complete for P if either:

- (1)  $B$  contains a gate which is not linear, and a gate which is not monotone, or
- (2)  $\{\wedge, \vee\} \subseteq B$ ,

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

### 3. Extended Turing machines

The definition of an alternating Turing machine (ATM) in [2] 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 the problem domain ( $0, 1 \in D; \perp \in D$ ),  $B = \{f_1, \dots, f_n\}$  is a finite set (basis) of fixed-arity functions  $f_i$  with arity  $a_i \geq 0$  respectively,  $f_i : D^{a_i} \rightarrow D$ ,  $1 \leq i \leq n$ ,  $k$  is the number of work tapes,  $Q$  is a finite set of states,  $\Sigma$  is a finite input alphabet,  $\Gamma$  is a finite work-tape alphabet,  $\delta \subseteq (Q \times \Gamma^k \times \Sigma) \times (Q \times \Gamma^k \times \{\text{left}, \text{right}\}^{k+1})$  is the next-move relation,  $q_0 \in Q$  is the initial state, and  $g : Q \rightarrow B \cup D$ .

**Definitions.** A configuration of an ETM  $M = (D, B, k, Q, \Sigma, \Gamma, \delta, q_0, g)$  is an element of  $C_m = Q \times \Sigma^* \times (\Gamma^*)^k \times \mathbb{N}^{k+1}$ , where  $\mathbb{N}$  denotes the set of natural numbers. If  $\alpha$  and  $\beta$  are configurations of  $M$ , we say that  $\beta$  is a successor of  $\alpha$  (written  $\alpha \vdash \beta$ ) if

$\beta$  follows from  $\alpha$  in one step according to the transition function  $\delta$ . The initial configuration of  $M$  on input  $x$  is  $\sigma_M(x) = (q_0, x, \lambda^k, 0^{k+1})$ , where  $\lambda$  denotes the empty string.

The semantics of an extended Turing machine are analogous to those of an alternating Turing machine. We give a brief sketch, following the formalism of [2]. We insist that the transition function  $\delta$  is such that, for all states  $q \in Q$ , every configuration containing  $q$  has exactly  $g(q)$  successors, where elements of the domain  $D$  are interpreted as functions of arity 0.

For  $f: D^a \rightarrow D$  where  $0 \in D$ ,  $\perp \in D$  we define the monotone extension  $\hat{f}: (D \cup \{\perp\})^a \rightarrow D \cup \{\perp\}$  of  $f$  as follows. If  $x \in D^a$ , then  $\hat{f}(x) = f(x)$  and, for  $1 \leq m \leq a$ , if  $x \in D^{m-1}$  and  $y \in (D \cup \{\perp\})^{a-m}$ ,

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

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

A labelling of configurations is a map

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

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

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

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

**Definition.** 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  $M$ ,  $L(M) = \{x \in \Sigma^* \mid M \text{ accepts } x\}$ .

Table 2  
Extensions of some functions in  $B_2$  to domain  $\{0, \perp, 1\}$ .

$\wedge$	1	$\perp$	0	$\vee$	1	$\perp$	0
1	1	$\perp$	0	1	1	1	1
$\perp$	$\perp$	$\perp$	0	$\perp$	$\perp$	$\perp$	0
0	0	0	0	0	1	$\perp$	0
$\oplus$	1	$\perp$	0	$\oplus$	1	$\perp$	0
1	1	$\perp$	0	1	0	$\perp$	1
$\perp$	1	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$
0	1	1	1	0	1	$\perp$	0

**Theorem 3.1.** *The extended Turing machines with computable bases precisely accept the r.e. sets.*

Note that extended Turing machines with domain the natural numbers and basis  $\{+\}$  are the counting Turing machines of Valiant [15]; and if we choose the domain to be the Boolean set  $\{0, 1\}$ , ETM's with basis  $\{\wedge, \vee, \neg\}$  are alternating Turing machines, those with basis  $\{\vee\}$  are nondeterministic Turing machines, and those with basis  $\{\wedge\}$  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  $D = \{0, 1\}$ ,  $B \subseteq B_2$ .

The concepts of ETM time and space can be defined in the same manner as ATM time and space [2].

**Definition.**  $\text{TIME}_B(T(n))$  and  $\text{SPACE}_B(S(n))$  are the class of languages accepted by an ETM over basis  $B$  in  $T(n)$  time and  $S(n)$  space, respectively.

