Assigning Signal Flow Directions to MOS Transistors

BY

Kuen-Jong Lee, Rajiv Gupta and Melvin A. Breuer

Technical Report No. CENG 90-09

February 1990

Electrical Engineering - Systems Department

University of Southern California

Los Angeles, CA. 90089-0781

Assigning Signal Flow Directions to MOS Transistors[*]

Kuen-Jong Lee, Rajiv Gupta and Melvin A. Breuer

Department of Electrical Engineering-Systems
University of Southern California
Los Angeles, CA 90089-0781

Abstract

Signal flow direction of MOS transistors is used in many CAD tools such as switch-level simulators, timing simulators, electrical rules checkers, testability analysis tools and test generators. This paper presents a new graph theoretic approach for determining signal flow directions.

A MOS circuit is partitioned into a number of channel-connected subcircuits each of which is modeled using an undirected graph (called ST-graph) with two special nodes s (source) and t (target). The direction assignment problem is modeled as the *two paths problem* (TPP) in each ST-graph. Existing algorithms for TPP are either computationally intensive or not easily amenable to implementation. In this paper we presents a sequence of simple and efficient algorithms which, when successively applied to an ST-graph, assign directions to most of the edges (i.e., transistors in the original circuit). Necessary and sufficient conditions under which a unique direction can be assigned to all the transistors in an ST-graph are derived. We prove that if all transistors in the circuit are unidirectional then our algorithms will assign a unique direction to them in linear time. The techniques described in this paper are general and can be applied to other problems such as the maximum flow problem and the reliability modeling problem.

[*]This work was supported by Defense Advanced Research Projects Agency and monitored by the Office of Naval Research under contract no. N00014-87-K-0861.

The views and conclusions contained in this document are those of the authors and should not be interpreted as necessarily representing the official policies, either expressed or implied, of the Defense Advanced Research Projects Agency or the U.S. Government.

Contents

1 Introduction 1

2 Related Research 3

3 Circuit Model and Problem Formulation 5

4 PS-reduction 8

5 Manipulating the reduced graph 13

6 Remaining edges 21

7 Extension to Non-static Circuits 22

8 Limitation 23

9 Applications to other problems 24

10 Conclusion 24

List of Figures

1 A NOR gate with transistor directions 27
2 Transistors PB, PC, PD cannot be assigned direction by rules in [1] 27
3 Transistors A and B are assigned incorrect direction by rules in [1] 28
4 Transistors PC and NC are bidirectional 28
5 Transistor groups of a MOS circuit 29
6 The ST-graph for TG_2 in Figure 5 30
7 The importance of merging input nodes while forming an ST-graph 30
8 S-reduction 31
9 P-reduction 31
10 An ST-graph (a) and its reduced graph (b) 32
11 Proof of Lemma 3, where the S-reduction on e_1 and e_2 does not affect $G_1^{d'}$ and G_2^d 32
12 A static carry lookahead gate and its corresponding ST-graph 33
13 Local articulation points 34
14 (a) Biconnected components and articulation points, and (b) slices 35
15 Unidirectionality of edges connected to articulation points 35
16 (a) Graph structure embedded in every partially reducible ST-graph, and (b) the seven paths used in Lemma 9 36
17 The smallest LSLC containing a LAP with the maximum level 37
18 Proof of Lemma 9 and Theorem 5 37
19 Unidirectional transistor in an AE-cut 38
20 Assigning direction to (u, v) — Case 2 39
21 An apparent "counter" example to Theorem 6 39
22 Non-static logic gates: (a) dynamic logic, (b) clocked CMOS logic, (c) domino logic, and (d) cascade voltage switch logic 40
23 A Manchester carry chain and its ST-graphs 41
24 A dynamic Flip-Flop 41
25 A dynamic RAM cell 42
26 An example showing the effect of circuit semantics on TPP 42

1 Introduction

A MOS transistor is inherently a bidirectional device as signals can propagate either from source to drain, or from drain to source when the transistor is in the on state. The actual signal flow direction is determined by the strengths or drive capabilities of other transistors connected to the source and drain terminals of the transistor. When a transistor is turned on, either the source or the drain terminal can be regarded as the *source*, and the other as the *target* or *destination* of signal flow. The direction of a transistor is defined as the direction from the source to the destination of the signal. Most MOS transistors operate in a unidirectional mode. For example, in the fully complementary CMOS NOR gate shown in Figure 1 each transistor can be assigned a direction as shown in the figure. Nodes u and v are the source and destination of signal flow through transistor PA, respectively.

For many computer-aided design and analysis tools, it has been found that if the direction of signal flow through transistors is known *a priori*, then both the computational effort and the accuracy of results can be considerably improved [1,2,3,4]. Jouppi [1] uses the transistor direction to guide the search for critical paths in a circuit to obtain accurate timing simulation data. He also uses direction information for electrical design rule checking. Cirit [2] uses direction information in calculating the controllabilities and observabilities of signal nets (or nodes). Chen *et al.* [3] suggest using direction information to reduce the search time in switch-level test generation. Transistor directions have also been used to speed up switch-level simulation [4].

In this paper we first consider the problem of assigning directions to transistors in a static MOS circuit. The applications of our method to non-static circuits and to other problems such as maximum flow and reliability analysis are then considered.

The direction assignment problem can be transformed into the *two paths problem* (TPP) in an undirected graph. A general TPP can be stated as follows. Given 4 nodes s_1, t_1, s_2, t_2 in an undirected graph G, determine whether there exist two vertex-disjoint paths in G, one from s_1 to t_1 and the other from s_2 to t_2. If there exists an edge e between t_1 and s_2, then the TPP problem is equivalent to the problem of determining the existence of a simple path from s_1 to t_2 which traverses e from t_1 to s_2.

The transformation of the transistor direction assignment problem to the TPP proceeds as follows. The circuit under consideration is divided into a number of *transistor groups* [5], each of which is modeled as an undirected graph called *ST-graph.* Each ST-graph contains two special nodes, a source node s and a destination (or target) node t which represent the inputs and the outputs of the transistor group, respectively. The question whether the direction u to v in the transistor modeled by the edge (u, v) is feasible depends on the existence of two vertex-disjoint paths from s to u and from v to t.

The existence of simultaneous vertex-disjoint paths in an undirected graph has been discussed in literature [6,7,8]. Two algorithms for solving the TPP have been proposed [6,7]. These algorithms are computationally intensive ($O(ne)$ for each edge or $O(ne^2)$ for all edges, where n and e are the numbers of nodes and edges in the graph, respectively) and/or not easily amendable to implementation. Algorithms for TPP will be discussed in Section 2.

The problem of assigning directions to transistors can be more efficiently solved if some special properties of MOS circuits are taken into account. A rule-based system which investigates these properties has been developed in [1] and extended in [2]. However a rule-based approach also suffers from several drawbacks which will be discussed in Section 2.

In this paper we present a sequence of efficient algorithms which deal with the two path problem for the ST-graphs derived from MOS circuits. Instead of directly solving the general TPP, our algorithms take into account the fact that, in practice, most transistors are designed to be unidirectional. In addition most circuits are constructed from a normal form which results in a series-parallel circuit structure. For example, to implement a function $f = \overline{C_1 + C_2 + \ldots + C_n}$ using NMOS technology, where each $C_i = l_{i_1} l_{i_2} \ldots l_{i_{im}}$ is a product term of input literals, the pull-down network will be constructed by connecting n C_i networks parallel, where each C_i inetwork is formed by serially connected transistors controlled by $l_{i_1}, l_{i_2}, \ldots, l_{i_{im}}$. Our algorithms use the series-parallel nature of circuits to reduce the size of the problem.

Because of the above facts, the algorithms presented in this paper are much more efficient than existing algorithms for the TPP. In fact all transistors in a circuit constructed in a series-parallel manner can be assigned a direction in linear time. If a circuit is not constructed in this manner, our algorithms can still assign direction to most transistors in linear time. Thus the average time complexity is near linear. This is a considerable improvement when compared with existing TPP algorithms for general graphs.

The sequence of algorithms used for direction assignment is described below.

The first algorithm merges transistors connected in series or in parallel into one transistor. This procedure, referred to as *PS-reduction*, runs in linear time and is, in general, able to reduce the size of an ST-graph significantly. We prove that if all transistors in the circuit are unidirectional then PS-reduction can reduce the corresponding ST-graph to one single edge. Interestingly, the converse of the above result is also true. Thus for circuits containing only unidirectional transistors the direction assignment problem can be solved in linear time.

If the reduced graph obtained by applying PS-reduction to the original ST-graph contains more than one edge, further processing is needed. The second algorithm uses a divide and conquer approach to recursively partition the reduced graph into smaller components by identifying a set of special nodes called *local articulation points.* It is shown that all the transistors connected to these special nodes are unidirectional. This step, called the *LAP-find* (*local articulation point find*), takes $O(e_1^2)$ time, where e_1 is the number of edges in the reduced graph.

The third algorithm, called *AE-cut*, is then used for each remaining transistor to identify more unidirectional transistors. AE-cut uses the components identified by LAP-find. The algorithm runs in $O(e_2)$ time, where e_2 is the number of edges of the smallest component containing the edge.

Since there may exist some bidirectional transistors, it is possible that all the above algorithms fail to assign directions to some transistors. Two theorems proved in this paper allow us to make use of the direction information obtained by the above algorithms to assign direction to some specific transistors. These theorems lead to two simple yet powerful procedures which can identify most bidirectional transistors immediately.

For the remaining transistors, the algorithm proposed by [7] can be applied to determine transistor direction. It is viable to use a computationally expensive algorithm (though still polynomial time) at this stage because, having used the algorithms just mentioned, (1) the resulting graph to which the TPP has to be solved is considerably simpler than the original ST-graph, and (2) the TPP has to be solved for only a few unassigned transistors rather than all the transistors in the circuit.

The remainder of this paper is organized as follows. In Section 2 rule-based approaches for direction assignment and existing algorithms for the TPP are described and their limitations discussed. Section 3 consists of a formulation of direction assignment problem and a description of the circuit model. PS-reduction is presented in Sections 4. LAP-find, AE-cut and the procedures that make use of the direction information obtained earlier to assign more directions are described in Sections 5. Section 6 describes the procedure for assigning directions to all transistors which remain unassigned after the above processes. Section 7 discusses the applications of the above procedures to circuits other than static MOS circuits. The limitation of our method is discussed in Section 8. The application to other problems is given in Section 9. Finally, concluding remarks are presented in Sections 10.

2 Related Research

In this section we discuss rule-based systems for assigning directions to transistors and the existing algorithms for TPP.

Rule-based Systems

In [1] a rule-based system is used to solve the direction problem. Eight "safe" rules and five "unsafe" rules are developed. This system, though empirically shown to have good transistor coverage, has the following problems. First, the applications of rules rely on pattern matching techniques which are often time-consuming. Second, only local effects are considered and thus some transistor directions which require global circuit information cannot be assigned. For example transistors *PB*, *PC* and *PD* in Figure 2 can be assigned directions towards the output node *O*, but none of the rules proposed in [1] can assign these directions. Third, since the rules are verified through informal arguments rather than formal techniques, some "safe" rules may turn out to be unsafe in some special cases. For example Figure 3(a) shows a six transistors memory cell whose equivalent circuit diagram is given in Figure 3(b). Transistors A and B are both bidirectional but the system developed in [1] will identify them as unidirectional because, according to "safe" rule 6, all transistors fed by the output of an inverter are considered as unidirectional.

In [2] the direction rules of [1] are argumented by identifying some special patterns such as transmission gate adders and Manchester carry chains. In addition Kirchoff's current rule is repeatedly applied to propagate the direction information throughout the circuit. However this method is based on the assumption that all transistors are unidirectional, which is not always true. One simple counter example is shown in Figure 4 where transistors PC and NC are actually bidirectional.

There are other concerns that need to be addressed while using a rule-based system, especially when the number of rules is large. A rule-based system can be thought of as a term rewriting system in which each rule or production replaces a part of the input term (a circuit in the present case) with another, hopefully, simpler and more specific term. Thus, each application of a rule is equivalent to a reduction.

For any rule-based system, one would like to guarantee the following two properties. (1) For any input term, there is no infinite sequence of reductions (*uniform termination* or *noetherian*), and (2) for any distinct terms a, b and c, if b and c are reachable from a, then there is another term d which is reachable from both b and c (*unique termination* or *confluence*). In general, establishing the noetherian and confluence properties for any rule-based system is difficult. In most system only informal arguments are given to support them. However, if these properties are not established, a rule-based system can at best be regarded as heuristic rather than algorithmic.

For the algorithmic reductions described in this paper, we prove both the unique and uniform termination properties.

Two-paths problem

As mentioned before, two algorithms for the TPP have been proposed in the literature [6,7]. In [6] the emphasis is on the derivation of the worst case complexity. A complex analysis is given to show that TPP can be solved in $O(ne)$ time for each edge in the graph, where n and e are the numbers of nodes and edges, respectively. This analysis involves six successive reductions each of which has the form "One may assume the graph has a property X, otherwise the problem is either already known to be solvable or can be transformed into a solvable one."

With these reductions the graphs under consideration are divided into a large number of cases, each of which can be solved using a specialized algorithm. Thus, a large number of algorithms have to be implemented in order for this result to be useful. Also, because of the large number of transformations involved, the effect of multiplicative constants, which are generally ignored in evaluating worst-case complexity, may make the average-case complexity unacceptable. In addition any implementation of this algorithm will entail an $O(ne)$ computation for each transistor, requiring a total time of $O(ne^2)$ for an ST-graph with n nodes and e edges. This time complexity may be unacceptable for large circuits.

