REDUCTIONS AND COMPLETENESS

Chapter 8

8.1 Reductions

- What is called a reduction?
- Why do we need reductions?
- Relation to the complexity classes

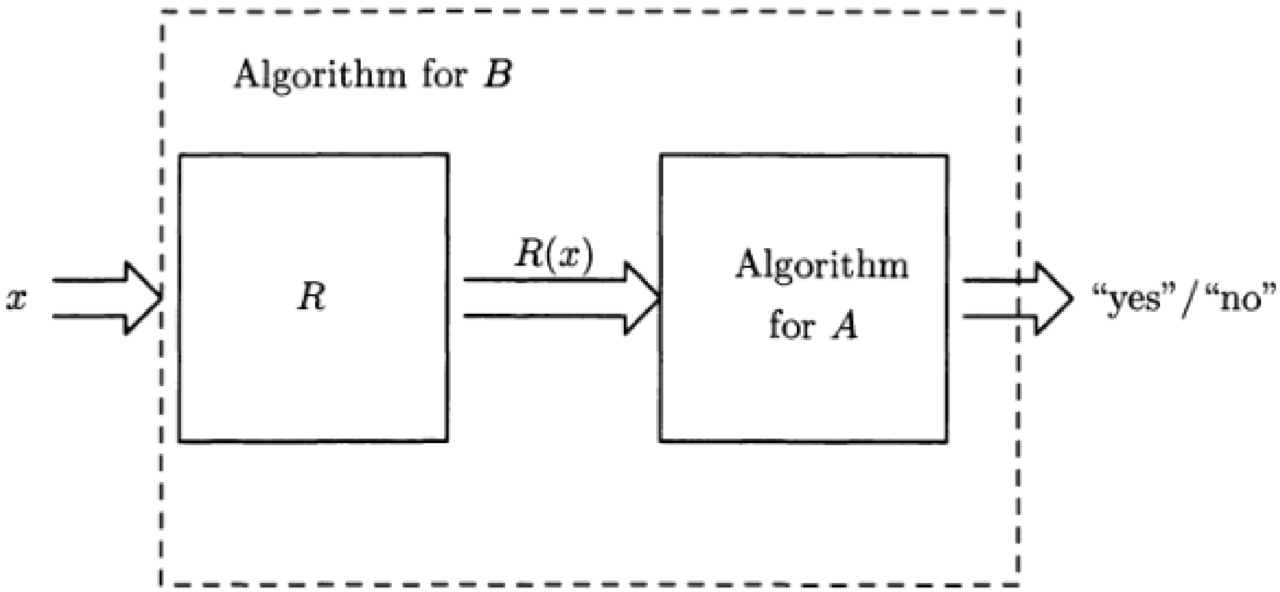


Figure 8-1. Reduction from B to A.

- Definition 8.1: We say that a language L1 is reducible to L_2 if there is a function R(x) from strings to strings computable by a deterministic Turing machine such that for all inputs x,
  $x \in L_1 \Leftrightarrow R(x) \in L_2$. An "efficient reduction" uses O(logn) space to be computed by a deterministic Turing machine.

- Proposition 8.1: If R is a Reduction as defined above then it will be computed in a polynomial number of steps.
- Proof: We have f(n)=O(logn) bits of storage ,where n=|x| (length of the input) , and k states of the turing machine. So the possible configurations are: $k*n*2^{f(n)} = O(n*c^{logn}) =$ O(poln). If one of them is repeated, then the machine will not halt. So every computation is completed in a polynomial number of steps.

- Example 8.1: Reduction of HAMILTON PATH to SAT
- Given a graph G we shall construct a boolean expression R(G) s.t: R(G) is satisfiable iff G has a hamilton path. The construction is as follows:

  We introduce the boolean variables:

  X_{ij}: “Node j is the ith node in the Hamilton path”.

