ECE 27000 S22 P 15

Topics We Did Not Cover

Combinational Minimization of Multi-Output Functions

Example: Find a minimal realization for
$f_1(x, y, z) = \sum(1, 3, 7)$ and
$f_2(x, y, z) = \sum(2, 6, 7)$

Each output separately:

	x	y	z	
1	0	0	1	√
3	0	1	1	√
7	1	1	1	√

	x	y	z
1,3	0	-	1
3,7	-	1	1

$f_1 = x'z + yz$

	x	y	z	
2	0	1	0	√
6	1	1	0	√
7	1	1	1	√

	x	y	z
2,6	-	1	0
6,7	1	1	-

$f_2 = yz' + xy$

The two functions
$f_1 = x'z + yz$ and
$f_2 = yz' + xy$
require four AND gates and two OR gates with a total of 6 gates and 8 literals.

It is possible to implement the common term xyz only once, and use

$h = xyz$

$f_1 = x'z + h$

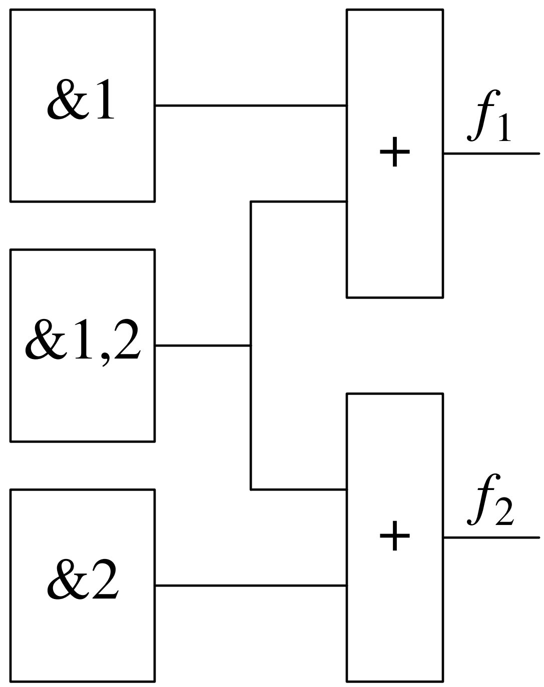
$f_2 = yz' + h$

which has three AND gates and two OR gates with a total of 5 gates and 7 literals.

The second implementation is more efficient.

xyz is a prime implicant of $f_1 \cdot f_2 = \Sigma(7)$.
It is not a prime implicant of either f_1 or f_2.

A general realization of a two-output function:



& 1 - AND gates driving only f_1. These should correspond to prime implicants of f_1.

& 2 - AND gates driving only f_2. These should correspond to prime implicants of f_2.

& 1,2 - AND gates driving both f_1 and f_2. These should correspond to prime implicants of $f_1 \cdot f_2$.

To design a minimal sum for a multi-output function with functions $f_1, f_2, \cdots, f_m$ we need the prime implicants of

$$f_1, f_2, \cdots, f_m,$$
$$f_1 \cdot f_2, f_1 \cdot f_3, \cdots, f_{m-1} \cdot f_m,$$
$$\cdots,$$
$$f_1 \cdot f_2 \cdot \cdots \cdot f_m.$$

This collection of prime implicants is called the *multiple-output prime implicants*.

A prime implicant map that includes all the multiple-output prime implicants can be used for selecting equations for a minimal multiple-output circuit.

Example:

$f_1(x, y, z) = \sum(1, 3, 7)$ and

$f_2(x, y, z) = \sum(2, 6, 7)$

$f_1 \cdot f_2 = \sum(7)$.

	001	011	111	010	110	111
0-1	x	x				
-11		x	x			
-10				x	x	
11-					x	x
111			x			x

Essential prime implicants:

f_1: 0-1

f_2: -10

Use 111 to cover the remaining minterm

Combinational Multilevel Minimization

Example:

$f = stv + stwx + styz + uv + uwx + uyz$

A two-level realization requires 6 AND gates and one OR gate.