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

**Definition**  $\text{AP} = \text{PTIME}_{\{\wedge, \vee, \neg\}}$ ,  $\text{NP} = \text{PTIME}_{\{\vee\}}$ , and  $\text{Co-NP} = \text{PTIME}_{\{\wedge\}}$ .

**Definition.** A basis is called P-complete iff  $\text{CVP}_B$  is log space complete for P.

**Theorem 3.2.** *For all P-complete bases  $B$ ,  $B' \subseteq B_2$ ,*

$$\text{TIME}_B(T(n)) \subseteq \text{TIME}_{B'}(d \cdot T(n)), \quad \text{SPACE}_B(S(n)) \subseteq \text{SPACE}_{B'}(S(n))$$

for some constant  $d$ .

**Proof.** In [2, Theorem 2.5] the result is proved for  $B = \{\wedge, \vee, \neg\}$  and  $B' = \{\wedge, \vee\}$ . The technique used is similar to the one used to show that the monotone circuit value problem is log space complete for P (Lemma 2.3). De Morgan's laws are used to push the negations down to the final states in the same manner as 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.  $\square$

Thus, extended Turing machines over the P-complete two-input Boolean bases are just as powerful, to within a constant factor, as alternating Turing machines. Chandra, Kozen and Stockmeyer [2] have shown that alternating Turing machines are as powerful, to within a polynomial, as any parallel machine. Theorem 3.2 implies that the complexity results on alternating Turing machines (notably [2, Theorems 3.1–3.4, and Corollaries 3.5 and 3.6]) apply equally well to extended Turing machines over P-complete bases.

**Theorem 3.3**

$$\text{TIME}_{\{\oplus\}}(T(n)) = \text{TIME}_{\{\leftrightarrow\}}(T(n)) = \text{TIME}_{\{\oplus, \leftrightarrow\}}(T(n)) = \text{TIME}_{\{\oplus, \leftrightarrow, \neg\}}(T(n)).$$

**Proof.** A simple modification to the proof of Lemma 2.10 suffices to give this result.  $\square$

**Definition.**  $\text{ETIME}(T(n)) = \text{TIME}_{\{\oplus\}}(T(n))$  and  $\text{EP} = \text{PTIME}_{\{\oplus\}}$

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 3.3, we see that the remaining languages fall into the three classes accepted by polynomial time bounded extended Turing machines over the bases  $\{\wedge\}$ ,  $\{\vee\}$ , 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, [1, 4]).

The last class is EP, the class of languages accepted in polynomial time by extended Turing machines over basis  $\{\oplus\}$  (E for Equivalence of 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  $\text{NP} \cap \text{Co-NP} \cap \text{EP} = \text{P}$  (see Fig. 1)? As with the question ' $\text{P} \neq \text{NP}$ ' there are complete problems for the question ' $\text{P} \neq \text{EP}$ '.

**Definition** 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  $B \in \text{EP}$  and, for all  $A \in \text{EP}$ ,  $A \leq_p B$ .

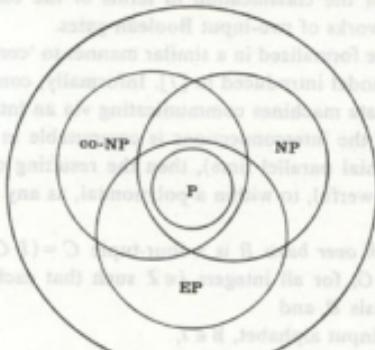


Fig. 1. The class EP.

**Definition.** Parity-SAT is the set of Boolean formulae which have an odd number of satisfying assignments.

**Theorem 3.4.** Parity-SAT is EP-complete.

**Proof.** Clearly, parity-SAT  $\in$  EP. We follow the proof of Cook's theorem (see, for example, [1]). 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 is an odd number of accepting computation paths of  $x$  iff there is an odd number of satisfying assignments to the Boolean formula of  $M$ .  $\square$

Similarly, determining the parity of the number of solutions to NP-complete problems is EP-complete providing the reduction from SAT is solution-preserving. The generalized Ladner's theorem [10] tells us that (provided EP  $\neq$  P) 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  $n^{\Omega(\log n)}$  time algorithm of [13]). Tournament isomorphism is in EP since the automorphism group of a tournament has odd order (hence, the number of isomorphisms between two tournaments is either zero or odd).