  R(G) will be in CNF form with clauses:
- Node j must appear in the path: $\forall j (x_{1j} \vee x_{2j} \vee \ldots \vee x_{nj})$

- Node i cannot appear both ith and kth:

$$\forall j \forall i \neq k(\neg x_{ij} \vee \neg x_{kj})$$

- Some Node must be ith:

$$\forall i(x_{i1} \vee x_{i2} \vee \ldots \vee x_{in})$$

- No two nodes should be ith:

$$\forall i \forall j \neq k(\neg x_{ij} \vee \neg x_{ik})$$

- If (i,j) is not an edge of G, then j shouldn't come after I in the Hamilton path:

$$\forall i \forall j[(i, j) \notin E(G) \Rightarrow (\neg x_{ki} \vee \neg x_{k+1,j})]$$

- Now suppose R(G) has a satisfying assignment T.

$$\forall j \exists! i : T(x_{ij}) = true$$
$$\forall i \exists! j : T(x_{ij}) = true$$

So let π(i)=j iff $T(x_{ij})$=True be a permutation of the nodes of G.

Also the clauses of the form: $(\neg x_{k,i} \vee \neg x_{k+1,i})$ guarantee that for all k, (π(k),π(k+1)) is an edge of G <=> (π(1),π(2),…,π(n)) is a Hamilton path of G.

- Conversely, suppose that G has a Hamilton path $(\pi(1),\pi(2),\ldots,\pi(n))$, where π is a permutation.

  Then by definition the truth assignment T: $T(x_{ij})$=True if $\pi(i)=j$, and $T(x_{ij})$=false if $\pi(i)\neq j$, satisfies all clauses of R(G).
- Space complexity of the reduction:

  A turing machine that will carry out this computation needs only 3 counters i,j,k to produce all the clauses. So the length of the binary representation of these counters is **O(logn)** where n=|x| because i,j,k<=n.

This completes the reduction.

- Example 8.2: Reduction of REACHABILITY to CIRCUIT VALUE
- Given a graph G, we are going to construct a variable-free circuit R(G) such that the output of R(G) is True iff there is a path from node 1 to node n in G.
- Let g_{ijk},h_{ijk} be boolean variables.

  $T(g_{ijk})$=**true** iff there is a path in G from node i to node j not using any intermediate node bigger than k.

  $T(h_{ijk})$=**true** iff there is a path in G from node i to node j which **uses k** but no other nodes bigger than k as intermediate nodes.

- All g_{ij0} gates are input gates (there are no h_{ij0} gates).

  In particular, $T(g_{ij0})$=true iff i=j or (i,j) is an edge of G.
- For k=1,2,…n , h_{ijk} is an AND gate, and its predecessors are $g_{i,k,k-1}$ and $g_{k,j,k-1}$ meaning that there is a path in G from node i to node j passing through k and no other bigger than k iff there are paths from i to k and from k to j not using any nodes bigger than k.
- Similarly, g_{ijk} is an OR gate , and its predecessors are $g_{i,j,k-1}$ and h_{ijk} .

Finally, g_{inn} is the output gate. So, we have inductively described the whole circuit R(G).

- Proof: We will use induction on k.

For k=0 the truth values of g_{ijk} are given according to their description.

if this is also true up to k-1 the definitions of h_{ijk} and g_{ijk} guarantee that it to be true for k as well.

So, g_{1nn} (the output) is true iff there is a path from node 1 to n in G.

- Finally, we shall show that the reduction can be computed in O(logn) space. Just like before, the space needed is only for storing the 3 indexes (I,j,k) whose value is no greater than n=|x|. So their binary representation is O(logn) bits long.

- Example 8.3: Reduction of CIRCUIT SAT to SAT
- Given a boolean circuit C, we wish to produce a Boolean expression R(C) such that R(C) is satisfyable iff C is satisfyable.
- R(C) contains a variable "g_i" for each gate of C.
- Depending on the type of the gates, we add the clauses:

  Variable gate: $(\neg g \vee x) \wedge (g \vee \neg x)$

  True gate: (g_i)

  False gate: $(\neg g_i)$

  NOT gate with predecessor gate h: $(\neg g \vee \neg h), (g \vee h)$

  OR gate with predecessors h and h': $(\neg h \vee g) \wedge (\neg h' \vee g) \wedge (h \vee h' \vee \neg g)$

  AND gate with predecessors h and h': $(\neg g \vee h) \wedge (\neg g \vee h')(\neg h \vee \neg h' \vee g)$

  Output gate: (g_i)

- R(C) is satisfiable iff C is satisfiable.
- The reductions uses O(logn) space (it only needs to store the predecessors).