Another algorithm for the TPP has been proposed in [7]. This algorithm requires a procedure that repeatedly examines whether there exists a subgraph $G' = (V', E')$ of $G = (V, E)$ such that (1) G' is connected, (2) V' does not contain s_1, t_1, s_2, t_2, and (3) $|\delta(V')| \leq 3$, where $\delta(V') = \{x | (x, v) \in E, x \notin V', v \in V'\}$. In addition this algorithm requires a modified planarity check on a transformed graph in order to determine the existence of two vertex-disjoint paths. Thus this algorithm may also become computationally intensive if it is used to determine the direction of all the edges in the ST-graph. We will use this algorithm to determine the direction of a subset of transistors in a reduced ST-graph.

3 Circuit Model and Problem Formulation

For ease of exposition, we initially assume that the circuits under consideration are static, i.e., the operation of circuits does not rely on charge retension or charge sharing effects. Thus, whenever a node affects the circuit operation, it must be connected to a power source (*VDD*, *GND*, or a primary input) through a path consisting of conducting transistors. The extensions of our direction assignment algorithms to non-static circuits will be discussed in Section 9.

A MOS circuit is modeled as an undirected graph, called the *circuit graph*, whose edges correspond to the transistor channels and whose vertices (or nodes) correspond to the nodes in the circuit. By disconnecting the gate terminals of all transistors and replicating the *VDD* and *GND* nodes such that each edge connected to *VDD* or *GND* has its own copy, the circuit graph is partitioned into a number of connected components called *transistor groups*(TGs) [5]. In effect each TG is a "channel-connected" component. Figure 5(a) and (b) show a CMOS circuit and its four TGs (node B forms a TG by itself).

For each TG, the sets of input and output nodes are defined as follows. Input nodes are those nodes which act as the source of signal flow to the corresponding TG. Possible input nodes to a TG are *VDD*, *GND* and all primary inputs. Output nodes are those nodes which act as the destination of signal flow. Possible output nodes are primary outputs and all nodes which control the gate terminals of non-depletion type transistors. In Figure 5, A and O_1 are the input node and output node of TG_1, respectively. Both *VDD* and *GND* are input nodes to TG_2 and TG_3. O_2 and O_3 are output nodes of TG_2, and O_4 is the output node of TG_3.

ST-graph

For each transistor group $TG_i = (N_i, T_i)$, where N_i and T_i are the sets of nodes and transistors (or edges), respectively, we define an ST-graph $ST_i = (V_i, E_i)$ as follows.

Definition 1 [*ST-graph*] *Let I and O be the sets of input nodes and output nodes of a transistor graph $TG_i = (N_i, T_i)$, respectively. An ST-graph $ST_i = (V_i, E_i)$ corresponding to TG_i is defined as*

$$\begin{aligned} V_i &= (N_i - I) \cup \{s\} \cup \{t\}, \textit{and} \\ E_i &= (T_i - \{(u,v) | u \textit{ or } v \in I\}) \cup \{(s,v) | \exists v \notin I, s_i \in I, \textit{such that} (s_i, v) \in T_i\} \\ &\quad \cup \{(u,t) | u \in O\}. \end{aligned}$$

In other words, we remove all transistors whose source and drain nodes are both input nodes, merge all input nodes into a new node s (source), and connect each output node to another new node t (target or destination) by a new edge. Figure 6 shows the ST-graph of TG_2 of Figure 5 which has two input nodes and two output nodes. Note the difference between the construction of s and t. The reasons for this difference will be given after the definition of an edge direction has been introduced.

In practice, not every undirected, connected graph can be an ST-graph. We shall assume that each edge in an ST-graph is on at least one simple path (a path with no loop) from s to t. This assumption will always be true since a transistor which is not on any simple path from an input node to an output node is clearly redundant and can be removed from the circuit. Also we assume that the degree of each node except s and t is bounded by some constant C. This is also generally true for most circuits because of underlying technological considerations.

Edge direction

Signal flow direction is formally defined as follows.

Definition 2 [*Direction*] *An edge (u,v) in an ST-graph can be assigned a direction from u to v if and only if there exists two simple vertex-disjoint paths, one between s and u and the other between v and t. (u,v) is bidirectional if and only if both directions from u to v and from v to u can be assigned. If only one direction can be assigned, then the edge is unidirectional.*

This definition captures the essential nature of signal flow directions of transistors in a static circuit. For a static circuit, each output node of a transistor group should be driven by an input node through a sequence of transistors. Each transistor can have a specific signal flow direction only if there exists a path from an input node to an output node which traverses the transistor in the specified direction. For an edge (u,v) in an ST-graph to have a direction from u to v, it is essential that there exist two vertex-disjoint paths, one from s to u and the other from v to t.

We now give the reasons why nodes s and t are constructed differently in an ST-graph. For any static circuit, during steady state operation an input node acts as an "active" node while an output node acts as a "passive" node (driven by an input node). We remove any transistor connecting two active nodes because such a transistor is either the result of a design error or is redundant and should be removed. After removing these transistors, all input nodes are merged into one node s because any transistor connected to an input node must be unidirectional (outward from the input node). For the output nodes, the situation is different. It is not uncommon for two output (or passive) nodes to be connected by a transistor. To preserve these transistors while retaining only one destination, an additional destination node t must be introduced into the ST-graph.

The correct construction of an ST-graph is critical to the direction assignment. Figure 7 shows how erroneous direction may be assigned if all input nodes were not merged into one node. Figure 7(a) is a TG with two input nodes I_1 and I_2 and one output node O. Clearly directions I_1 to x and I_2 to x should be assigned. Figure 7(b) shows the situation when I_1 and I_2 are not merged into one node; instead a new node s is added. This results in edge (I_1, x) being assigned as bidirectional because of the existence of vertex-disjoint paths s to I_1 and x to t, and s to x and I_1 to t. A similar argument holds for the edge (I_2, x). Figure 7(c) is the correct ST-graph in which all edges are assigned the correct direction.

Notation

The following symbols and notations are used throughout this paper.

(G, s, t): An ST-graph G with source s and destination t. Sometimes only G is used if no confusion is possible because of the context.

(u, v): An edge of an ST-graph with u and v as its end nodes.

$u \rightarrow v$ or $v \leftarrow u$: (u, v) can be assigned a direction from u to v; the other direction (from v to u) is unknown.

$u \Rightarrow v$ or $v \Leftarrow u$: (u, v) can be assigned a direction from u to v but the other direction is impossible.

$u \leftrightarrow v$ or $v \leftrightarrow u$: (u, v) is bidirectional.

$x \longleftrightarrow y$: There exists a simple path between nodes x and y. $x \longleftrightarrow x$ is always true.

$x \overset{p}{\longleftrightarrow} y$: These exists a simple path p between nodes x and y. The length of p, denoted by $|p|$, is the number of transistors on p.

$(x \longleftrightarrow u, v \longleftrightarrow y)$: There exist two vertex-disjoint paths, one between x and u and the other between v and y.

4 PS-reduction

This section describes the PS-reduction procedure which, in general, reduces the problem size significantly. PS-reduction consists of two basic operations: P-reduction and S-reduction. We shall prove that the resulting graphs obtained from an ST-graph through different PS-reductions are isomorphic. A linear time algorithm for PS-reduction is then described. The relationship between the edge directions in the original graph and the "reduced" graph obtained after PS-reduction are established. In Section 5 we shall prove a theorem which shows that an ST-graph can be reduced to a single edge if and only if all edges in the ST-graph are unidirectional.

PS-reduction is formally defined as follows.

Definition 3 [*S-reduction*] *A node v in an ST-graph (G, s, t) is S-reducible if $v \notin \{s, t\}$ and $deg(v) = 2$. Let the two neighbors of v be u and x. The S-reduction on v (or on (u, v) and (v, x)) in G is the operation of replacing (u, v) and (v, x) with one new edge (u, x), and removing v from G (see Figure 8).*

Definition 4 [*P-reduction*] *If two edges e_1 and e_2 of an ST-graph (G, s, t) have the same two end nodes u and v, then these two edges are P-reducible in G; the P-reduction on e_1 and e_2 in G is the operation of replacing these two edges with one new edge e_3 which has the same two end nodes (see Figure 9).*

Definition 5 [*PS-reduction, reduced graphs*] *A PS-reduction on an ST-graph G is a sequence of P-reductions and S-reductions on G such that no further P-reduction or S-reduction can be applied to the resulting graph. The resulting graph is called a reduced ST-graph of G.*

Since each P- or S-reduction reduces the number of edges in the graph by one, uniform termination is guaranteed. Next we shall prove that any two different PS-reductions will reduce an ST-graph into isomorphic graphs, which establishes the desired unique termination property. To prove this, the following lemmas are needed.

Lemma 1 *Let G' be the reduced graph obtained from G via a PS-reduction PS. For each $e_i = (u_i, v_i)$ in G', there exists a corresponding subgraph G_i in G such that G_i is itself an ST-graph with v_i and u_i as its source and destination nodes. In addition, all G_i's are mutually edge-disjoint.*

Proof: Each edge $e_i = (u_i, v_i)$ in G' is an ST-graph by definition. Considering the reverse procedure of PS which recovers G from G'. To undo one S-reduction that created an edge $e = (u, v)$, we add a node x to e such that x splits e into two edges (u, x) and (x, v); to undo a P-reduction that created (u, v), we simply add one more parallel edge (u, v). Clearly, these two operations simply transform an ST-graph into another one with the same source and destination nodes. Thus, the lemma follows by induction. □

The relation between edges in G' and in G can be illustrated by Figure 10. The next lemma uses the above result.

Lemma 2 *Let G' be the reduced graph of G, and v be any node of G' (also of G). Then no S-reduction in* any *PS-reduction on G can be applied to v.*

Proof: If v is s or t then it cannot be S-reduced by definition. Thus we may assume $v \neq s$ and $v \neq t$. Since v cannot be further reduced in G', $deg(v) \geq 3$ (see Figure 10). There must exist three distinct edges in G', say $e_i = (v, v_i)$, $e_j = (v, v_j)$ and $e_k = (v, v_k)$, such that they all have v as one end node (Figure 10(b)). From the previous lemma, e_i, e_j and e_k in G' correspond to three edge-disjoint subgraphs G_i, G_j and G_k in G. For v to become S-reducible, all the edges in one of these subgraphs must be completely reduced (removed) to make $deg(v) = 2$. Thus one of v_i, v_j and v_k must be S-reduced. It can be proved by induction on the total number of P- and S-reductions that none of v_i, v_j or v_k can be S-reduced by any sequence of P- or S-reductions. □

Lemma 3 *If there exists a PS-reduction which reduces an ST-graph G to a single edge, then any PS-reduction on G will reduce it to a single edge.*

Proof: We prove this lemma by induction on $|E|$, the number of edges in the original ST-graph. The lemma obviously holds when $|E| = 1$. Assume it also holds when $|E| = n > 1$. Let PS_1 be a PS-reduction which reduces a graph with $|E| = n + 1$ into an edge (u, v). PS_1 must contain n P-reductions and S-reductions in total. Consider the two edges e_1 and e_2 in the graph just before the last reduction of PS_1. The original graph must contain two subgraphs G_1 and G_2 (either in "parallel" or in "series") such that G_1 and G_2 can be reduced to e_1 and e_2, respectively. Obviously all reductions of PS_1 involving edges in G_1 are independent of those involving edges in G_2. By the induction hypothesis any PS-reduction on G_1 (or G_2) reduces it to one edge. Thus any PS-reduction on G that first reduces G_1 and G_2 individually to one edge also reduces G to a single edge.

Now consider a PS-reduction which contains one or more reductions that involve one edge from G_1^d and another from G_2^d, where G_1^d and G_2^d are obtained from G_1 and G_2. Let R be the first such reduction in this PS-reduction. R has the effect of removing one edge from G_1^d and no effect on G_2^d or vice versa. Without loss of generality assume R removes an edge e_1 from G_1^d. If R is an S-reduction then Figure 11(a) and (b) show this situation right before and after R is executed. It is clear that R cannot affect the applicability of any reduction on any edge in $G_1^{d'}$ (the graph obtained by removing e_1 from G_1^d) and G_2^d. Thus by induction, G_2^d and $G_1^{d'}$ (which both have less edges than $|E|$) can both be reduced to one edge and hence the lemma holds. Similar arguments can show that the lemma also holds when R is a P-reduction. □

Theorem 1 *All reduced graphs of an ST-graph are isomorphic.*

Proof: Let G' be a reduced graph obtained from an ST-graph G, and $e_i, i = 1, 2, \dots$ be the edges of G' which are reduced from the subgraphs $G_i, i = 1, 2, \dots$ of G, respectively. By Lemma 2, no node in G' can be reduced. In addition, every node in G' is also in G. By Lemma 3 all edges in a G_i can be reduced to one edge by any PS-reduction on G_i. Since no node in G' can be reduced, no PS-reduction on G can involve two edges from two distinct G_is. Thus an arbitrary PS-reduction on G is equivalent to some permutation of individual reductions on G_is. Since reductions on G_i are independent of those on G_j, $i \neq j$, an arbitrary PS-reduction will independently reduce each G_i to a single edge. Thus all reduced graphs are isomorphic to G'. □

By virtue of the above theorem, one simplistic way to obtain the reduced graph of an ST-graph G is by iteratively checking whether any nodes or edges are S- or P-reducible. If either reduction can be applied, it is executed and the process continues with the new graph. This is done until no further reduction can be applied. This method, which is indeed followed by rule-based systems, is not efficient as it requires several passes over the graph. In the following we present a linear time algorithm which accomplishes the task in one pass over the graph.

