On The Circuit Implementation Problem

Wing Ning Li, Andrew Lim, Prathima Agrawal, Sartaj Sahni

1 INTRODUCTION

Often, several implementations for each of the modules in a circuit are possible. These implementations differ in their area and power requirements as well as in their delay. The *circuit implementation problem* is that of selecting an implementation for each of the circuit modules so that the area and delay of the overall circuit are below some specified limits. In this paper we show that two rather restricted formulations of this problem are *NP*-hard. We also develop a pseudo-polynomial time algorithm for series-parallel circuits and a heuristic for more general circuits.

2 TERMINOLOGY AND NOTATION

A *circuit* is an interconnected set of modules. A p input q output module has p input pins and q output pins. Such a module is called a (p,q)–module. The interconnects of the circuit connect an output pin of one module to an input pin of another module. Some input pins are labeled as *primary inputs* and some output pins are labeled as *primary outputs*. We shall limit ourselves to circuits in which the modules can be ordered such that every interconnect connects an output pin of a module to an input pin of another module to its right. So, all interconnects are directed left to right. The graph for such a circuit is directed and acyclic. I.e., it is a dag. Figure 5.1 shows an example circuit with $n=5$ modules. The modules are shown as rectangles. Module 1 is a $(2,3)$–module while module 4 is a $(2,1)$–module. The *primary inputs* are *A*, *B*, *C*, *D*, and *E* and the *primary outputs* are *F*, *G*, *H*, and *I*.

Suppose there are n_i possible implementations of module i and that module i is (p_i,q_i)-module. Let $A(i,j)$ denote the area of the j´th implementation of this module, $1 \le j \le n_i$ and let $D(i,j,u,v)$ be the delay from input u to output v of the j´th implementation of module i, $1 \le j \le n_i$, $1 \le u \le p_i$, $1 \le v \le q_i$. A *circuit implementation* is a vector $S=(s_1,s_2,\ldots,s_n)$ where s_i is the selected

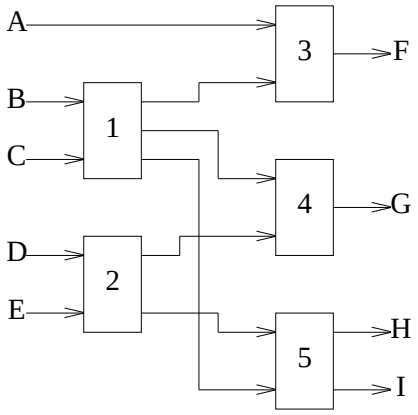


Figure 5.1: An example circuit

implementation of module *i* of the circuit ($s_i \varepsilon [1,n_i]$, $1 \le i \le n$). If $(n_1,n_2,n_3,n_4,n_5) = (2,4,3,5,3)$ for the example circuit of Figure 5.1, then $S=(2,3,1,5,3)$ is a possible implementation. There are actually $2*4*3*5*3=360$ possible implementations for this circuit. The area of the implementation $(2,3,1,5,3)$ is $A(1,2)+A(2,3)+A(3,1)+A(4,5)+A(5,3)$. The *area* of the implementation $S=(s_1,s_2,\ldots,s_n)$ is:

$$\sum_{i=1}^{n} A(i,s_i)$$

Once an implementation *S* of a circuit has been selected we can compute the delay of each primary input to primary output path. For the above implementation, the delay along the path: B, (input1 of module1 to input 1 of module4), (input 1 of module4 to output G) is $D(1,2,1,2)+D(4,5,1,1)$ (we assume that propagation delays along interconnects are factored into the module delays).

The *circuit delay* of implementation *S* is the maximum of the delays along all primary input to primary output paths. The inputs to the *general circuit implementation* (GCI) problem are a circuit as described above and two values *area* and *delay*. The output is an implementation *S* with area no more that *area* and circuit delay no more than *delay*. If no such implemen-

tation exists, the output is S =(0,0,...,0).

We now define a special class of circuits called *basic circuits*. In a basic circuit each module *i* has the property that in each of its implementations the delay between each pair of input-output terminals is the same. So, for implementation *j* of module *i* we have:

$$D(i,j,u,v)=d_{ij},\ \ 1\le u\le p_i,\ \ 1\le v\le q_i$$

The *basic circuit implementation* (BCI) problem differs from the general circuit implementation problem only in that the allowable input circuits are basic circuits.