- Example 8.4: Reduction by generalization.
- Problem A is a special case of problem B: the input of A is a subset of the input of B, and for this input A,B give the same answers.
- For example CIRCUIT SAT is a generalization of CIRCUIT VALUE.

- Proposition 8.2: If R is a reduction from language L1 to L2 and R' is a reduction from L2 to L3, then RoR' is a reduction from L1 to L3.
- Proof: It is trivial that: $x \in L_1 \Leftrightarrow R'(R(x)) \in L_3$

  But we have to show that RoR' can be computed using O(logn) space.

  If we were using a string R(x) as the output of M_R and input for $M_{R'}$ the computation could require a polynomial amound of space since the output of a TM can be of the same size as the time of computation.

  Solution: We could only store the cursor position in R(x) in a variable i (logn bits). So, the output symbols of M_R will be generated one by one or the computation of R(x) will be restarted until i is reached, if needed.

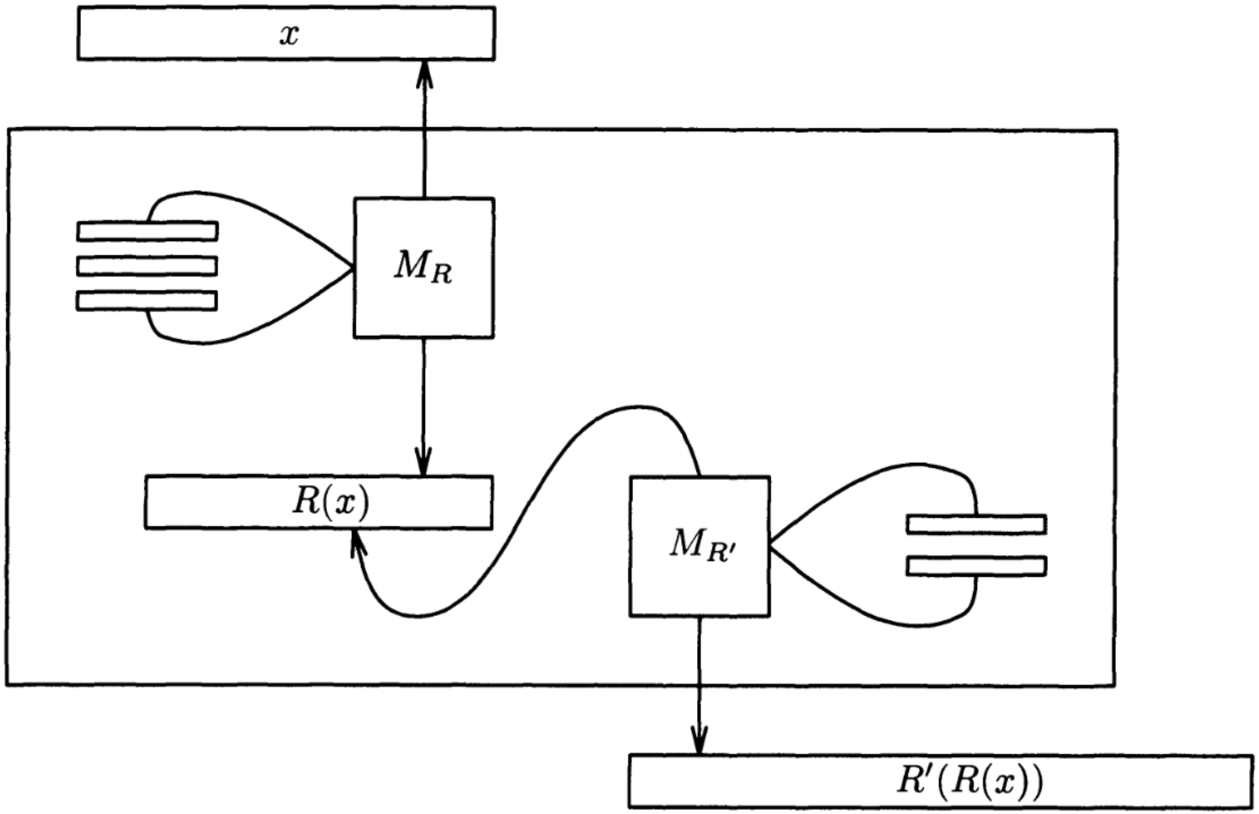


Figure 8-2. How *not* to compose reductions.