Algorithm 1: (PS-reduction)

```
Procedure PS_reduction($G, V, E, s, t$)
{  1)  $V' = V$; $U'$ = List of nodes in $V - \{s, t\}$; $E' = E$;
   2)  Mark all nodes in $U'$ as unreduced
   3)  while ( $U' \neq null$) do
   4)  { remove up to the first unreduced node $v$ from $U'$, if no such $v$ exists, go to (7)
   5)    else { if $deg(v) = 2$ then S_reduction($v$)
   6)           else for each neighbor $u$ of $v$ do
                  if Parallel($u, v, p\_list$) then P_reduction($u, v, p\_list$);}}
   7)  Remove from $E'$ all but one edge whose two end nodes are $s$ and $t$.
}
Procedure S_reduction($v$)
{  8)  Mark $v$ as reduced; Let $u, x$ be such that $(u, v), (v, x) \in E'$
   9)  $E' = (E' - (u, v) - (v, x)) \cup (u, x)$; $V' = V' - \{v\}$;
   10) if Parallel($u, x, p\_list$) then P_reduction($u, x, p\_list$)
   11) else if ($u \neq s$ and $u \neq t$ and $deg(u) = 2$) S_reduction($u$)
   12) else if ($x \neq s$ and $x \neq t$ and $deg(x) = 2$) S_reduction($x$)
}
Procedure P_reduction($u, v, p\_list$)
{  13) Let $p\_list = \{e_1, e_2, \dots, e_k\}$
   14) $E' = E' - e_2 - e_3 - \dots - e_k$
   15) $deg(u) = deg(u) - k + 1$; $deg(v) = deg(v) - k + 1$
   16) if ($u \neq s$ and $u \neq t$ and $deg(u) = 2$) then S_reduction($u$)
```

```
    17) else if ($v \neq s$ and $v \neq t$ and $deg(v) = 2$) then S_reduction($v$)
}
Function Parallel($u, v, p\_list$): Boolean
{   18) If ($u = s$ and $v = t$) or ($u = t$ and $v = s$) then return(false)
    19) else
    20) {   if ($u \neq s$ and $u \neq t$) then
    21)         for each edge $e$ connected to $u$, if the other end
    22)         node of $e$ is $v$ then put this edge in $p\_list$
    23)     else for each edge $e$ connected to $v$, if the other end
    24)         node of $e$ is $u$ then put this edge in $p\_list$
    25)     return($|p\_list| > 1$) }
}
```

In the above algorithm, V and E are respectively the sets of nodes and edges in the original ST-graph. U' is a list of nodes which need further processing. V' and E' are the temporary sets of nodes and edges during processing. At the end of PS-reduction they represent the final reduced graph.

The algorithm is *event driven*, with P- and S-reductions driving each other as long as possible. Procedure PS_reduction examines each node in list U' to initiate a P- or S-reduction. Whenever a node is examined it is removed from U'. Due to the event driven nature of the algorithm, a node may have been marked as reduced when it is examined. In this case the node is simply removed. Thus it is guaranteed that each node is examined at most once.

An S-reduction can be performed on a node v if v is not s or t and v has two neighbors. If v is S-reducible, procedure S-reduction marks v as *reduced*, removes v from V' and replaces the two edges connected to v with an edge which connects the two neighbors of v. After reducing the node, procedure S-reduction initiates another P- or S-reduction if possible.

Parallel(u, v, p_list) is a function which determines whether a P-reduction should be applied to the edges connecting u and v. Because the degrees of s and t are not restricted, all edges which connect both s and t are not examined until the final step of PS-reduction. If only one of u and v is s or t, then only the edges connected to the non-s (or non-t) node are examined. These two steps are necessary to guarantee the linearity of the whole PS-reduction procedure. Parallel(u, v, p_list) returns a true value only if $\{u, v\} \neq \{s, t\}$ and the number of edges whose two end nodes are u and v is greater than 1. p_list is used to stored these edges.

Procedure P-reduction(u, v, p_list) removes all but one edge in p_list and reduces the degrees of u and v appropriately. It also initiates another S-reduction when possible.

The correctness and linearity of this algorithm are proved next.

Theorem 2 *The reduced graph of any ST-graph $G = (V, E)$ can be obtained using Algo-*

rithm 1 in $O(|E|)$ time.

Proof of correctness: We prove the correctness of the algorithm by considering the possible sources of P-reductions and S-reductions. Two edges can be P-reducible due to two reasons: (1) they are P-reducible in the original graph, or (2) one or both of these two edges are created by an S-reduction. Note that an edge created in a P-reduction cannot be involved in another P-reduction immediately since all edges in parallel with this edge have just been removed. Both categories of P-reductions can be processed either in procedure PS_reduction or in procedure S_reduction. Note that we do not have to explicitly examine nodes s and t in procedure PS_reduction since any P-reduction involving either s or t, but not both, will be processed when the neighbors of s and t are examined, and the P-reduction involving both s and t is processed after all other reductions have been executed. A node can undergo an S-reduction due to three reasons: (1) It is S-reducible in the original graph, (2) it becomes S-reducible due to a P-reduction, and (3) it is S-reducible and an S-reduction was just executed on one of its neighbors. All these cases are taken care of by Algorithm 1.

Proof of linearity: Consider procedure PS_reduction first. The *while* loop can be executed at most $|V| - 2$ times since each node is examined at most once. Line 4 contributes at most $|V| - 2$ time steps for the *while* loop. If the condition of line 5 is *true*, then an S-reduction is executed. In each iteration of the while loop, line 6 is checked a constant number of times since we assume that the degree of each node (except s and t) is bounded by a constant. Line 7 contributes at most $O(|E|)$ time steps. Now consider the total time taken by procedures S_ and P_reduction. Since each type of reduction eliminates one edge, there are at most $|E| - 1$ S- and P-reductions. Hence procedures S_ and P_reduction can be executed at most $|E| - 1$ times. Since each of these procedures takes constant time, the total time taken by them is at most $O(|E|)$. □

Lemma 4 *If G' is the ST-graph obtained by executing an S-reduction on v in (G, s, t), and u and x are the two neighbors of v in G, then $u \rightarrow x$ in G' if and only if $u \rightarrow v$ and $v \rightarrow x$ in G, and $u \leftarrow x$ in G' if and only if $u \leftarrow v$ and $v \leftarrow x$ in G.*

Proof: Refer to Figure 8. It is obvious that the following three conditions are equivalent (s and t may be interchanged).

1. $(s \longleftrightarrow u,\ v \longleftrightarrow t)$ in G.
2. $(s \longleftrightarrow v,\ x \longleftrightarrow t)$ in G.
3. $(s \longleftrightarrow u,\ x \longleftrightarrow t)$ in G'.

Thus by Definition 2 the lemma follows. □

Lemma 5 *If G' is the ST-graph obtained after a P-reduction on e_1 and e_2 in (G, s, t), and e_3 is the new edge, then e_1 and e_2 in G have the same direction as e_3 has in G'.*

Proof: Refer to Figure 9. Since the following two conditions are equivalent (u and v are the two end nodes of e_1, e_2 and e_3; s and t may be interchanged), the lemma follows directly from Definition 2.

1. $(s \longleftrightarrow u, v \longleftrightarrow t)$ in G.
2. $(s \longleftrightarrow u, v \longleftrightarrow t)$ in G' □

Theorem 3 *Let G' be the graph obtained by executing a PS-reduction on an ST-graph G. If a subgraph of G, say G_1, is reduced to an edge e in G' by this PS-reduction, then all edges in G_1 are unidirectional in G if and only if e is unidirectional in G'.*

Proof: By induction based on Lemmas 4 and 5. □

After PS-reduction, we may assign unidirections to all edges which are connected to s or t in the reduced graph. Then by applying Theorem 3 we can determine the directions of all edges (in G) which have been removed during PS-reduction. The "recovery" of these edges can easily be achieved by maintaining a first-in-last-out stack at the time of PS-reduction.

Example: Figure 12 shows a static carry lookahead gate [9] and its corresponding ST-graphs. It is easy to see that this ST-graph can be reduced to a single edge. Thus each transistor in this circuit is unidirectional. □

5 Manipulating the reduced graph

If an ST-graph cannot be reduced to one edge by PS-reduction, further processing is needed in order to assign directions to all edges. This section describes a sequence of procedures for manipulating the reduced graphs.

Local articulation points

Consider the reduced ST-graph shown in Figure 13(a). Only the edges (and those edges in the original ST-graph which have been reduced to these edges by PS-reduction) which are connected to s or t (i.e., edges 1, 16, 28, 32, 33) can be assigned a unique direction. Many other unidirectional edges exist in this graph. In fact all edges connected to a node with double circles are unidirectional. A systematic method to find all these unidirectional edges is given next. We first define articulation points and some related terms in an ST-graph.

Definition 6 [*Articulation point* (AP)*, extended articulation point* (EAP) *and consecutive EAPs*] *An AP of an ST-graph is a node whose removal disconnects s and t. An EAP of an ST-graph is either s, t or an AP of this graph. Two EAPs are consecutive if every path between these two nodes contains no other EAP.*

Using a depth-first search algorithm [10] starting with s, all APs in an ST-graph can be identified in $O(|E|)$ time. Since the removal of any AP disconnects s and t, all paths from s to t must pass through all APs. Thus the depth-first search algorithm identifies all APs in the order shown in Figure 14, where the closer an AP is to t, the earlier it is identified. Let $EAP_0 = t$, and $EAP_i = AP_i, i = 1, \ldots, n-1$ and $EAP_n = s$. Then each pair of EAP_i and $EAP_{i+1}, i = 0, \ldots, n-1$ are consecutive.

Definition 7 [*Biconnected component* (BCC)] *A BCC of an ST-graph G is a connected subgraph G' of G such that G' contains at least one edge, and for any two distinct edges e_1, e_2 in G, if e_1 is in G', then e_2 is also in G' if and only if there exists a loop in G which contains both e_1 and e_2.*

By definition, each pair of consecutive EAPs defines a BCC and each BCC is by itself an ST-graph with the two EAPs as its source and destination. We shall say that a BCC is *induced* by two consecutive EAPs if the BCC contains these two nodes. For example in Figure 14, each G'_i is induced by EAP_i and $EAP_{i+1}, i = 0, \ldots, n-1$. One can think of articulation points as defining "vertical" partitions of an ST-graph. We next define *slices* which partition an ST-graph "horizontally".

Definition 8 [*Slice* (SLC)] *A SLC of an ST-graph (G, s, t) is a connected subgraph G'' of G such that G'' contains at least one edge, and for any two distinct edges e_1, e_2 in G, if e_1 is in G'', then e_2 is also in G'' if and only if there exists a path in G which contains both e_1 and e_2, but not s or t except as end nodes.*

Like a BCC, each SLC of an ST-graph (G, s, t) is itself an ST-graph with s and t as source and destination, respectively. An ST-graph can either be "vertically" or "horizontally" divided, but not both. Since each BCC is an ST-graph, it may be horizontally divided into several SLCs. Similarly each SLC may be vertically divided into several BCCs. Each of the partitions obtained in this recursive partitioning is very similar to a BCC or a SLC, though at a local level. The following recursive definitions formalize these structures.

Definition 9 [*Local biconnected component* ($LBCC$)] *A LBCC of an ST-graph G is a BCC of G or is a BCC of a LSLC* (*defined below*) *of G.*

Definition 10 [*Local slice* ($LSLC$)] *A $LSLC$ of an ST-graph G is a SLC of G or is a SLC of a $LBCC$ of G.*

Definition 11 [*Local articulation point* ((LAP))] *A LAP of an ST-graph G is a source or destination node of a $LBCC$ of G.*

Definition 12 [*Indivisible ST-graph*] *An ST-graph is indivisible if it contains only one $LSLC$.*

Figure 13 illustrates the above definitions. Figure 13(a) shows an ST-graph G with an articulation point D. D divides G into two ST-graphs (or BCCs) G'_1 and G'_2 as shown in Figure 13(b). Node D becomes both the destination node of G'_1 and the source node (denoted as D') of G'_2. G'_1 can be further divided into three SLCs G''_{11}, G''_{12}, and G''_{13} as shown in Figure 13(c). While G''_{12} is indivisible, G''_{11} and G''_{13} both can be divided into 4 BCCs as shown in 13(d). Each of these BCCs are indivisible. All the nodes that are denoted by double circles are LAPs of G.

The following three lemmas are used to prove Theorem 4, which guarantees that each edge connected to a LAP is unidirectional.

Lemma 6 *All edges connected to an EAP of an ST-graph (G, s, t) are unidirectional.*

Proof: Edges connected to s and t are clearly unidirectional with edge directions away from s and towards t. Thus only APs need be considered. Let u be an AP of G as shown in Figure 15. u divides G into two parts, one contains s and the other contains t. Without loss of generality consider an edge (u, v) in the part containing t. Obviously there exists two disjoint paths, one between s and u and the other between v and t. Thus $u \rightarrow v$ can be assigned. However any two paths p_1 and p_2 such that $s \xleftrightarrow{p_1} v$ and $u \xleftrightarrow{p_2} t$ will not be vertex-disjoint since both of them must contain u. Hence $u \Rightarrow v$ is established. □

Lemma 7 *Let $EAP_i, i = 0, \ldots, n$ be the EAPs of an ST-graph (G, s, t) and G'_i be the BCC induced by EAP_i and EAP_{i+1}. For any edge (u, v) in G'_i, the direction of (u, v) in (G'_i, EAP_{i+1}, EAP_i) is the same as the direction of (u, v) in (G, s, t).*

Proof: Refer to Figure 14(a). Clearly $(s \longleftrightarrow u, v \longleftrightarrow t)$ in (G, s, t) is equivalent to $(EAP_{i+1} \longleftrightarrow u, v \longleftrightarrow EAP_i)$ in (G'_i, EAP_{i+1}, EAP_i) since any path from s to u (v to t) can be divided into two paths, $s \longleftrightarrow EAP_{i+1}$ and $EAP_{i+1} \longleftrightarrow u$ ($v \longleftrightarrow EAP_i$ and $EAP_i \longleftrightarrow t$). The path $s \longleftrightarrow EAP_{i+1}$ ($EAP_i \longleftrightarrow t$) always exists and is independent of any path from EAP_{i+1} to u (from v to EAP_i). Similarly $(s \longleftrightarrow v, u \longleftrightarrow t)$ in (G, s, t) is equivalent to $(EAP_{i+1} \longleftrightarrow v, u \longleftrightarrow EAP_i)$ in (G'_i, EAP_{i+1}, EAP_i). □

Lemma 8 *Let $G''_1, G''_2, \ldots, G''_k$ be the SLCs of an ST-graph (G, s, t), and (u, v) be in G''_j. The direction of (u, v) in (G, s, t) is the same as the direction of (u, v) in (G''_j, s, t).*

Proof: Refer to Figure 14(b). This lemma holds since $(s \longleftrightarrow u, v \longleftrightarrow t)$ in (G, s, t) is equivalent to $(s \longleftrightarrow u, v \longleftrightarrow t)$ in (G''_j, s, t), and $(s \longleftrightarrow v, u \longleftrightarrow t)$ in (G, s, t) is equivalent to $(s \longleftrightarrow v, u \longleftrightarrow t)$ in (G''_j, s, t). □

Theorem 4 *Each edge connected to a local articulation point of an ST-graph is unidirectional.*

Proof: By induction based on Lemmas 6, 7 and 8. □

Theorem 4 establishes the importance of LAPs. The following algorithm finds all the LAPs and assigns direction to the edges connected to them.