3 COMPLEXITY RESULTS

To establish our complexity results we shall use the following known *NP*-complete problems [GARE79]:

PARTITION

Input: A finite set B and a size $s(b)\varepsilon Z^+$, for each $b\varepsilon B$.

Output: "Yes" iff there is a subset $B'\subset B$ such that $\sum_{b\varepsilon B'} s(b) = \sum_{b\varepsilon B-B'} s(b)$.

3SAT

Input: Collection of clauses C_1, C_2,..., C_n over variables x_1, x_2,..., x_n such that each clause is the disjunction of exactly three literals.

Output: "Yes" iff there is a truth assignment to the variables such that each clause is true.

Theorem 5.1: The BCI problem is *NP*-hard even when all modules are (1,1)–modules interconnected to form a chain and each module has only two possible implementations.

Proof: We shall show that any instance of the *partition* problem can be transformed, in polynomial time, into an instance of the BCI problem for which there is an (*A,D*) implementation iff the output to the partition instance is "yes". The constructed BCI instance satisfies the remaining requirements of the theorem.

The BCI instance is obtained from the *partition* instance as follows. For each $b \varepsilon B$ we have a distinct (1,1)–module, i, with two possible implementations.

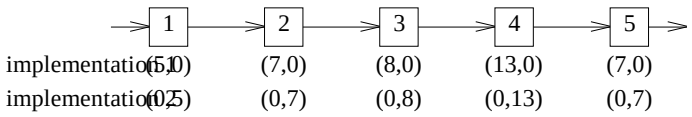
$$A(i,1)=s(b),\ D(i,1,1,1)=0$$
$$A(i,2)=0,\ D(i,2,1,1)=s(b)$$

The modules are connected in any order to form a chain and the values of A and D are $A=D=\sum_{b \varepsilon B} s(b)/2$. An example is given in Figure 5.2. One may easily verify that the constructed BCI instance has an (A,D) implementation iff the corresponding *partition* instance has output "yes". □

Partition instance:

$B=(b_1,b_2,b_3,b_4,b_5)$, $s(b_1)=5$, $s(b_2)=7$, $s(b_3)=8$, $s(b_4)=13$, $s(b_5)=7$.

BCI instance:



(The first value in each pair is the area and the second one is the delay)

$$A=D=\sum s(b_i)/2=20$$

Figure 5.2: An example of the construction used in Theorem 5.1.

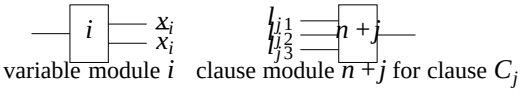
Theorem 5.2: The GCI problem is *NP*-hard even if all implementations of all modules have the same area.

Proof: This time we use the 3*SAT* problem. For any m clause n variable instance of 3*SAT* we construct a GCI instance with $n+m$ modules. These may be arranged into two columns as in Figure 5.3. The first column contains n modules called variable modules and the second one contains m modules called clause modules. Each variable module is a (1,2)–module and each clause module is a (3,1)–module. For each variable x_i in the 3*SAT* instance we have a variable

module with two implementations with delays:

$$D(i,1,1,1)=0,\ D(i,1,1,2)=1$$
$$D(i,2,1,1)=1,\ D(i,2,1,2)=0$$

Selecting implementation 1 of module i corresponds to setting x_i to true and selecting implementation 2 corresponds to setting it to false.



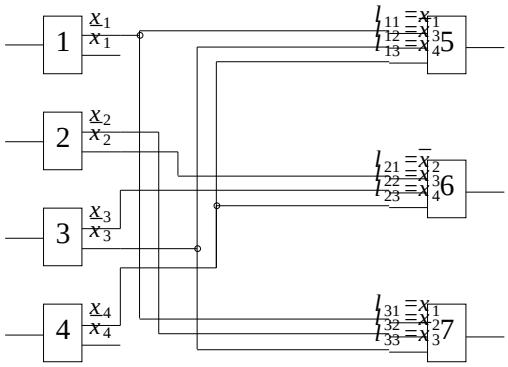
variable module i clause module $n+j$ for clause C_j

(a) (b)

3SAT INSTANCE:

$$C_1=(x_1+\bar{x}_3+x_4)\quad C_2=(\bar{x}_2+x_3+x_4)\quad C_3=(x_1+x_2+\bar{x}_3)$$

GCI INSTANCE:



(c)

Figure 5.3: An example of the construction used in Theorem 5.2.