#### 4. Networks

In Section 2 we classified 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. Subsequently, we showed that this classification has relevance to the computational power of extended Turing machines. In this section we give a further application of the classification in terms of the computational power of (possibly cyclic) networks of two-input Boolean gates.

Cyclic networks are formalized in a similar manner to 'conglomerates' which are a parallel machine model introduced in [7]. Informally, conglomerates consists of synchronous finite state machines communicating via an interconnection network. When the pattern of the interconnections is computable in polynomial space (or equivalently polynomial parallel time), then the resulting class of conglomerates turns out to be as powerful, to within a polynomial, as any parallel machine.

**Definition.** A network over basis  $B$  is a four-tuple  $C = (I, G, f, h)$ , where  $G$  is an infinite set of gates  $G_i$  for all integers  $i \in \mathbb{Z}$  such that each gate in  $G$  realizes a function from the basis  $B$ , and

(1)  $I$  is the finite input alphabet,  $\mathcal{Y} \subseteq I$ ,

(2)  $h : \mathbb{Z} \rightarrow \{1, 2, \dots, |B|\}$  is defined so that  $h(j) = i$  if  $G_j$  realizes the  $i$ th function in  $B$ ,

(3)  $f: \{1, 2, \dots, r\}^* \rightarrow Z \cup \{\text{TRUE}, \text{FALSE}, \phi_1, \phi_2\}$  is the connector, defined similarly to that of conglomerates, where  $r$  is the maximum fan-in of any function in  $B$ , **TRUE** and **FALSE** represent an input being always 1 or always 0 respectively, and  $\phi_1, \phi_2$  represent an input being connected to the corresponding ports of a two-phase clock (see Fig. 2).

A computation of the network over basis  $B$  begins at time 0 with the outputs of gates  $G_1, G_2, \dots, G_{2n-1}$  being set to  $w_1, \dots, w_n$  where  $w_1, \dots, w_n$  is a Boolean representation of the input string over alphabet  $I$ . This Boolean representation of the symbols in  $I \cup \{\emptyset\}$  is such that some fixed number of bits are used to represent each symbol. The outputs of gates  $G_{2n+1}, G_{2n+3}, \dots$  and  $G_{-1}, G_{-3}, \dots$  are initially set to the Boolean representation of an infinite sequence of blanks  $\emptyset$ , and, for all integers  $i$ , the output of gate  $G_{2i}$  will initially be set to the negation of the output of gate  $G_{2i-1}$ . The reason for this input convention will become apparent shortly. Note that the network can detect the end of the input string by checking for the first trailing blank character.



Fig. 2. Two-phase clock.

Each type of gate has an associated delay time—some integral number of time quanta—during which it computes its corresponding function of the values on its inputs, setting its output value equal to the result. The computation proceeds in discrete steps so that an input connected to  $\phi_1$  of the clock will have the value  $i \bmod 2$ , and an input connected to  $\phi_2$  will have the value  $(i+1) \bmod 2$ , during step  $i$  of the computation. Each discrete step of the computation represents some fixed period of time, measured as an integral number of time quanta.

The network over basis  $B$  is said to accept its input  $w$  iff the output of  $G_0$  is ever equal to 1.  $C$  accepts  $w$  within time  $t$  iff the output of  $G_0$  is equal to 1 on or before step  $t$  of the computation.  $C$  accepts a language  $L \subseteq I^*$  in time  $T(n)$  if, for each  $w \in L$ ,  $C$  accepts  $w$  within time  $T(|w|)$ , and, for each  $w \notin L$ ,  $C$  does not accept  $w$ .

Both conglomerates and networks have enormous computational power depending on the complexity of the connection function  $f$  and the function  $h$ . However, it has been shown [7] that if  $f$  and  $h$  are computable in polynomial space (i.e., parallel polynomial time), then the computational power of conglomerates does not exceed that of other parallel computer models such as alternating Turing machines.

So we are interested in studying the relationship between our classification of the two-input Boolean functions and the computational power of networks whose functions are computable in polynomial space.

**Theorem 4.1**

$$\text{NETWORK-TIME}_B(T(n)) \subseteq \text{CONGLOMERATE-TIME}(d \cdot T(n))$$

for some constant  $d$ .