Algorithm 2 (Find all LAPs and assign direction to edges connected to LAPs):

```
Procedure LAP_find($G, s, t$)
{  $EAP_0 = t$
   Start with $s$, do depth-first search on $G$ to find the sequence of
        extended articulation points $EAP_1, EAP_2, \ldots, EAP_{n-1}, EAP_n (= s)$.
   For each BCC $G'_i$ induced by $EAP_{i+1}$ and $EAP_i$, $i = 0, 1, \ldots, n-1$, do
   {  Split $G'_i$ into $G''_1, G''_2, \ldots, G''_m$ where each $G''_j = (V''_j, E''_j)$ is a SLC of $G'_i$
      If $m = 1$ then assign directions to all edges connected to $EAP_i$ and $EAP_{i+1}$ in $G'_i$,
      else for each $G''_j$ do
           if $|E''_j| = 1$ then assign the direction $EAP_{i+1} \Rightarrow EAP_i$ to this edge,
           else do LAP_find($G''_j, EAP_{i+1}, EAP_i$) }
}
```

Algorithm 2 recursively divides an ST-graph into smaller and smaller ST-graphs until no further division is possible. All the EAPs of G are first found using a depth-first search algorithm [10]. Each BCC induced by two consecutive EAPs is then split into slices. If only one slice exists, then the edges connected to the EAPs are assigned an appropriate direction. Otherwise for each trivial slice (a slice containing only one edge) a direction is assigned, and for each non-trivial slice LAP_find is invoked again. This procedure continues until all local slices become indivisible.

The exact complexity of Algorithm 2 is clearly problem dependent. Here we shall only consider the worst case complexity. Let the *level* of a LAP v in an ST-graph $G = (V, E)$, denoted as $l(v)$, be the minimum number of recursive calls to the procedure LAP_find that are required to mark v as a LAP. The LAP level of G, denoted by $L(G)$, is defined as

$\max_{v \in V} l(v)$. For example in Figure 13, $l(s) = l(t) = 0$, $l(D) = 1$, $l(A) = l(B) = l(C) = 2$ and L(G)=2. The complexity of Algorithm 2 is bounded by $O(|E| \cdot L(G))$. In the worst case $L(G) = O(|E|)$. Thus the worst case complexity of Algorithm 2 is $O(|E|^2)$. Note, however, that Algorithm 2 is performed on a reduced graph obtained after PS-reduction. In general a reduced graph will have much fewer edges than the original ST-graphs. Thus the expected complexity will be much less than that obtained by solving the TPP problem directly.

PS-reduction in conjunction with the LAP-find algorithm provides a fairly useful direction assignment procedure. Its utility is enhanced by the fact that most transistors operate in a unidirectional mode. In particular we shall prove in Theorem 5 that all the edges in an ST-graph are unidirectional if and only if PS-reduction can reduce the graph into a single edge and thereby assign direction to all edges. First we define *completely reducible ST-graphs* and give a lemma to show an important property of a *partially reducible graph.*

Definition 13 [*Completely reducible ST-graph*] *An ST-graph is completely reducible if and only if its reduced graph contains only one edge, otherwise it is a partially reducible graph.*

Lemma 9 *An ST-graph is not completely reducible if and only if the graph shown in Figure 16(a) is embedded in it such that there exist two vertex disjoint paths* $s \longleftrightarrow A$ *and* $D \longleftrightarrow t$*; i.e., an ST-graph is not completely reducible if and only if it contains 4 distinct nodes* A, B, C, D *(*A *may be* s *and* D *may be* t*) with* $B, C \notin \{s, t\}, A \neq t, D \neq s$*, and 7 paths* $s \longleftrightarrow A$, $A \longleftrightarrow B$, $A \longleftrightarrow C$, $B \longleftrightarrow C$, $B \longleftrightarrow D$, $C \longleftrightarrow D$, $D \longleftrightarrow t$*, such that these paths are mutually vertex-disjoint except at the ends (e.g.,* $B \longleftrightarrow C$ *and* $B \longleftrightarrow D$ *share node* B*) (see Figure 16(b)).*

Proof: Let G be the reduced graph of an partially reducible ST-graph. We first prove that there exists an indivisible LSLC in G which contains more than one edge. Assume every LBCC in G contains only one edge. Let v be one of the LAPs with the largest level. If $v = s$ or $v = t$ then the largest level of LAPs is 0. This, however, is a contradiction as the reduced graph would contain only one edge while G is not completely reducible. Now consider the case when $v \neq s$ and $v \neq t$. Let (G'', s', t') be the smallest LSLC of G for which v is an AP, and let u and w be two EAPs of G'' which immediately precedes and follows v respectively, as shown in Figure 17. Let G'_1 and G'_2 be the LBCCs induced by u, v and v, w, respectively. Since v has the largest level, no SLC of G'_1 is dividable. Thus each SLC of G'_1 contains at most one edge. This implies that G'_1 contains only one edge because G is a reduced graph. Similarly G'_2 contains only one edge. Thus v is S-reducible.

This contradicts that G is a reduced graph. The contradiction is due to the assumption that each LSLC of G contains only one edge. Thus there exists at least one indivisible LSLC which contains more than one edge.

Let K be such an indivisible LSLC. Refer to Figure 18. Let the source and destination nodes of K be s_k and t_k respectively. s_k must have at least two neighbors in K, otherwise the only neighbor of s_k in K is a LAP and hence K is divisible. Let a and b be two neighbors of s_k in K. Since K is indivisible, there must exist one path, say p, from a to b which does not contain s_k or t_k. This is because if all paths from a to b pass through either s_k or t_k, then K can be divided into two slices, one contains a and the other contains b. Furthermore since K is indivisible, no node on p can be a LAP. Thus there must exist two distinct nodes, say c and d on p such that $c \overset{p_1}{\longleftrightarrow} t_k$ and $d \overset{p_2}{\longleftrightarrow} t_k$ and p_1, p_2 contain no nodes on p except c and d, respectively (c or d may be the same as a or b). Let e be the node on both p_1 and p_2 which is closest to c. This node must exist since p_1 and p_2 intersect at least at t_k. Now let $s_k = A$, $c = B$, $d = C$, $e = D$ then these 4 nodes satisfy the requirement in this lemma. □

Theorem 5 *All edges in an ST-graph are unidirectional if and only if the graph is completely reducible.*

Proof:
If part: By Theorem 3 if an ST-graph G is completely reducible then all edges in G are unidirectional.
Only if part: From Lemma 9, it is easy to see that every edge on $B \longleftrightarrow C$ is bidirectional.
□

AE-cut

Algorithms 1 and 2 assign direction to most but not necessarily all unidirectional transistors. For example, edges 5 and 6 in Figure 13 are actually unidirectional but remain unassigned. An additional algorithm that may be used to assign direction is described next. We first define *AE-cut* and then show that an edge in an AE-cut is unidirectional.

Definition 14 [*AE-cut*] *An AE-cut consists of an edge e and a node z in an ST-graph (G, s, t) such that if e is removed from G, z becomes an articulation point which separates s and t.*

Theorem 6 *Any edge in an AE-cut of an ST-graph is unidirectional.*

Proof: Consider Figure 19 where (x, y) is such an edge and z is the articulation point when (x, y) is removed. Without loss of generality, assume x and s are in the same connected component and y and t are in the other connected component of the graph when both (x, y) and z are removed. It is impossible to have two vertex-disjoint paths, one from s to y and the other from x to t because each path from s to y must go through x or z, and in either case all paths from t to x are "blocked." Thus the direction from y to x is impossible. $\square$

For an edge in an ST-graph with $|E|$ edges, a depth-first search can find whether it is in an AE-cut in $O(|E|)$ time. From Lemmas 7 and 8 it is clear that we only have to deal with the smallest subgraphs which cannot be further divided using the LAP-find algorithm. Thus $|E|$ can be very small as compared to the number of edges in the original ST-graph. For example, edge 5 and node H in Figure 13(d) form an AE-cut of the ST-graph induced by A' and B. Thus edge 5 is unidirectional and can be assigned $E \Rightarrow G$. Similarly edge 6 can be assigned $F \Rightarrow H$.

Edges with directed neighbors

It is possible to assign directions to more edges if their neighbors have been assigned directions via the algorithms discussed so far. Consider for example the ST-graph shown in Figure 13(d). No direction has been assigned to edges 4, 34, 7, 12, 19, 24. In fact these edges are all bidirectional. Consider edge 34. By definition we must find two disjoint paths $A' \longleftrightarrow E$ and $H \longleftrightarrow B$ to ensure $E \rightarrow H$, and two disjoint paths $A' \rightarrow H$ and $E \longleftrightarrow B$ to ensure $H \rightarrow E$. The following theorem, which uses the direction information obtained thus far, immediately assigns directions to some edges that were not identified earlier.

Theorem 7 *After PS-reduction, LAP-find and the AE-cut algorithms, if an edge (u, v) satisfies the following conditions, then $u \rightarrow v$ can be assigned.*

1. *(u, v) has not been assigned a direction.*
2. *There exists a neighbor w of u such that $w \Rightarrow u$ has been assigned.*
3. *There exists a neighbor x of v such that $v \Rightarrow x$ has been assigned.*

Proof: Since (u, v) has not been assigned a direction, neither u nor v is a LAP. Thus w, u, v and x must be in the same indivisible LSLC, say (G', s', t'). $v \Rightarrow x$ can be assigned only due to (1) x is a LAP (x may be the same as t), or (2) (v, x) is in an AE-cut of G'. If x is a LAP it must be the same as t'. Similar statements hold for (w, u). Thus we may consider two mutually disjoint cases: (1) both w and x are LAPs or (2) at least one of w and x is not a LAP.

Case 1: $w = s', x = t'$: Since $(s' \longleftrightarrow u, v \longleftrightarrow t')$ in G' is true, thus $u \rightarrow v$ in G' follows. By Lemma 8, $u \rightarrow v$ in G is also true.
Case 2: $w \neq s'$ or $x \neq t'$: Without loss of generality, assume $x \neq t'$. Refer to Figure 20(a). Let y be the AP and G'_1 and G'_2 be the two parts of G' separated by y when (v, x) is removed. There must exist a path, say p_1, from x to t' in G'_2 which does not go through y, otherwise G'_2 will be separated into two disjoint parts by y and so will G' (see Figure 20(b)). This implies that y is an articulation point of G' , which contradicts the assumption that G' is indivisible. There must also exist a path, say p_2, from s' to u in G'_1 which does not pass through v in G'_1, otherwise v will separate G'_1 into two disjoint parts making it an AP of G' (see Figure 20(c)). Clearly p_1 and p_2 are vertex-disjoint and the theorem follows. □

By virtue of this theorem, all the remaining unassigned edges in Figure 13 can be immediately identified as bidirectional. Since in general most transistors in real circuits are unidirectional, the probability that Theorem 7 can be applied to an unassigned edge is quite high. It should also be pointed out that despite its simplicity, this theorem is by no means trivial. The following apparent "counter" example will highlight the difficulty.

Refer to Figure 21 where $w \Rightarrow u$ and $v \Rightarrow x$ have been assigned. Because of Theorem 7, it appears that one may assign $u \rightarrow v$. However this is not true as only $v \Rightarrow u$ is possible. This "contradiction" arises because in this example the direction of (u, v) would have been assigned while Algorithms 1, 2 or the AE-cut are executed. Thus condition 1 of Theorem 7 is not satisfied.

It can be verified that when proving Case 2 of Theorem 7, only condition 2 or condition 3 of the Theorem, but not necessarily both, is required. In other words, if one of the two edges (w, u) and (v, x) has been assigned a direction ($w \Rightarrow u$ or $v \Rightarrow x$) by the AE-cut algorithm, then no matter whether the direction of the other edge has been assigned or not, $u \rightarrow v$ can be assigned. The following theorem formally states this fact.

Theorem 8 *After PS-reduction, LAP-find and the AE-cut algorithm, if an edge (v, x) is assigned a direction $v \Rightarrow x$ by the AE-cut algorithm, then any unassigned edge (u, v) can be assigned a direction $u \rightarrow v$ and any unassigned edge (x, y) can be assigned a direction $x \rightarrow y$.*

6 Remaining edges

If there still exist unassigned edges after all the above procedures have been executed, several approaches are possible for assigning directions to them. One can simply regard the unassigned edges as bidirectional. Clearly, the over-all result will be pessimistic but not wrong.

Since it is possible to efficiently count the number of paths between two nodes—though enumerating them is computationally intensive—one may wish to first count the number of paths from s to u and v to t in order to assign direction to the edge (u, v). If the product of these two counts is not very large, one can determine the existence of two disjoint paths via exhaustive search. Because the problem size has become much smaller, it is likely that the largest indivisible LSLC containing unassigned edges is quite manageable. An advantage of this approach is that it can use the direction information obtained by earlier procedures to guide the search for disjoint paths and prune the search space.