The total number of gate inputs is

$3 + 4 + 4 + 2 + 3 + 3 + 6 = 25$

A multilevel realization:

$$f = stv + stwx + styz + uv + uwx + uyz$$
$$= st(v + wx + yz) + u(v + wx + yz)$$
$$= (st + u)(v + wx + yz)$$

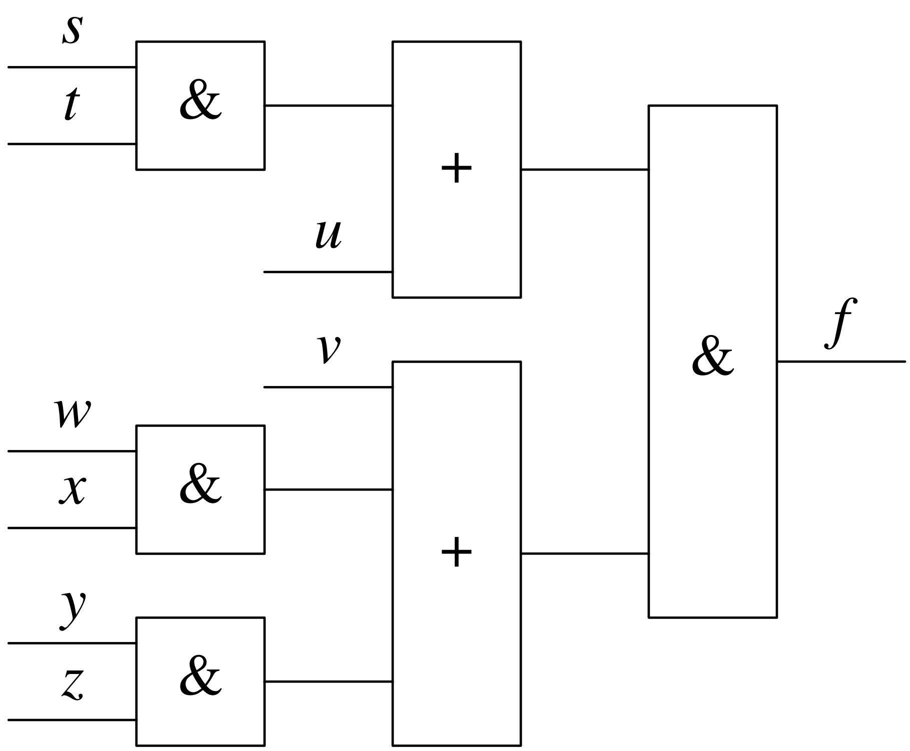
This process is called *factoring*.

The last expression corresponds to three levels of gates.