- 8.2: Completeness
- Definition 8.2: Let C be a complexity class, and let L be a language in C. We say that L is C-complete if any language L’∈ C can be reduced to L.
- Definition: We say that a class C is closed under reductions if whenever L is reducible to L’ and L’∈ C, then also L ∈ C.
- Proposition 8.3: P, NP, coNP, L, NL, PSPACE, EXP are all closed under reductions.

- Proposition 8.4: If two classes C and C' are both closed under reductions, and there is a language L which is complete for both C and C', then C=C'.
- Proof: L is C-complete $\Rightarrow \forall L' \in C$, L' reduces to L $\in$ C' $\Rightarrow L' \in C' \Rightarrow C \subseteq C'$ (C' is closed under reductions).

  In a similar way: $C' \subseteq C$. So, C=C'.

- The table method:

  Consider a polynomial-time Turing machine $M=(K,\Sigma,\delta,s)$ deciding language L. Its computation is shown in the following

  $|x|^k$ X $|x|^k$ table (where $|x|^k$ is the time bound).
- The (i,j) table entry represents the contents of position j of the string of M at time i.
- Also if an entry has a subscript (which is the symbol of the current state), then this denotes that the cursor at that time is at this position.

$\triangleright$	0_s	1	1	0	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$
$\triangleright$	$\triangleright$	1_{q_0}	1	0	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$
$\triangleright$	$\triangleright$	1	1_{q_0}	0	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$
$\triangleright$	$\triangleright$	1	1	0_{q_0}	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$
$\triangleright$	$\triangleright$	1	1	0	$\sqcup_{q_0}$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$
$\triangleright$	$\triangleright$	1	1	$0_{q'_0}$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$
$\triangleright$	$\triangleright$	1	1_q	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$
$\triangleright$	$\triangleright$	1_q	1	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$
$\triangleright$	$\triangleright_q$	1	1	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$
$\triangleright$	$\triangleright$	1_s	1	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$
$\triangleright$	$\triangleright$	$\triangleright$	1_{q_1}	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$
$\triangleright$	$\triangleright$	$\triangleright$	1	$\sqcup_{q_1}$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$
$\triangleright$	$\triangleright$	$\triangleright$	$1_{q'_1}$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$
$\triangleright$	$\triangleright$	$\triangleright_q$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$
$\triangleright$	$\triangleright$	$\triangleright$	$\sqcup_s$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$
$\triangleright$	$\triangleright$	$\triangleright$	"yes"	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$

Figure 8.3. Computation table.

- Figure 8.3 shows the computation table of a TM deciding palindromes in $O(n^2)$ time, when we put 0110 as input.
- Proposition 8.5: M accepts x iff the computational table of M on input x is accepting.

- Theorem 8.1: CIRCUIT VALUE is P-complete.
- Proof:

CIRCUIT VALUE is in P. So, we have to show that any problem – language $L \in P$ can be reduced to CIRCUIT VALUE.

Equivalently, given an input x and a TM, we have to construct a variable – free cirquit R(x) such that

$x \in L$ iff the output of R(x) is **true**.

Let M: The deterministic Turing machine that decides L in time n^k

T: The computational table of M.

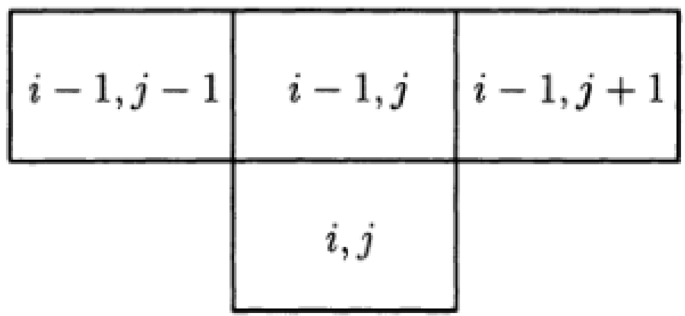
Now, consider some special cases:

T_{0j}=the j-th symbol of x or a "$\sqcup$".