If the above two approaches are not applicable, one can use the algorithm presented by Seymour [7] for a general two-paths problem to complete the entire direction assignment task. In order to assign direction to an edge, only the smallest LSLC containing it is considered. For the sake of completeness, the following procedure, adopted from [7], can be used to determine if the direction $u \rightarrow v$ is feasible for an unassigned edge (u, v) in an ST-graph (G, s', t').

Decide if there exists a non-empty set of nodes $A \subseteq V - \{s', u, v, t'\}$ with $|\delta(A)| \leq 3$, where $\delta(A)$ is a set of vertices defined as $\delta(A) = \{x | x \notin A$ and $\exists$ a vertex $a \in A$ such that $(x, a) \in E\}$.
If "yes", choose such an A with $G|A$ connected, where $G|A$ is the subgraph $(A, <A>)$ of G with $<A>$ being a set of edges with both ends in A. Delete A from the graph and add new edges joining every pair of distinct vertices in $\delta(A)$ to form G'. (It is easy to see that the two paths exist in G if and only if they exist in G'). Repeat with G' replacing G.
If "no", add two new edges (s', u) and (v, t') to G to form G', and test if G' can be drawn in the plane with (s', u) and (v, t') crossing once and with no other crossings. (It can be done by modifying a planarity testing algorithm). If "yes", then the paths do not exist; if "no", then they do.

The details of the algorithm can be found in [7].

7 Extension to Non-static Circuits

So far we have assumed that the circuits under consideration are static. In this section we discuss the applicability of our method to non-static circuits, which refer to those circuits whose operations rely on charge retention or charge sharing effects. We shall consider two types of circuits: *non-static logic gates* and *non-static memory cells.*

Non-static logic gates

According to the classification used in [9], there are mainly 4 types of non-static circuits: (1) dynamic logic, (2) clocked CMOS logic (C^2MOS), (3) CMOS domino logic, and (4) cascade voltage switch logic (CVSL). The basic structure of these logic gates are shown in Figure 22(a)–(d). Detailed operations of these circuits are out of the scope of this paper. We shall only examine some properties of these circuits which affect the signal flow directions of transistors.

These circuits share two common properties. (1) They are all precharged logics, and (2) to guarantee proper operations, the charge sharing effect should not invalidate the condition set up during the precharge phase. Thus, in a properly designed circuit, charge retention rather than charge sharing determines the final state of the circuit. Therefore the precharged node in a transistor group (or logic gate) is always the signal flow destination of that group. Thus the ST-graph model and the direction assignment algorithms presented in this paper can be extended to all these circuits.

The above discussion also applies to some special precharge circuits if the requirement that charge sharing effect does not invalidate the precharged state is met. For example, Figure 23 shows a dynamic Manchester carry chain [9] and its corresponding ST-graphs. Both ST-graphs can be reduced to one single edge by PS-reduction, and thus all transistors can be assigned a proper direction in time proportional to the size of the circuit.

Non-static memory cells

A typical dynamic D flip-flop design is shown in Figure 24 where A and B are two storage nodes. Since A and B are both connected to the gate terminal of the inverters, they are the output nodes of the corresponding transistor groups. It can be seen that A and B only serve as the signal flow destination. Thus our direction assignment algorithms assign correct directions to the pass transistors as shown in Figure 24.

In some designs a storage node can serve both as a source and a destination of signal flow. Consider, for example, the circuit in Figure 25(a) which shows a typical design of a dynamic RAM. The source and drain of transistor M are connected to form a capacitor. The value of node v can be read or written through the pass transistor PA. The reading process relies on the charge sharing between nodes v and u and the fact that v dominates u if u is not driven by any other source. While writing, u is controlled by a strong power source and the voltage value at u can overwrite the value at v. Since v can serve as both a signal flow source and destination, PA is used as a bidirectional transistor in this design. In order to correctly model this circuit we add two new edges for v when constructing the ST-graph. One edge connects v to s and the other connects v to t, as shown in Figure 25(b). Note that we do not merge v into s but just connect it to s. Our algorithms will now identify transistor PA as bidirectional. In general for a non-static circuit which relies on charge sharing, any storage node which dominates the charge sharing effect (e.g., v in the above example) can be modeled in this manner.

8 Limitation

One limitation of our method is that it does not consider the semantics of the circuit. It is possible that a path from s to t contains transistors that cannot be turned on at the same time. Consider, for example, the circuit shown in Figure 26. The existence of paths p_1 and p_2 does not guarantee the direction $X \rightarrow Y$ because transistors T_1, T_2 and T_3 cannot be turned on simultaneously. This problem is in general very difficult to solve for two reasons. First, it is difficult to derive the boolean function which encodes the conditions that cause a transistor to turn on purely from a structural description of the circuit. Second, even if the activation condition for each transistor were known, determining whether all transistors in a path can be turned on simultaneously is equivalent to solving the general satisfiability problem [11], a well-known NP-complete problem.

Fortunately since the goal of direction assignment algorithms is to identify as many unidirectional transistors as possible, the above problem does not invalidate the direction assignment results. Our disregard for the semantics of circuits may result in *pessimistic*, but never *wrong* results, i.e., it only results in some unidirectional transistors being assigned as bidirectional, but never the other way. Furthermore once a transistor is identified as unidirectional by our algorithms, the assigned direction must be correct because the other direction is impossible.

9 Applications to other problems

The method used for reducing the ST-graph and finding the local articulation points can be applied to many other problems. We shall give two examples, viz., the *maximum flow problem* and the *reliability modeling problem*, where the algorithms presented in this paper can greatly reduce the computational effort.

Consider the problem of finding the maximum flow from one node s to another node t in an undirected graph (similar argument can be applied to directed graphs). We may consider the graph as an ST-graph and use PS-reduction to obtain a reduced graph. When executing an S-reduction (P-reduction), the capacity of the new edge is the minimum (sum) of the capacities the two old edges. After PS-reduction, we can use the LAP-find algorithm to partition the graph into indivisible LSLCs. The maximal flow of each LBCC is the sum of the flows of its SLCs, and the maximal flow of each LSLC is the minimum of the flows of its BCCs. In this manner, the problem size of the maximal flow problem is reduced to the size of the maximum indivisible LSLC, which can be very small compared to the original graph.

The reliability modeling problem refers to the problem of determining the probability that a network of components will operate correctly, given that each component C_i can independently fail with a probability of $(1 - r_i)$. It is assumed that if two components are connected in series both of them must operate correctly to guarantee correct operation of the network. Also for components connected in parallel, at least one should be functional. The network is assumed to have two predefined nodes s and t which act as the source and sink of data. One can model the component graph with an ST-graph and apply PS-reduction to reduce it. The reliability of the new edge when executing an S-reduction and a P-reduction on two edge e_i and e_j are $r_i \cdot r_j$ and $1-(1-r_i)(1-r_j)$, respectively. The LAP-find algorithm then can be used to partition the graph into indivisible LSLCs. The reliability of a LBCC containing k SLCs, $SLC_1, SLC_2, \ldots, SLC_k$, is given by $1 - \prod_{i=1}^{k}(1 - R_i)$ where R_i is the reliability of SLC_i; and the reliability of a LSLC containing $BCC_1, BCC_2, \ldots, BCC_l$ is $\prod_{j=1}^{l} R_j$ where R_j is the reliability of BCC_j. Again, the problem size can be reduced dramatically.

10 Conclusion

We have presented a new method for assigning direction to MOS transistors. Our method is based on formal graph theoretic results rather than *ad hoc* techniques. A new circuit model and a sequence of simple and efficient algorithms have been given. All static circuits and most non-static circuits can be processed using our method. By slightly modifying the model, the direction problem for some special dynamic circuits can also be solved. Finally, the methodology and concept described in this paper are not limited to the direction assignment problem only. We have shown that our method can also be applied to other problems such as the maximum flow problem and the reliability modeling problem.

References

[1] N.P. Jouppi. Derivation of signal flow direction in MOS VLSI. *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, CAD-6(3):480–490, May 1987.

[2] M.A. Cirit. Switch level random pattern testability analysis. In *Proc. Design Automation Conf.*, pages 587–590, 1988.

[3] H.H. Chen, R.G. Mathews, and J.A. Newkirk. An algorithm to generate tests for MOS circuits at the switch level. In *Proc. Int'l. Test Conf.*, pages 304–312, 1985.

[4] I. Spillinger and G.M. Silberman. Improving the performance of a switch level simulator targeted for a logic simulation machine. *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, CAD-5(4):396–404, July 1986.

[5] R.E. Bryant. An algorithm for MOS LSI logic simulation. *LAMBDA*, pages 46–53, Fourth Quarter 1980.

[6] Y. Shiloach. A polynomial solution to the undirected two paths problem. *Journal of the ACM*, 27(3):445–456, July 1980.

[7] P.D. Seymour. Disjoint paths in graphs. *Discrete Mathematics*, 29(1980) 293-309.

[8] N. Robertson and P.D. Seymour. Disjoint paths — A survey. *SIAM Jour. Alg. Disc. Math.*, pages 300–305, Apr. 1985.

[9] N. Weste and K. Eshraghian. *Principles of CMOS VLSI Design*, chapter 6, pages 259–269. Addison-Wesley, Mass., 1985.

[10] A.V. Aho, J.E. Hopcroft, and J.D. Ullman. *The Design and Analysis of Computer Algorithms.* Addison-Wesley Publishing Company, 1974.

[11] M.R. Garey and D.S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W.H. Freeman and Company, New York, 1979.

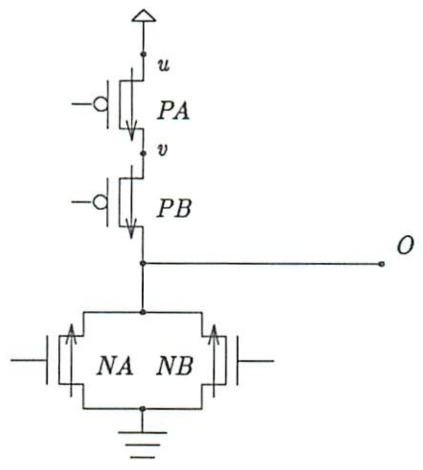


Figure 1: A NOR gate with transistor directions

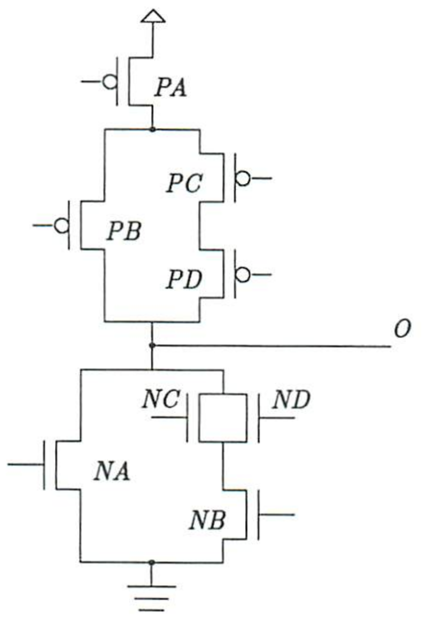


Figure 2: Transistors PB, PC, PD cannot be assigned direction by rules in [1]

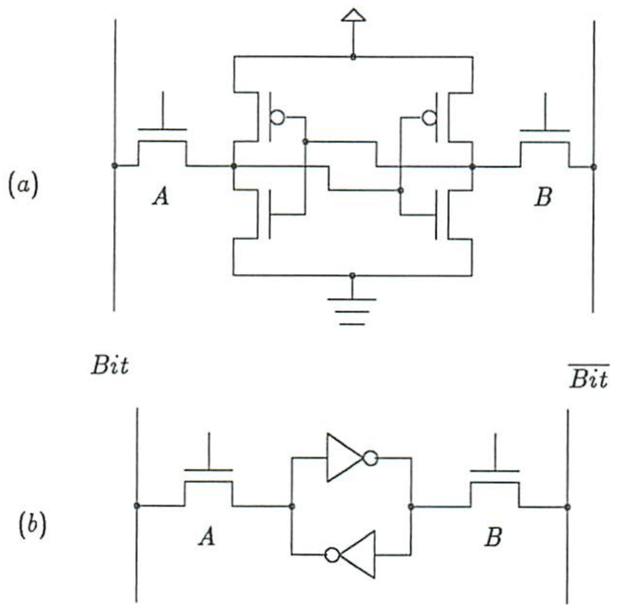


Figure 3: Transistors A and B are assigned incorrect direction by rules in [1]