**Proof.** The gates of the network over basis  $B$  can be simulated by finite controls. Each finite control can 'know' which gate from  $B$  it is simulating by leaving appropriate inputs to that finite control unconnected. Also, each finite control will count up to the number of time quanta which represent the delay time of the gate being simulated. Only after that delay time has elapsed will the finite control update its output value. The two-phase clock can be simulated by two finite controls, one representing  $\phi_1$  and the other  $\phi_2$ , which simply count the number of time quanta starting from time 0, the count being modulo the number of quanta which comprise one step of the computation of the network. It is straightforward to check that the connection function of the conglomerate as constructed above can be computed in polynomial space, given that  $f$  and  $h$  of the network can be computed in polynomial space.  $\square$

**Theorem 4.2.** For all complete bases  $B$ ,

$$\text{CONGLOMERATE-TIME}(T(n)) \subseteq \text{NETWORK-TIME}_B(2T(n)).$$

**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 inputs 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  $B$  forming 'flip-flop' circuits, e.g., if  $B = \{\wedge, \vee, \neg\}$ , the flip-flop could be as in Fig. 3. The number of time-quanta comprising one step of the computation should be chosen to be greater than the

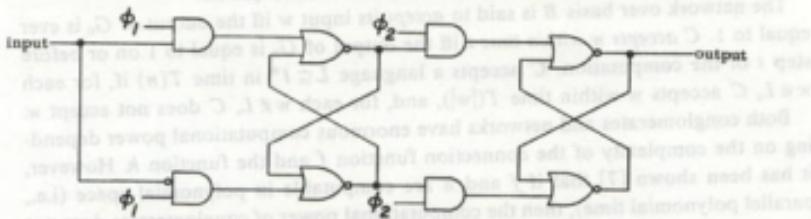


Fig. 3. Standard flip-flop.

longest delay through any of the combinational circuits as constructed above. So, each step of the conglomerate is simulated by two steps of the network. It is straightforward to check that  $f$  and  $h$  of the network can be computed in polynomial space, given that the connection function of the conglomerate can be computed in polynomial space.  $\square$

**Theorem 4.3.** *For all P-complete bases  $B$ ,*

$$\text{CONGLOMERATE-TIME}(T(n)) \leq \text{NETWORK-TIME}_B(2T(n)).$$

**Proof.** Consider the case when  $B = \{\wedge, \vee\}$ . We will perform a similar simulation to that of Theorem 4.2, except that no NOT gates are available for use in the network. The standard memory element shown in Fig. 3 can be replaced by a 'monotone memory' element shown in Fig. 4.

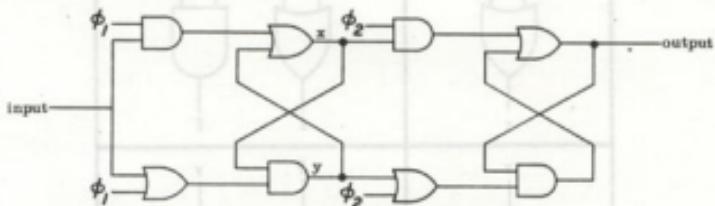


Fig. 4. 'Monotone memory' element.

This monotone memory element operates in a similar fashion to the standard flip-flop. When  $\phi_1 = 0$  and  $\phi_2 = 1$ ,  $x$  and  $y$  retain their previous values, independent of any change in the input value. Furthermore, the value of  $x$  ( $=y$ ) will be copied into the second stage (i.e.,  $\text{output} = x = y$ ). This two stage memory element is used to eliminate 'race conditions' which could otherwise occur in a cyclic network.

The combinational circuits introduced in the proof of Theorem 4.2 may also contain NOT gates, which need to be eliminated in the current simulation. The idea is to use 'double rail logic' [5]. Assuming that for each input  $x$  to a combinational circuit another input  $\bar{x}$  is available which carries its negation  $\neg x$ , each AND, OR, and NOT gate may be replaced by only AND and OR gates as shown in Table 3.

It is clear from Table 3 that for every output  $z$  of the new combinational circuit there will be an additional output  $\bar{z}$  which carries its negation  $\neg z$ . Therefore, if two monotone memory elements are utilized, one for  $z$  and the other for  $\bar{z}$ , then our assumption that the negation of each input to the combinational circuit is available will be met. In addition, any input connected to TRUE or FALSE will have to have its negation connected to FALSE or TRUE, respectively. It remains to ensure that the values on all the wires start correctly at the beginning of the simulation. This is achieved by the input convention which has the negation of each input initially available.