T_{i0}=a "$\triangleright$"

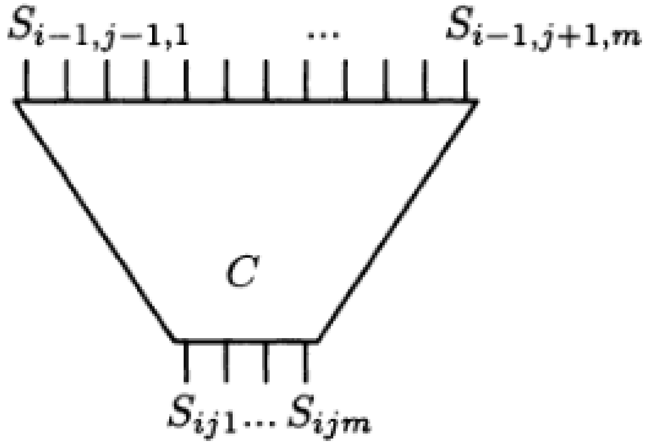
T_{ij}="$\sqcup$" for $j=|x|^k-1$

- For every $0 <= i,j <= |x|^k - 1$, T_{ij} depends only on the entries: $T_{i-1,j-1}$, $T_{i-1,j}$, $T_{i-1,j+1}$ as illustrated below:

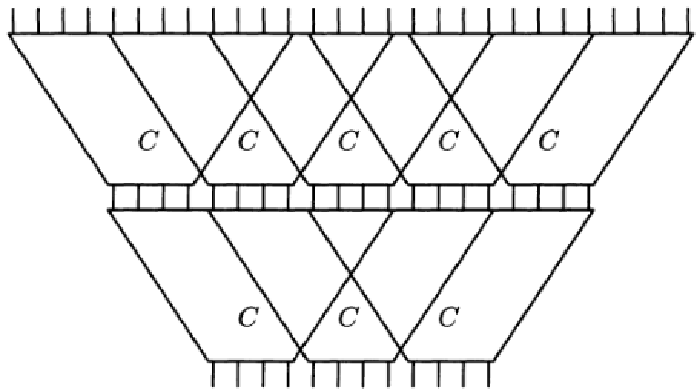


- Now, we encode each symbol $\sigma \in \Gamma$ as a vector of the m-dimensional space: $\{0,1\}^m$, where $m = \lceil \log |\Gamma| \rceil$.

- Let S_{ijl} be the l-th bit of the encoding of T_{ij}. We can see that the value of each of these m bits depends only (through a boolean cirquit C depending only on M) on the 3m bits corresponding to $T_{i-1,j-1}$, $T_{i-1,j}$, $T_{i-1,j+1}$ as shown in the following figure:



- For each x, R(x) will consist of $(|x|^k-1)*(|x|^k-2)$ cirquits (copies of C) connected as illustrated below:



- Input gates of R(x): 1st row and 1st and last column.
- Output gate: The first output of the cirquit $C(|x|^k-1,1)$ (without harming generality).
- Note that we choose the first bit of the encoding of "yes" to be 1, whereas the first bit of the encoding of "no" is 0.

- We are going to prove that R(x) is true iff x ∈L.

  If the value of R(x) is true (1) then the 1st bit of the encoding of the answer is 1.So, the answer is “yes” and so

  M accepts x $\Rightarrow x \in L$. Conversely, if $x \in L$ the answer is “yes” and thus the value of R(x) (1st bit of C(|x|^{k}-1)) is **true**.
- Finally, we have to argue that R can be carried out in O(log|x|) space. This is actually easy, since we can construct every copy of C only using indexes: i,j,l<=|x|, which require log|x| bits to be represented.
- Note that during the reduction, we did not use any NOT gates. This means that CIRQUIT VALUE as well as MONOTONE CIRQUIT VALUE are P-complete.

- Theorem 8.2 (Cook's Theorem): SAT is NP-complete.
- Proof:

SAT $\in$ NP: The verification of a satisfying truth assignment takes polynomial time.