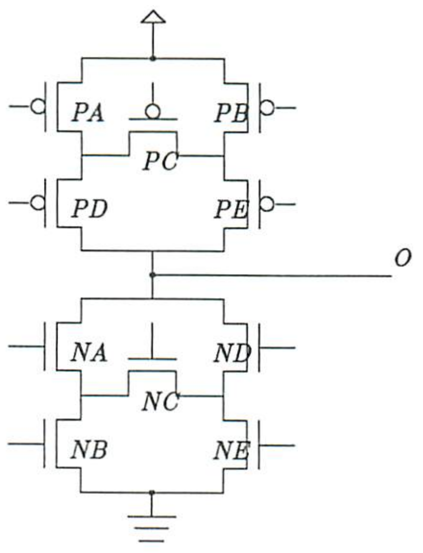


Figure 4: Transistors PC and NC are bidirectional

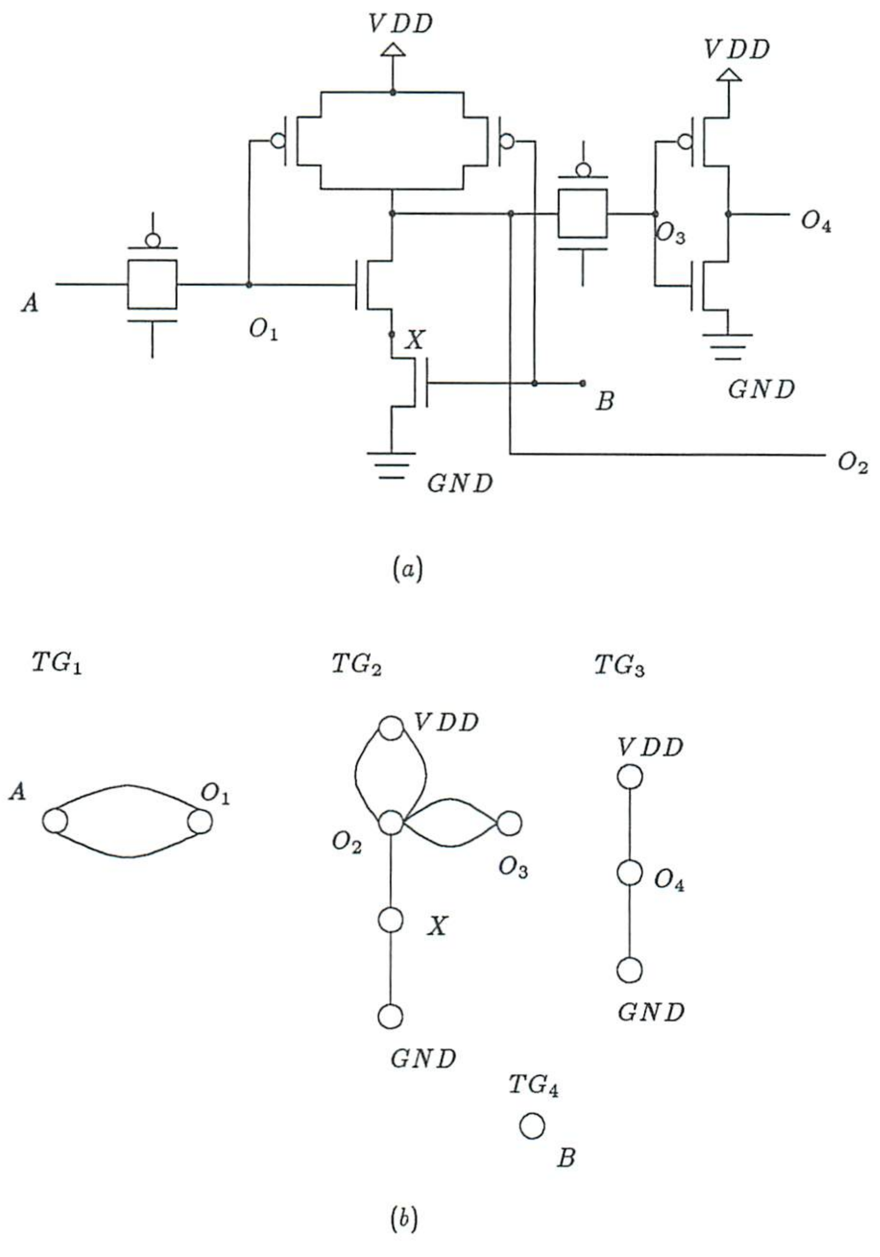


Figure 5: Transistor groups of a MOS circuit

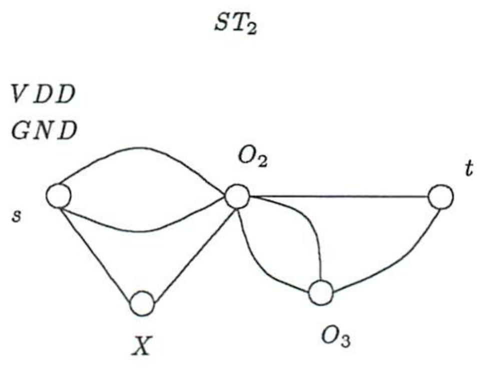


Figure 6: The ST-graph for TG_2 in Figure 5

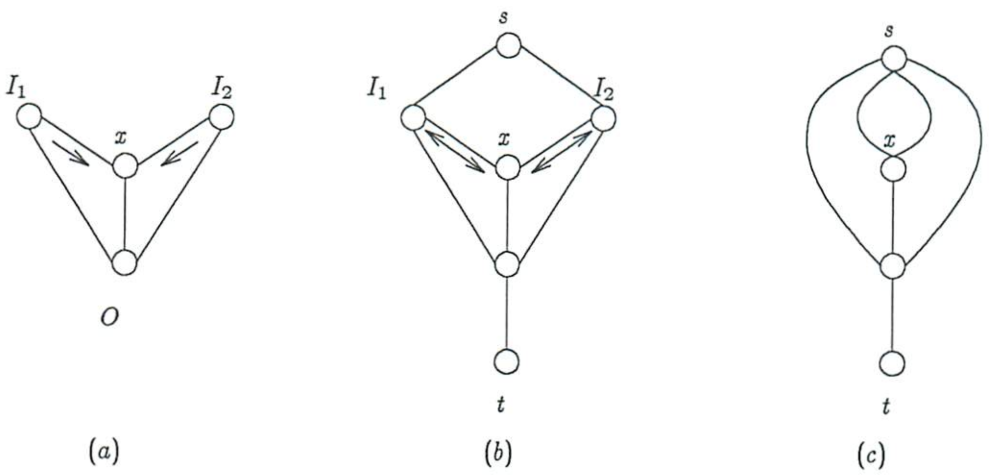


Figure 7: The importance of merging input nodes while forming an ST-graph

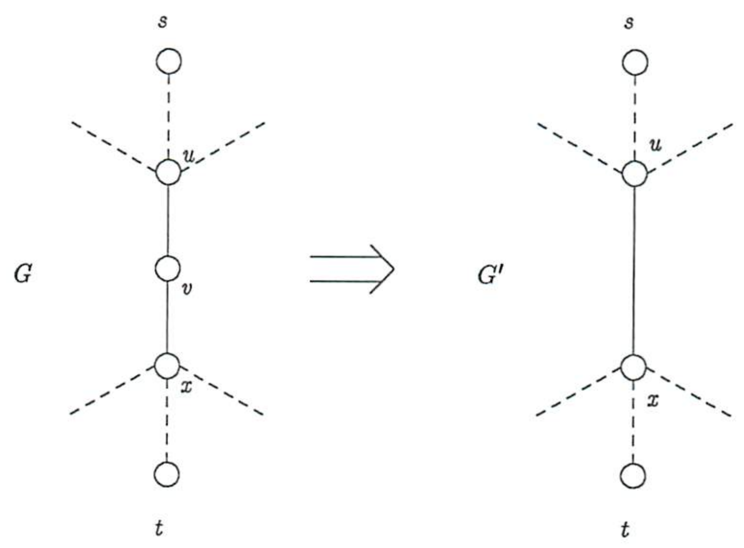


Figure 8: S-reduction

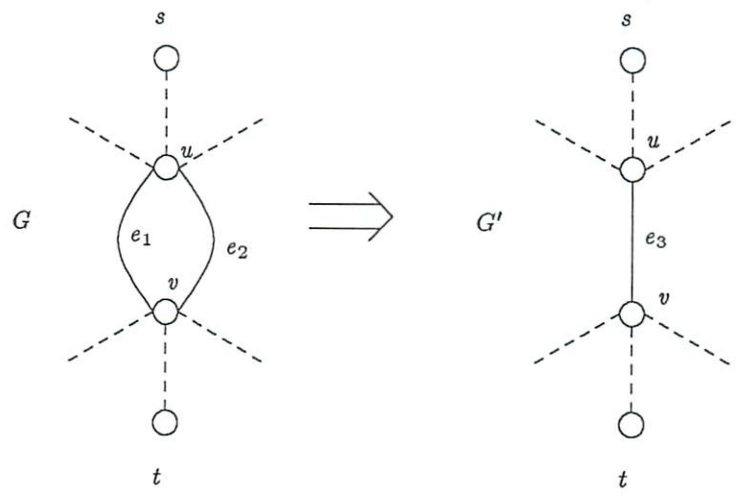


Figure 9: P-reduction

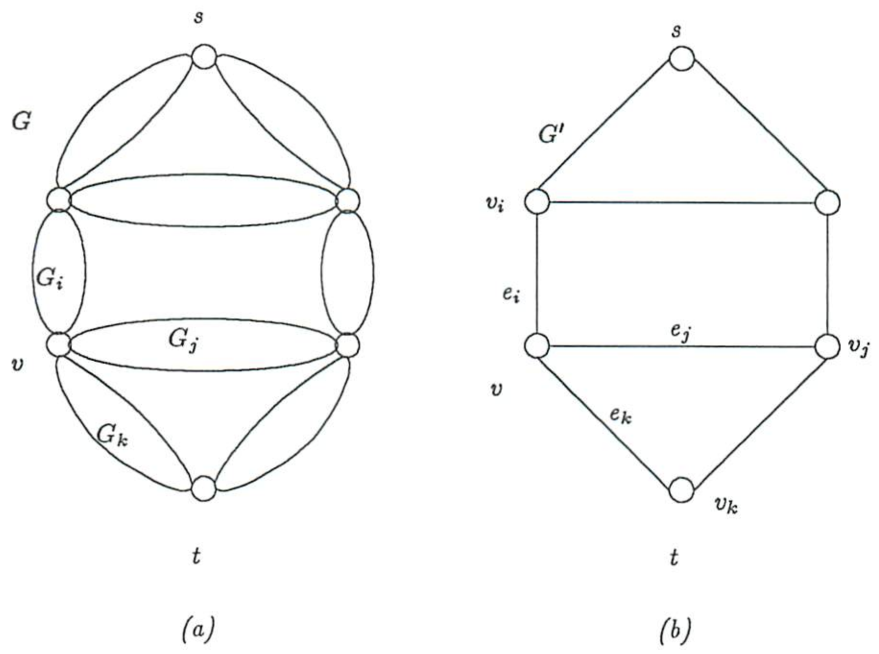


Figure 10: An ST-graph (a) and its reduced graph (b)

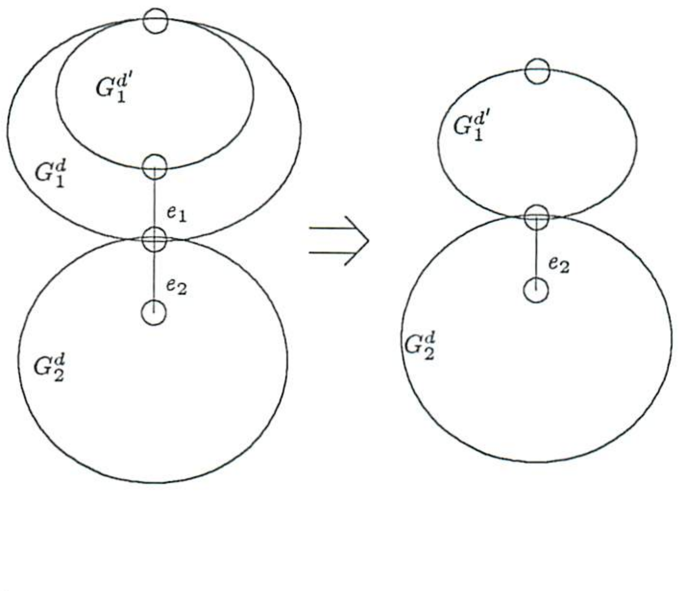
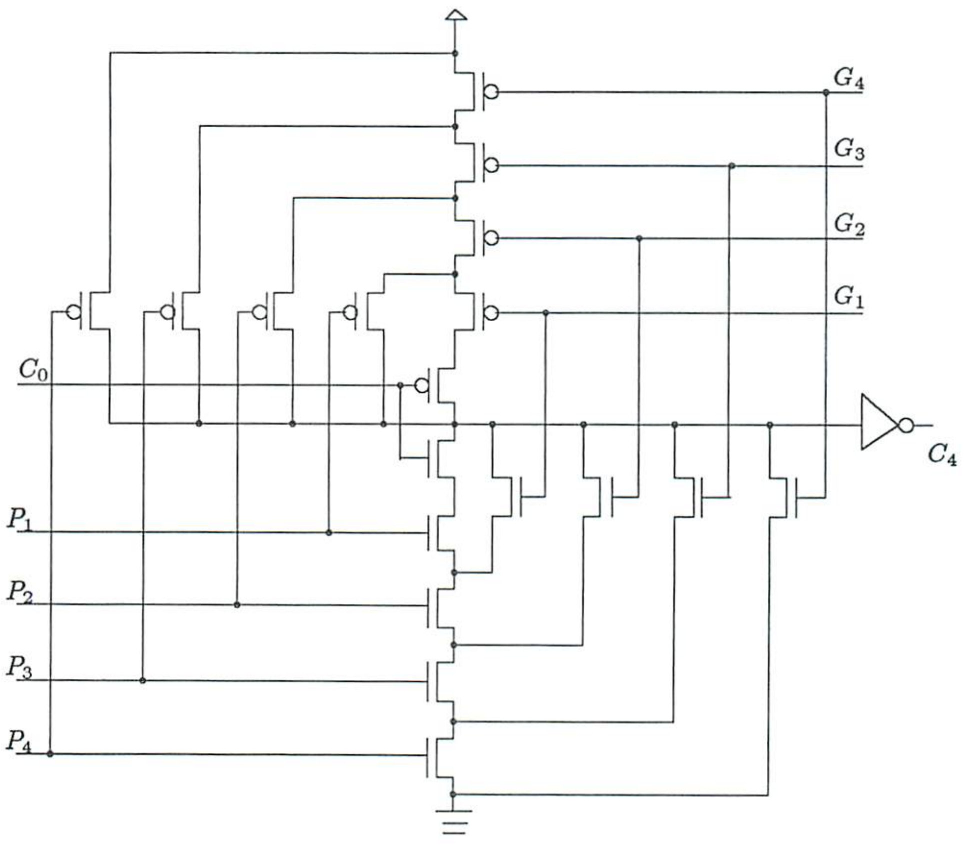