For each clause C_j, there is a clause module $n+j$ with three possible implementations. The delays associated with these are:

$$D(n+j,1,1,1)=1,\ D(n+j,1,2,1)=0,\ D(n+j,1,3,1)=0$$
$$D(n+j,2,1,1)=0,\ D(n+j,2,2,1)=1,\ D(n+j,2,3,1)=0$$
$$D(n+j,3,1,1)=0,\ D(n+j,3,2,1)=0,\ D(n+j,3,3,1)=1$$

If $x_i = l_{jk}$ where l_{jk} is the k´th literal of C_j, $1 \le k \le 3$, then the output labeled x_i in module i (Figure 5.3(a)) is connected to the input labeled l_{jk} in module $n+j$. The delay of any primary input to primary output path is the sum of a variable module delay and a clause module delay. Each such path is uniquely characterized by the x_i and l_{jk} labels on it.

We shall now show that the 3*SAT* instance is satisfiable iff the constructed GCI instance has an implementation with area $(m+n)a$ and delay 1. Since every implementation has area $(m+n)a$ we need concern ourselves only with the delay. If the 3*SAT* instance is satisfiable then consider a truth assignment for which the instance is true. For module i, select implementation 1 if x_i is true in this truth assignment and select implementation 2 if it is false. For module $n+j$ select implementation 1 if l_{j1} is true, otherwise select implementation 2 if l_{j2} is true, otherwise select implementation 3 (in this case l_{j3} must be true). Consider any primary input to primary output path with the labels x_i and l_{jk}. If x_i is true the delay on this path is at most 1. If x_i is false, then l_{jk} is also false. The delay contributed by variable module i is 1 and that by the clause module $n+j$ is 0. So the path delay is 1. Similarly if the path labels are $\bar{x}_i$ and l_{jk} the path delay is at most 1. Hence the circuit delay is 1 (as every implementation has a delay of 1 or 2).

If there is a delay 1 implementation of the constructed GCI instance, then the 3*SAT* formula is satisfiable. To see this consider the three paths that go through module $n+j$. Regardless of the implementation of this module the clause module delay on exactly one of these is 1. The variable module delay on this path must therefore be 0. Hence clause j is true, $1 \le j \le m$. By choosing the truth assignments that correspond to the selected implementation of the n variable modules the 3*SAT* instance is satisfied. □

4 SERIES-PARALLEL CIRCUITS

A *simple parallel circuit* is a basic circuit that is comprised of several chains that have the same first and last module. Figure 5.4(b) gives an example. A *series-paralllel circuit* is a basic

circuit recursively defined as:

a) A chain of basic modules is a series-parallel circuit

b) A simple parallel circuit is a series-parallel circuit

c) A circuit obtained from a series-parallel circuit C by replacing any interconnect of C by another series-parallel circuit is also a series-parallel circuit.

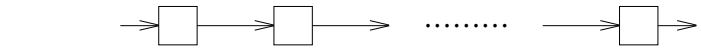
We shall assume that the area and delay values associated with each implementation of each module are integers. An algorithm for the basic circuit realization problem is a *pseudo-polynomial time algorithm* if its complexity is some polynomial in n (the number of modules), m (maximum possible implementations of any module), *area* (the maximum allowable area), and *delay* (the maximum allowable delay).

An implementation that has area a and delay d is an *optimal implementation* iff there is no implementation with area $\leq a$ and delay $< d$ or with area $< a$ and delay $\leq d$. In developing our pseudo-polynomial time algorithm for series-parallel graphs we shall first develop pseudo-polynomial time algorithms to obtain an optimal implementation for each distinct value of area and delay that is possible. This is done first for chains, then for simple parallel graphs, and finally for series-parallel graphs. The pseudo-polynomial time algorithm to obtain an optimal implementation for a given pair (*area,delay*) is easily obtained from this algorithm.