Table 3.

Table 3 shows how to replace each of the three basic gates by a combination of two single rail logic gates. The first row shows how to replace an AND gate by two single rail AND gates. The second row shows how to replace an OR gate by two single rail OR gates. The third row shows how to replace an INVERTER gate by two single rail NOT gates.

original gate	replace by

Thus, the theorem holds when  $B = \{\wedge, \vee\}$ , the complexity of the network increasing by no more than a constant factor. The theorem holds for all other P-complete bases  $B$  using the techniques of Lemma 2.4 and 2.5.  $\square$

**Theorem 4.4.** *If  $B$  is not P-complete, then networks 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 be used to simulate the conglomerate which computes the NAND function  $\text{NAND}(b_1, b_2) = f(b_1, \neg b_1, b_2, \neg b_2)$ . 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  $d$  [14]. Also note that any clock signals coming into a gate in the unrolled circuit can be set to a constant value representing the value of the 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,  $\text{CVP}_{\{\cdot\}} \leq_{\log} \text{CVP}_B$  and so  $B$  is P-complete, contradiction.  $\square$

Loosely speaking we can summarize this section by saying that a particular basis  $B \subseteq B_2$  can be used to build general purpose 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' one. The remaining bases of two-input Boolean functions are not believed to be P-complete (unless  $P = \text{SPACE}(\log^k n)$  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 Boolean networks 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  $\{\vee\}$ ,  $\{\wedge\}$ , the one-input functions, and  $\{\oplus\}$ , corresponding to nondeterministic, 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  $\{\wedge, \vee\}$  is not a powerful computational basis for planar circuits [6]. It would be nice to know more about the class EP. (For example, is it identical to a previously studied class?) What is the relationship between EP, P, NP, and Co-NP? Is there a 'natural' problem which is EP-complete?

## Acknowledgment

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

## References

- [1] A.V. Aho, J.E. Hopcroft and J.D. Ullman, *The Design and Analysis of Computer Algorithms* (Addison-Wesley, Reading, MA, 1974).
- [2] A.K. Chandra, D.C. Kozen and L.J. Stockmeyer, Alternation, *J. ACM* **28**(1) (1981) 114-133.
- [3] S.A. Cook, The complexity of theorem proving procedures, *Proc. 3rd ACM Symp. on Theory of Computing* (1971) 151-148.
- [4] M.R. Garey and D.S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness* (Freeman, San Francisco, CA, 1979).
- [5] L.M. Goldschlager, The monotone and planar circuit value problems are log space complete for P, *SIGACT News* **9**(2) (1977) 25-29.
- [6] L.M. Goldschlager, A space efficient algorithm for the monotone planar circuit value problem, *Inform. Process. Lett.* **10**(1) (1980) 25-27.
- [7] L.M. Goldschlager, A universal interconnection pattern for parallel computers, *J. ACM* **29**(4) (1982) 1073-1086.
- [8] L.M. Goldschlager, A characterization of sets of  $n$ -input gates in terms of their computational power, Tech. Rept. 216, Basser Dept. of Computer Science, Univ. of Sydney, 1983.
- [9] N.D. Jones and W.T. Laaser, Complete problems for deterministic polynomial time, *Theoret. Comput. Sci.* **3** (1977) 105-117.
- [10] T. Kamimura and G. Slutzki, Some results on pseudopolynomial algorithms, Tech. Rept. TR-80-6, Dept. of Computer Science, Univ. of Kansas, 1980.
- [11] R.M. Karp, Reducibility among combinatorial problems, in: J.W. Thatcher, ed., *Complexity of Computer Computations* (Plenum Press, New York, 1972).
- [12] R.E. Ladner, The circuit value problem is log space complete for P, *SIGACT News* **7**(1) (1975) 18-20.
- [13] E.M. Luks, Isomorphism of graphs of bounded valence can be tested in polynomial time, *Proc. 21st Ann. IEEE Symp. on Foundations of Computer Science* (1980) 42-49.
- [14] J.E. Savage, Computational work and time on finite machines, *J. ACM* **19**(4) (1972) 660-674.
- [15] L.G. Valiant, The complexity of enumeration and reliability problems, *SIAM J. Comput.* **8**(3) (1979) 410-421.