Figure 11: Proof of Lemma 3, where the S-reduction on e_1 and e_2 does not affect $G_1^{d'}$ and G_2^d



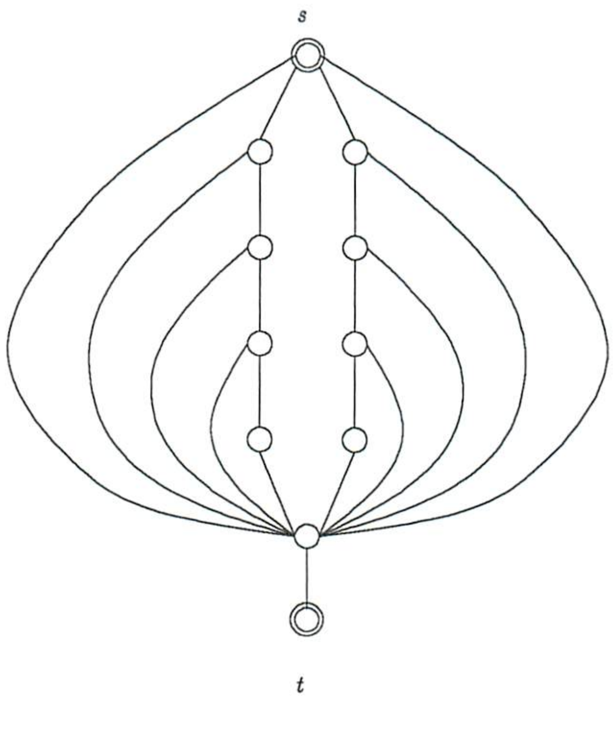


Figure 12: A static carry lookahead gate and its corresponding ST-graph

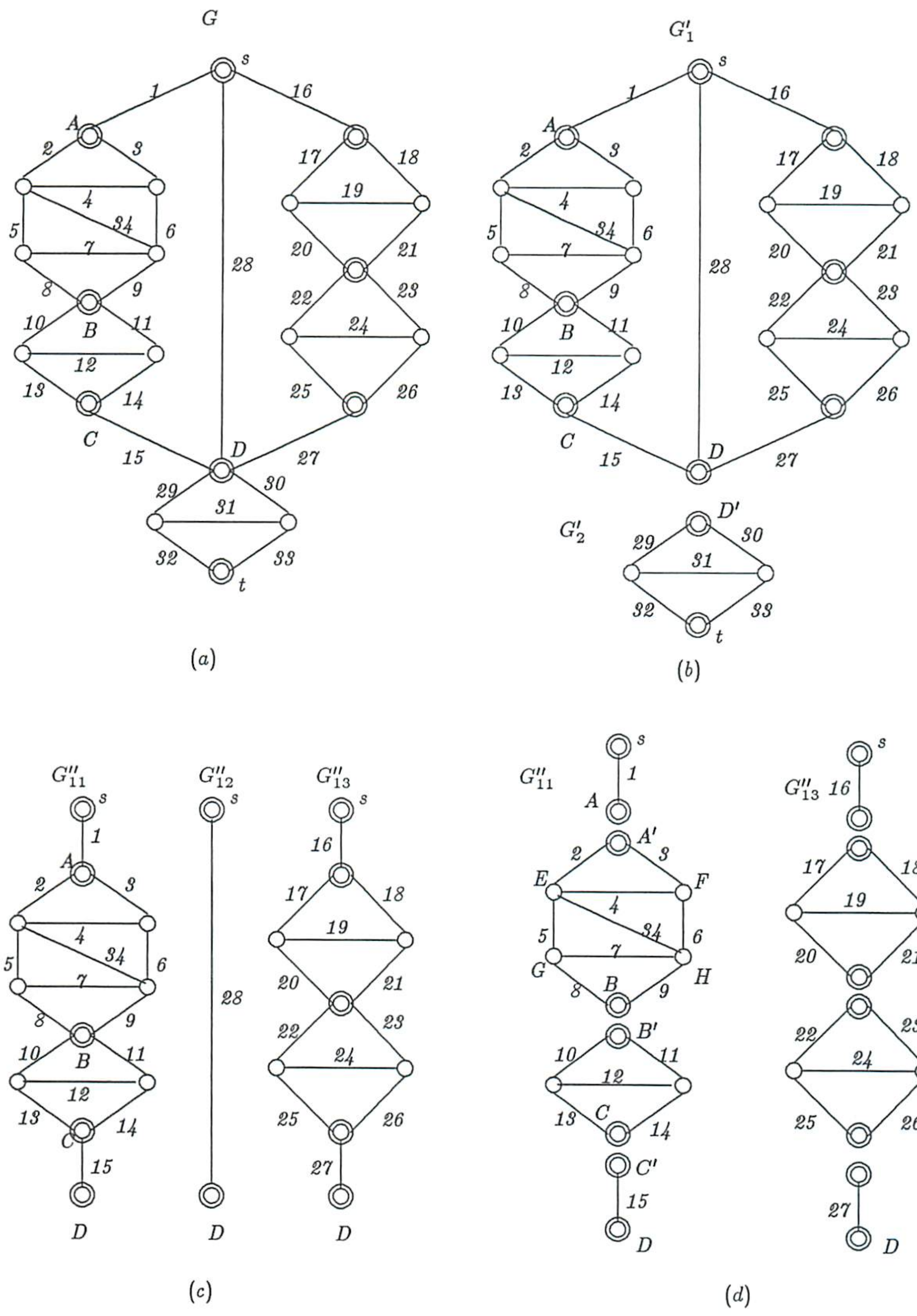


Figure 13: Local articulation points

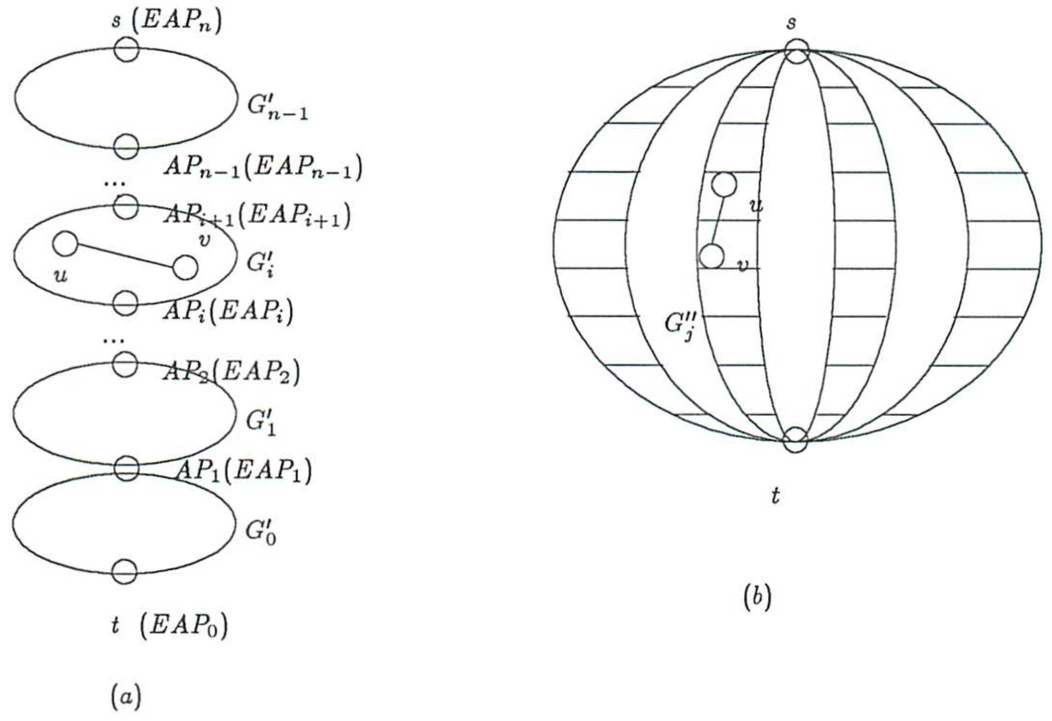


Figure 14: (a) Biconnected components and articulation points, and (b) slices

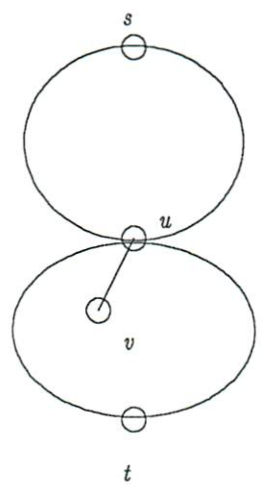
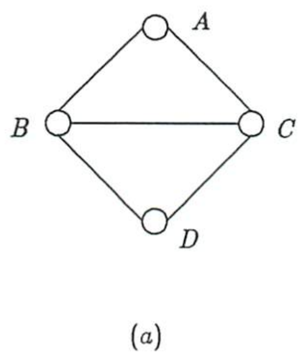


Figure 15: Unidirectionality of edges connected to articulation points



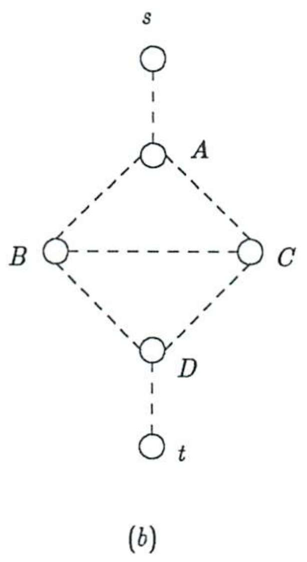


Figure 16: (a) Graph structure embedded in every partially reducible ST-graph, and (b) the seven paths used in Lemma 9