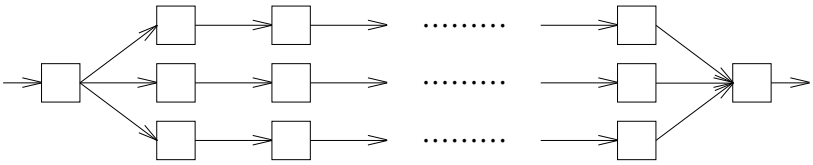
4.1 CHAINS

The optimal implementations for a chain may be obtained by enumerating the different possibilities using a dynamic programming type approach (Chapter 5, [HORO78]). Suppose the chain has c modules $M_1, M_2,..., M_c$ where M_i is to the left of M_{i+1}, $1 \leq i < c$. Let L_r be an ordered list of pairs (a_i,d_i) where a_i and d_i are, respectively, the area and delay associated with an optimal implementation of the chain $M_1, M_2,..., M_r$. L_r has the form:

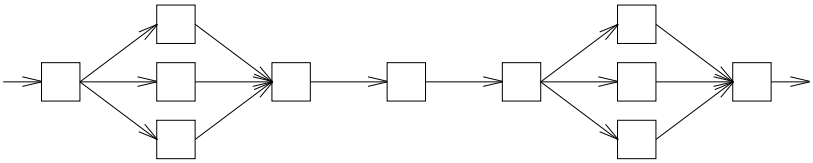
$$L_r=\{(a_1,d_1),...,(a_k,d_k)\}$$



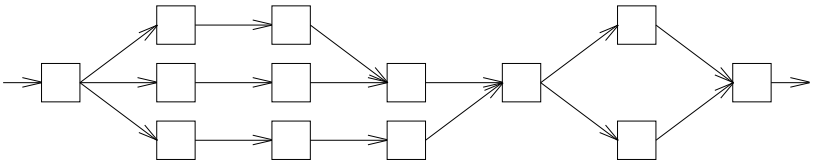
(a) Chain



(b) Simple parallel circuit



(c) A series-parallel circuit



(d) A series-parallel circuit

Figure 5.4: Series-parallel circuit examples.

where

$$a_i < a_{i+1} \text{ and } d_i > d_{i+1}, \quad 1 \le i < k \tag{1}$$

Observe that from the definition of an optimal implementation it follows that L_r cannot have two pairs (a_i,d_i) and (a_j,d_j) such that

$$(a_i \le a_j \text{ and } d_i < d_j) \text{ or } (a_i < a_j \text{ and } d_i \le d_j) \tag{2}$$

The set L_1 is simply those implementations of M_1 that are optimal. Hence if the set of implementations of M_1 includes pairs (a_i,d_i) and (a_j,d_j) that satisfy (2) then the pair (a_j,d_j) is

eliminated. Note that a suboptimal implementation of M_1 cannot be used in any optimal implementation of any of the chains $M_1, M_2,..., M_s$, $1 \le s \le c$. Similarly no suboptimal implementation of the chain $M_1, M_2,..., M_s$ can be used to obtain an optimal implementation of $M_1, M_2,..., M_{s+1}$, $1 \le s < c$.

From L_r we may obtain L_{r+1} by considering each of the possible optimal implementations of M_{r+1}. For each optimal implementation (a_e,d_e) of M_{r+1} we create an ordered list L_{r+1} as below:

$L^e_{r+1} = \varnothing$

for $i = 1$ to $|L_r|$ **do**

append $(a_i + a_e, d_i + d_e)$ to L^e_{r+1};

{(a_i,d_i) is the i´th pair in L_r}

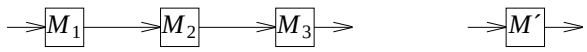
end;

Next, the lists L^e_{r+1}, $1 \le e \le$ (number of optimal implementations of M_{r+1}) are merged to obtain L_{r+1}. The merging is done so that L_{r+1} satisfies (1). This requires the elimination of pairs (a_j,d_j) that satisfy (2).