Now, we are going to prove that every problem-language in NP can be reduced to CIRQUIT SAT, which can be then reduced to SAT (example 8.3).

The reduction is similar to the one we made before for the P-completeness of CIRQUIT VALUE, but we have to introduce a few more ideas.

Let L $\in$ NP. There is a non-deterministic Turing machine $M(K,\Sigma,\Delta,s)$ that decides L in time n^k .

- With no loss of generality we can assume that at each step of the computation we have 2 non-deterministic choices. In case we have more, we can make the conversion described below:

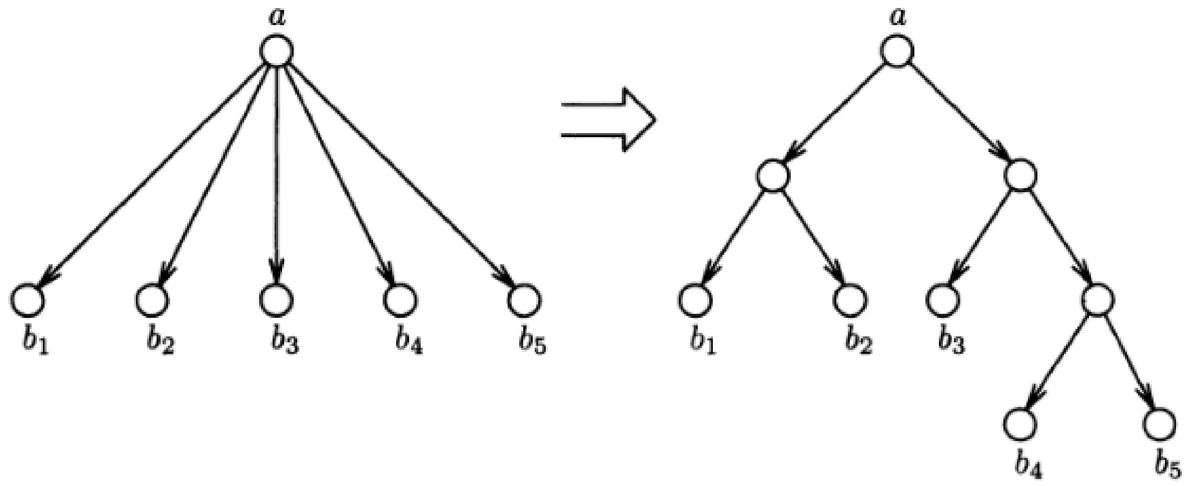
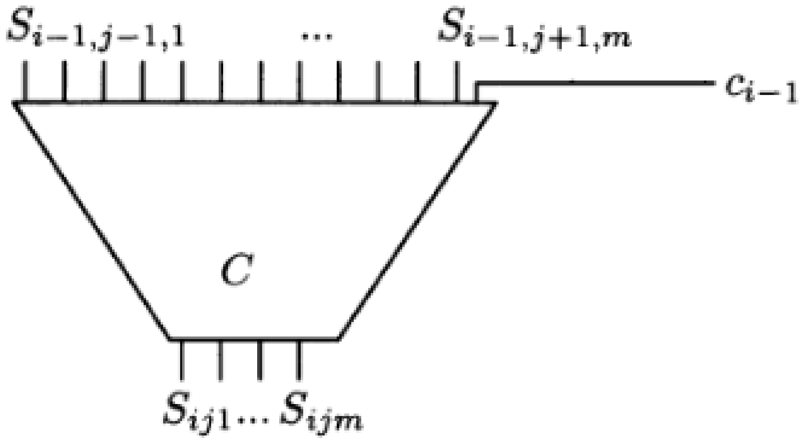


Figure 8-5. Reducing the degree of nondeterminism.

- We make a construction similar to the previous one, and also add an extra bit (c_{i-1}) as input of each boolean cirquit C, corresponding to the non-deterministic choice of the Turing machine as shown below:



- We consider the gates c_i that correspond to the non-deterministic choices, as input variables of the cirquit. So, L was eventually reduced to CIRQUIT SAT. That is, $x \in L$ iff the constructed boolean cirquit has a satisfying truth assignment.
- Finally, it is easy to show that R can be computed using log|x| space in a similar way as the previous one.
- SAT is NP-complete.