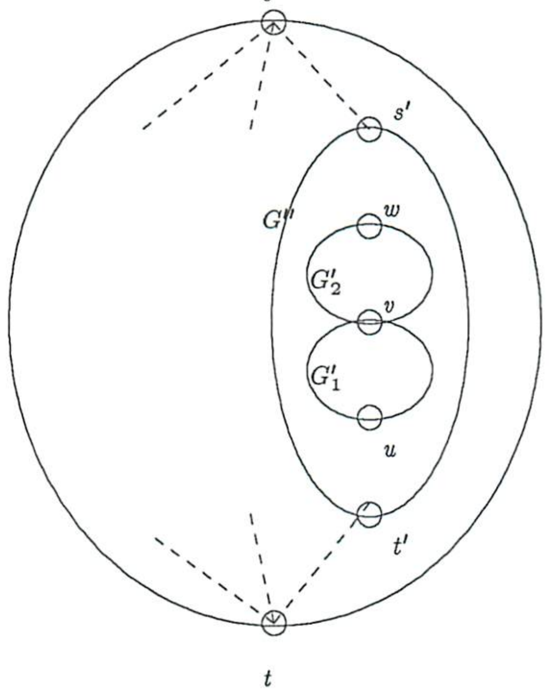


Figure 17: The smallest LSLC containing a LAP with the maximum level

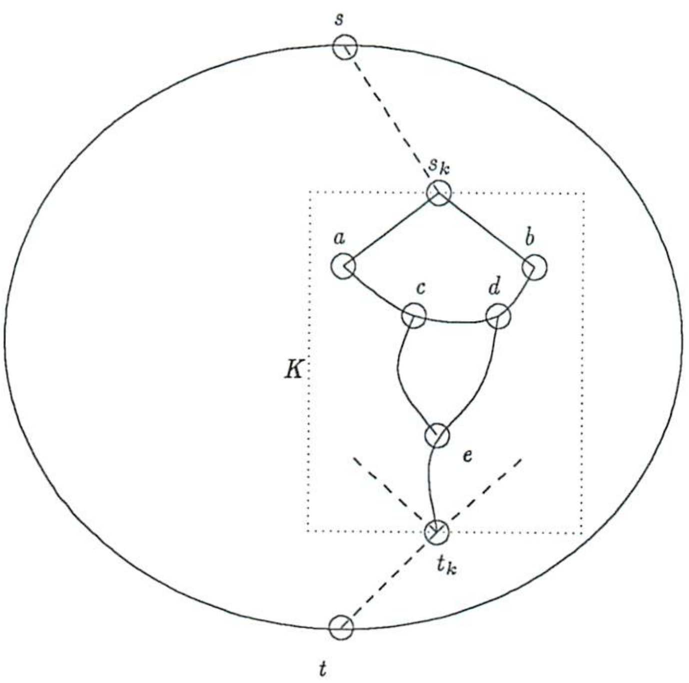


Figure 18: Proof of Lemma 9 and Theorem 5

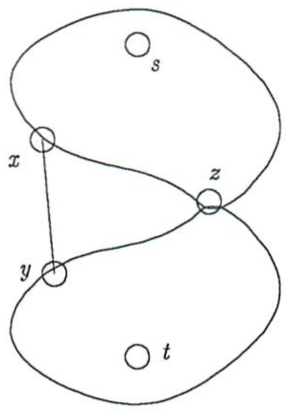


Figure 19: Unidirectional transistor in an AE-cut

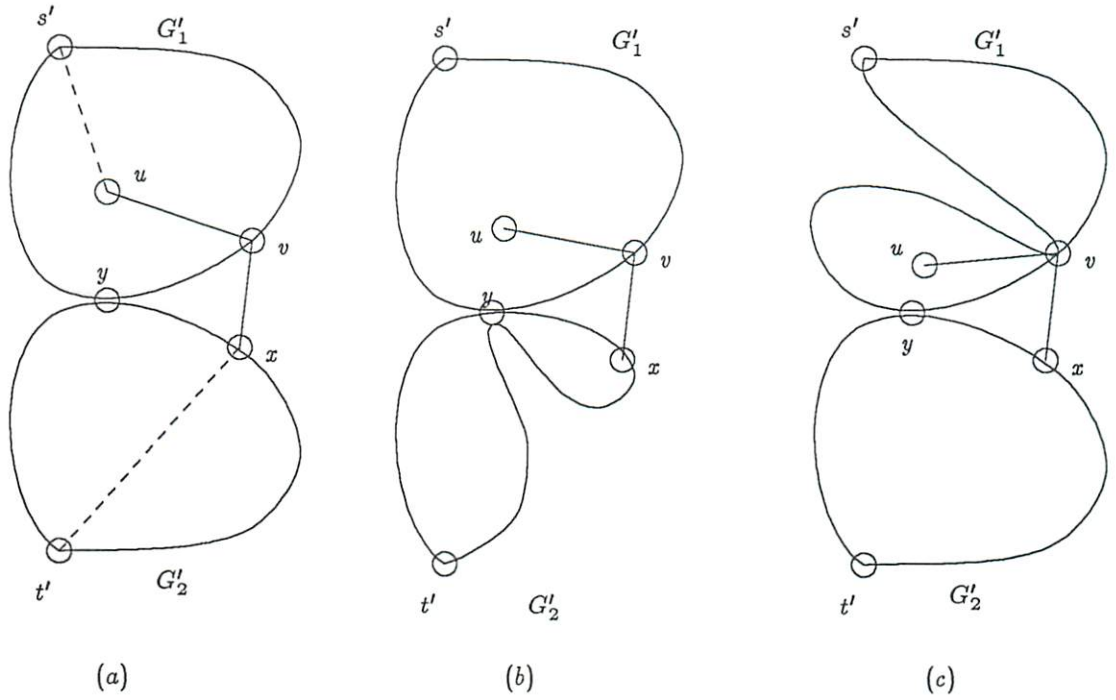


Figure 20: Assigning direction to (u, v) — Case 2

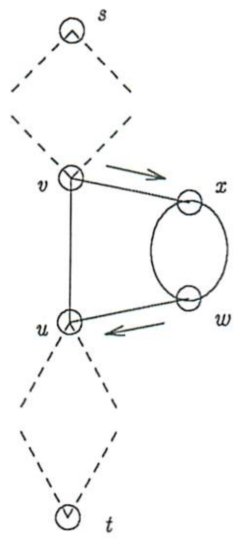


Figure 21: An apparent "counter" example to Theorem 6

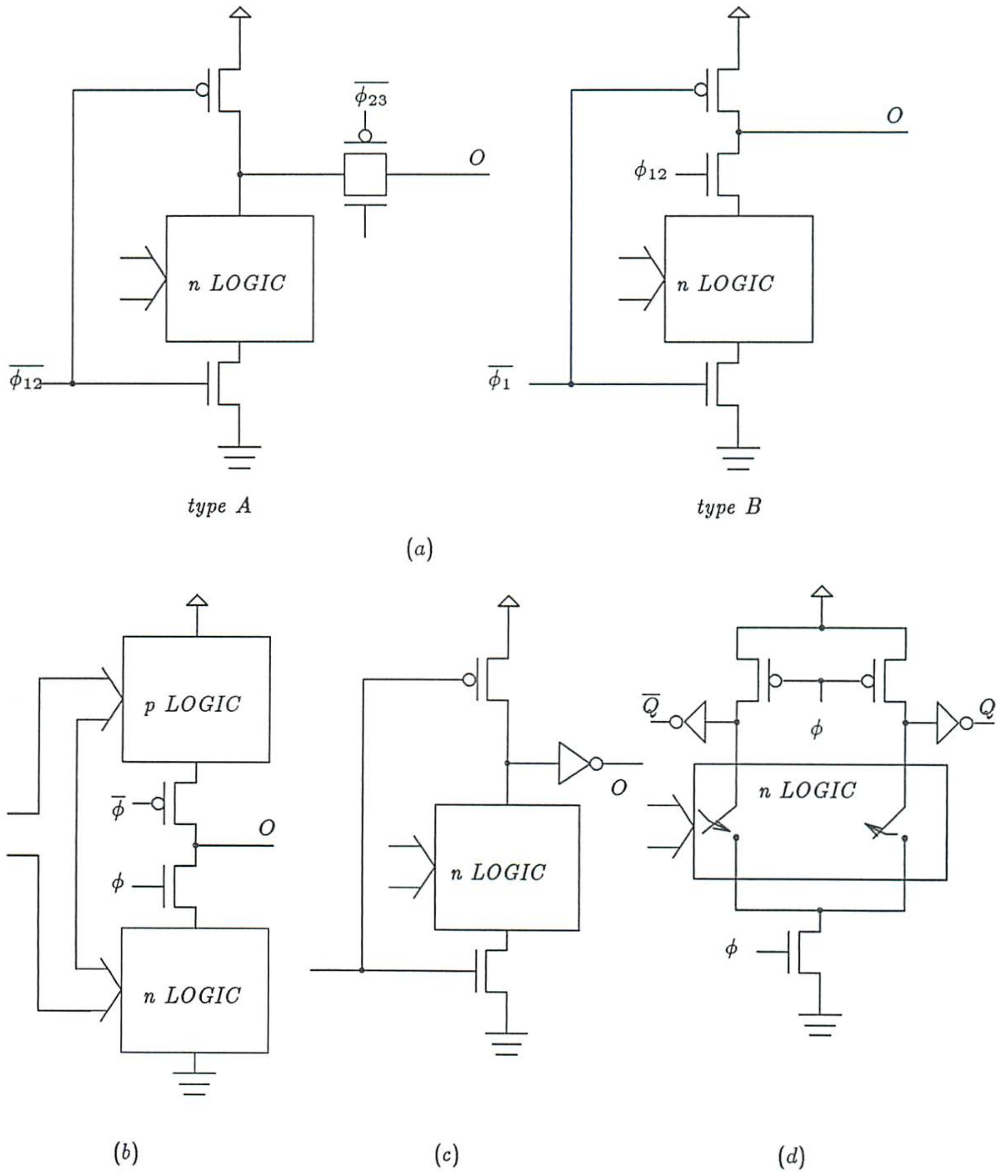


Figure 22: Non-static logic gates: (a) dynamic logic, (b) clocked CMOS logic, (c) domino logic, and (d) cascade voltage switch logic

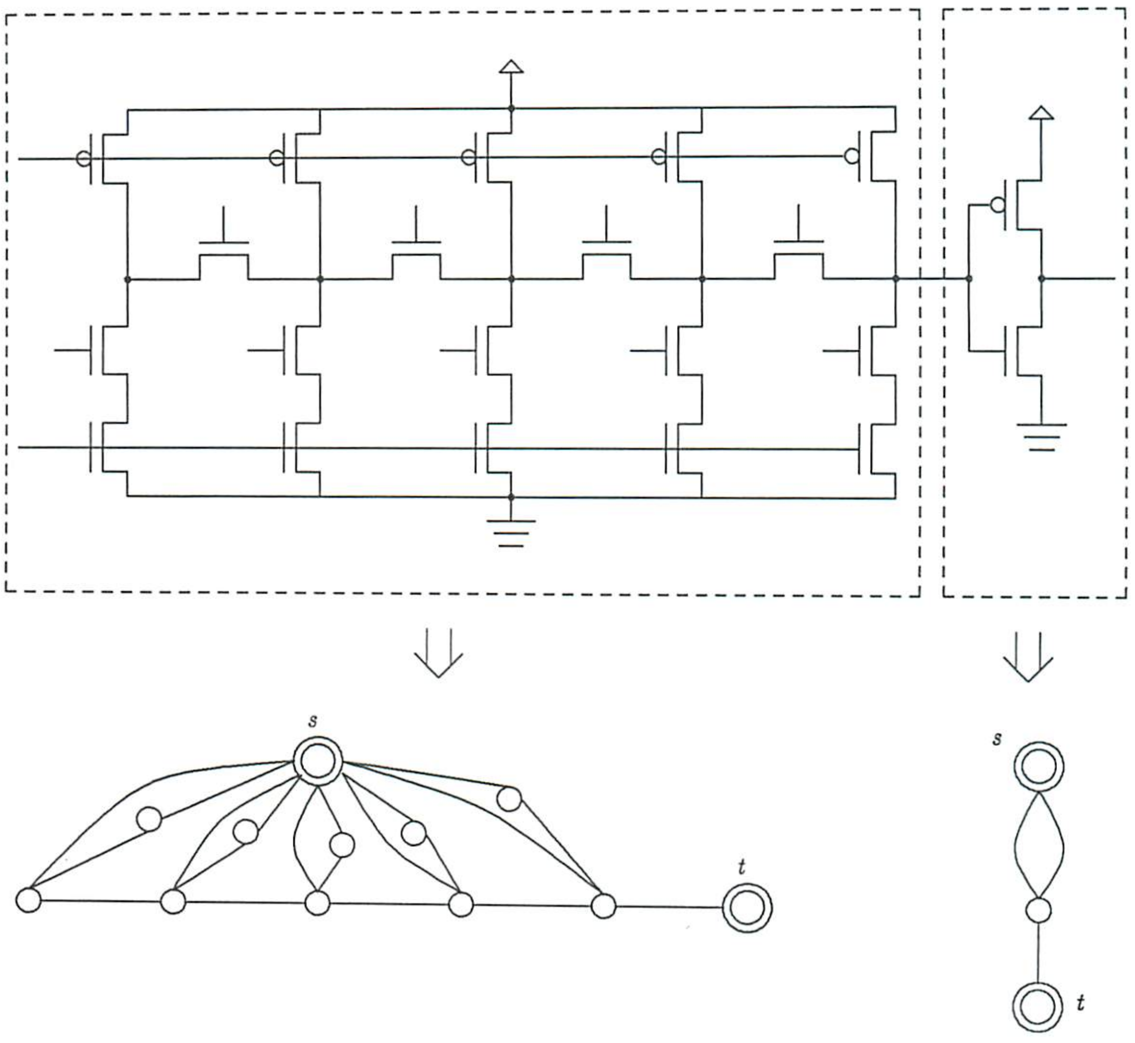


Figure 23: A Manchester carry chain and its ST-graphs

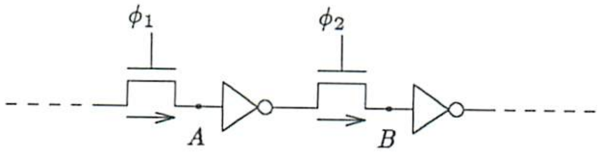


Figure 24: A dynamic Flip-Flop

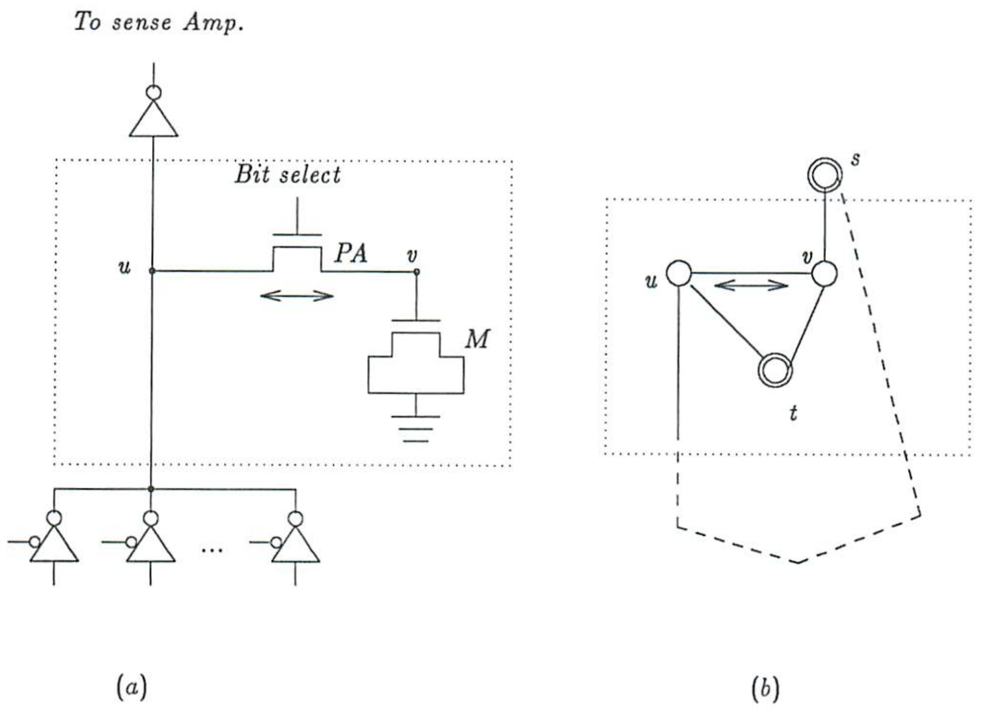


Figure 25: A dynamic RAM cell

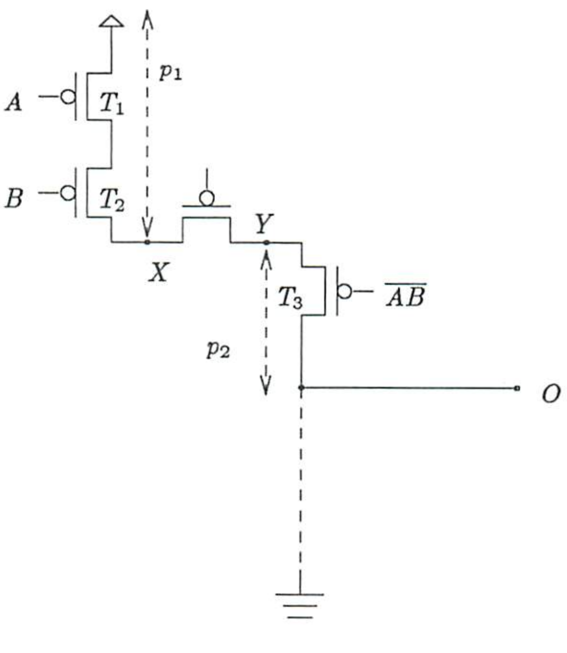


Figure 26: An example showing the effect of circuit semantics on TPP