Using this strategy we begin with L_1, then obtain L_2, then L_3,..., and finally L_c. For each pair (a_i,d_i) in L_c the actual implementation can be obtained using a traceback as in Section 5.5 of [HORO78].

If we are interested only in an optimal implementation with area $\le$*area* and delay $\le$*delay* then there is no need to retain pairs (a_i,d_i) with $a_i >$ *area* or $d_i >$ *delay* in any of the lists L_r. Also, because of (1) all pairs in a list L_r have distinct a_i value and distinct d_i values. Hence all the lists L^e_{r+1} for any fixed r can be generated in O(m*max{*area*,*delay*}) time. These lists can be merged to obtain L_{r+1} in O(m*max{*area*,*delay*}) time by repeatedly merging pairs of lists and noting that the size of each merged list is at most max{*area*,*delay*}. The time to obtain L_c is

Example 5.1:



M_1 = { (5,7), (6,6), (7,5), (10,3) }
M_2 = { (10,4), (8,8), (3,9) }
M_3 = { (2,8), (4,6), (5,5) }
$L_1 = M_1$
L_2^1 = { (15,11), (16,10), (17,9), (20,7) }
L_2^2 = { (13,15), (14,14), (15,13), (18,11) }
L_2^3 = { (8,16), (9,15), (10,14), (13,12) }
L_2 = { (8,16), (9,15), (10,14), (13,12), (15,11), (16,10), (17,9), (20,7) }
L_3^1 = { (10,24), (11,23), (12,22), (15,20), (17,19), (18,18), (19,17), (22,15) }
L_3^2 = { (12,22), (13,21), (14,20), (17,18), (19,17), (20,16), (21,15), (24,13) }
L_3^3 = { (13,21), (14,20), (15,19), (18,17), (20,16), (21,15), (22,14), (25,12) }
L_3 = { (10,24), (11,23), (12,22), (13,21), (14,20), (15,19), (17,18), (18,17), (20,16), (21,15), (22,14), (24,13), (25,12) }
$M' = L_3$

therefore O(c*m*max{*area*,*delay*}).

The above algorithm for a chain also allows us to obtain a chain to module transformation in which any chain of modules can be replaced by a single module, the delay and area of whose possible implementations are given by the pairs in L_c. This transformation is shown in Figure 5.5.

4.2 SIMPLE PARALLEL CIRCUITS

A *basic tree circuit* has two columns of basic modules as in Figure 5.6(a). The second column has exactly one module. The area and delay pairs of the optimal implementations of a basic tree circuit can be obtained by first obtaining those for the column one modules. This

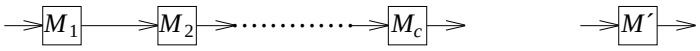


Figure 5.5: A chain to a module transformation.

allows us to replace the circuit of Figure 5.6(a) with the two module chain of Figure 5.6(b) where the possible implementations of module A have area and delay corresponding to the optimal implementations of the column 1 modules. The optimal implementation of the two module chain of Figure 5.6(b) may be obtained using the algorithm for a chain. In this way, the basic tree of Figure 5.6(a) is transformed into a single module as in Figure 5.6(c).

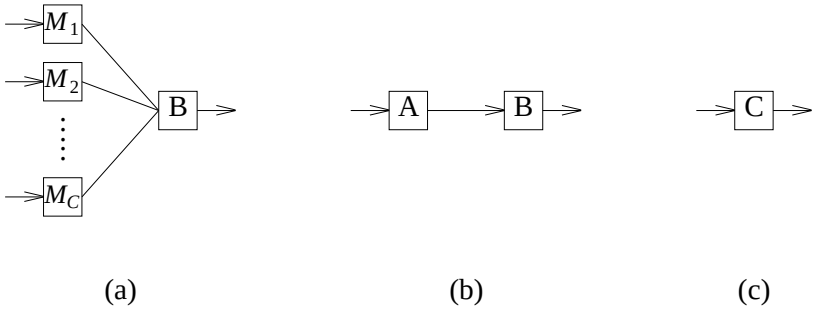
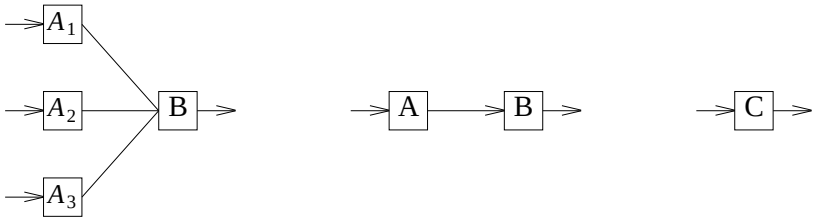


Figure 5.6: A basic tree to a module transformation.

By beginning at the leaves and using this transformation repeatedly, we can obtain the optimal implementations of any tree.

The c modules of column 1 of the basic tree of Figure 5.6(a) may be transformed into the equivalent single module A of Figure 5.6(b) by constructing the ordered lists $L_1, L_2,..., L_c$ as was done for a chain. In the code to construct L^e_{r+1} we replace (a_i+a_e, d_i+d_e) by $(a_i+a_e, \max\{d_i, d_a\})$.

Example 5.2: [Transforming a basic tree into a single module]



A_1 = { (5,7), (6,6), (7,5), (10,3) }

A_2 = { (10,4), (8,8), (3,9) }

A_3 = { (2,8), (4,6), (5,5) }

$L_1 = A_1$

L_2^1 = { (15,7), (16,6), (17,5), (20,4) }

L_2^2 = { (13,8) }

L_2^3 = { (8,9) }

L_2 = { (8,9), (13,8), (15,7), (16,6), (17,5), (20,4) }

L_3^1 = { (10,9), (15,8) }

L_3^2 = { (12,9), (17,8), (19,7), (20,6) }

L_3^3 = { (13,9), (18,8), (20,7), (21,6), (22,5) }

L_3 = { (10,9), (15,8), (19,7), (20,6), (22,5) }

$A = L_3$

B = { (1,2), (2,1) }

C = { (11,11), (12,10), (17,9), (20,9), (21,8), (22,7), (24,6) }

The optimal implementations of a simple parallel graph can be obtained by using the transformations shown in Figure 5.7. The steps are:

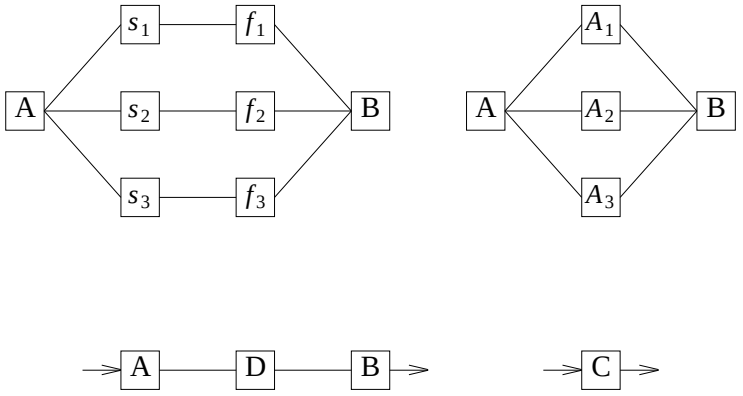
step1: The modules from s_i to f_i of Figure 5.7(a) form a chain. Using the chain transformation of Section 4.1 these are replaced by an equivalent single module A_i

step2: The modules $A_1, A_2, ..., A_c$ of Figure 5.7(b) are replaced by a single module D using the transformation used to go from Figure 5.6(a) to Figure 5.6(b).

step3: Use the chain transformation of Section 4.1 to go from Figure 5.6(c) to Figure

5.6(d).

Example 5.3: [Transforming a simple parallel graph into a single module]



s_1 = { (5,7), (6,6) } f_1 = { (7,5), (10,3) }
A_1 = { (12,12), (13,11), (15,10), (16,9) }
s_2 = { (8,8), (10,4) } f_2 = { (3,9), (8,8) }
A_2 = { (11,17), (13,13), (18,12) }
s_3 = { (2,8), (4,6)} f_3 = { (4,6), (5,5) }
A_3 = { (6,14), (7,13), (8,12), (9,11) }
D = { (29,17), (31,14), (32,13), (38,12) }
A = { (5,7), (6,6) }
B = { (4,6), (5,5) }
C = { (38,30), (39,29), (40,27), (41,26), (42,25), (43,24), (49,23) }

4.3 SERIES-PARALLEL CIRCUITS

The optimal implementation of any series-parallel circuit can be obtained by repeatedly using the chain and simple parallel circuit transformations on sub-circuits of the given series-parallel circuit. Example 5.4 illustrates how this is accomplished.

Since the number of subcircuits examined in the above process is O(n) (n is the number

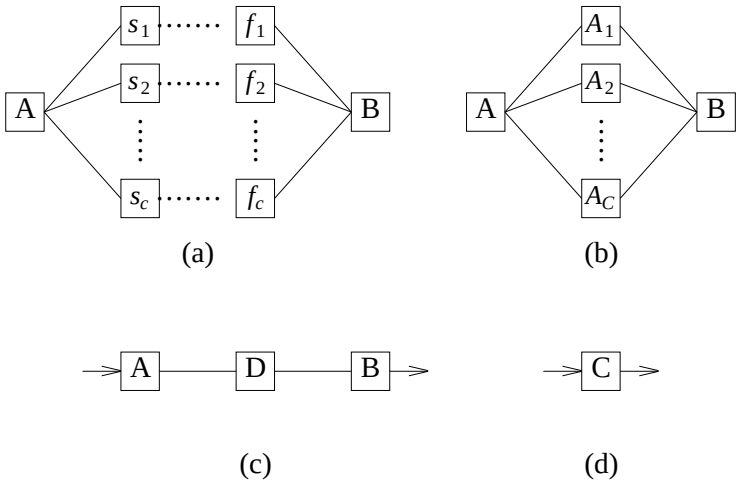
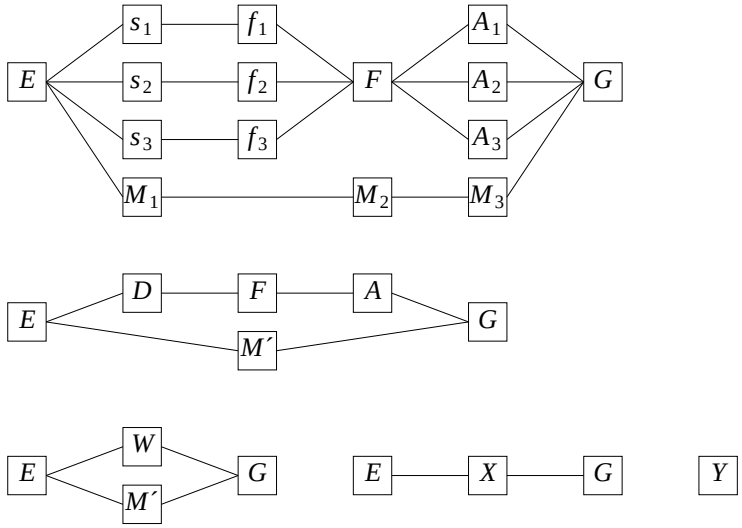


Figure 5.7: A simple parallel graph to a module transformation.

of modules), the time complexity remains pseudo-polynomial.

Example 5.4: [Transforming a series-parallel graph into a single module]



S_1 through S_3 and f_1 through f_3 are the same as in Example 5.3. A_1 through A_3 are the same as in Example 5.2. M_1 through M_3 are the same as in Example 5.1.

E = { (1,1) }

F = { (1,1) }

G = { (1,1) }

D = { (29,17), (31,14), (32,13), (38,12) }

A = { (10,9), (15,8), (19,7), (20,6), (22,5) }

$M´$ = { (10,24), (11,23), (12,22), (13,21), (14,20), (15,19), (17,18), (18,17), (20,16), (21,15), (22,14), (24,13), (25,12) }

W = { (40,27), (42,24), (43,23), (48,22), (52,21), (53,20), (55,19), (61,18) }

X = { (50,27), (52,24), (54,23), (60,20), (65,21), (67,20), (70,19), (78,18) }

Y = { (52,29), (54,26), (56,25), (62,22), (67,23), (69,22), (72,21), (80,20) }