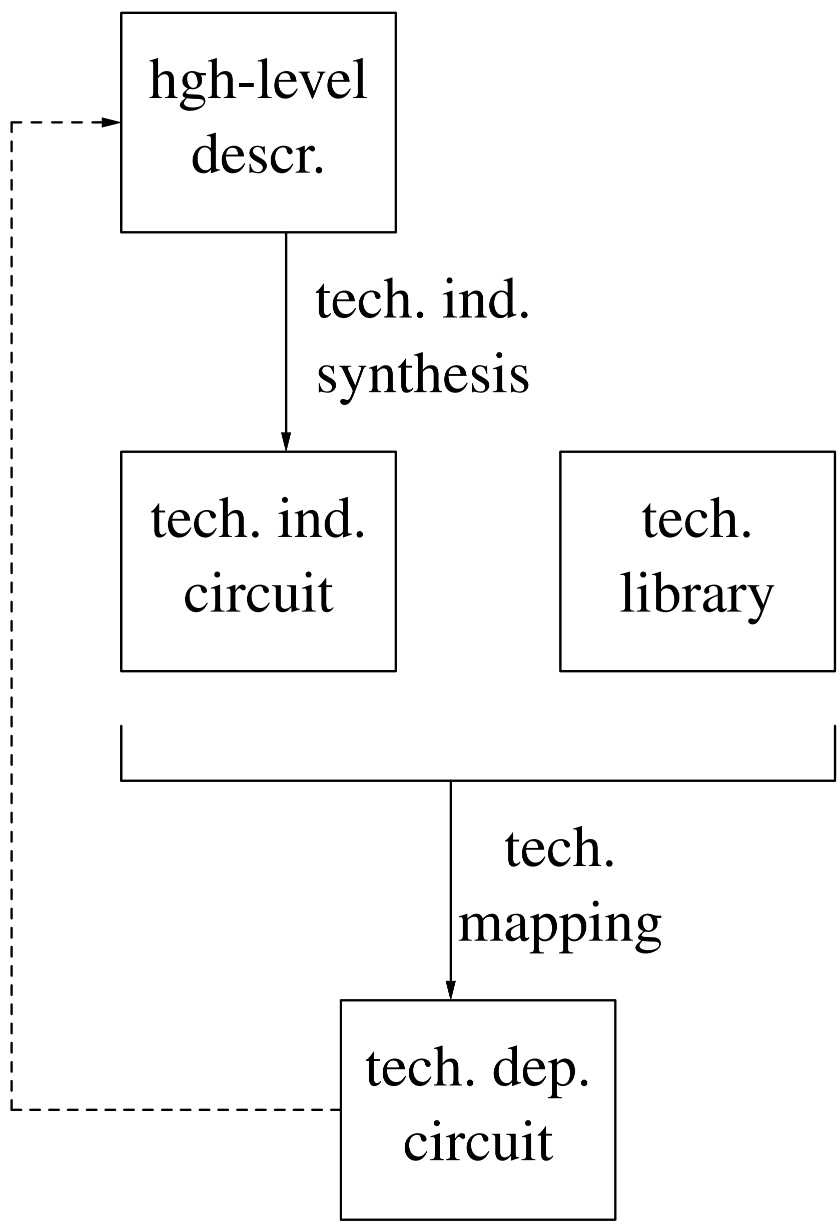
It requires 4 AND gates and 2 OR gates.

The total number of gate inputs is

$2 + 2 + 2 + 2 + 2 + 3 = 13.$


s
t
&
+
u
v
w
x
&
y
z
&
+
&
f

Technology Mapping



Special Functions

Many useful functions have properties that can be used to facilitate their representation and realization.

Certain properties are useful in applications such as logic verification, technology mapping, etc..

Symmetric Functions

Symmetry refers to the ability to interchange variables without changing the value of the function.

Definition: A function $f(x_1, x_2, \cdots, x_n)$ is *totally symmetric* iff it is unchanged by any permutation of its variables.

Example: The three-variable odd parity function

x	y	z	f
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	0
0	0	1	1
0	1	0	1
1	0	0	1
1	1	1	1

Definition: A function $f(x_1, x_2, \cdots, x_n)$ is *partially symmetric* in $x_{i1}, x_{i2}, \cdots, x_{im}$ iff it is unchanged by any permutation of the variables $x_{i1}, x_{i2}, \cdots, x_{im}$.

Unate Functions

Definition: A function $f(x_1, x_2, \cdots, x_n)$ is said to be *positive* in x_i if it is possible to write a sum-of-products expression for f in which x_i' does not appear.

$f_1(w, x, y, z) = w + xy + xz$ is positive in w, x, y and z.
$f_2(w, x, y, z) = w + xy + xz'$ is positive in w, x and y.
$f_3(w, x, y, z) = wx + wy + x'z + y'z'$ is positive in w.

Changing the value of a positive variable from 0 to 1 can only change the value of the function from 0 to 1.

Changing the value of a positive variable from 1 to 0 can only change the value of the function from 1 to 0.

We obtain an equivalent definition:
A function $f(x_1, x_2, \cdots, x_n)$ is *positive* in x_1 if and only if
$f(0, x_2, \cdots, x_n) \leq f(1, x_2, \cdots, x_n)$.
$f_{x_1'} \leq f_{x_1}$.

Example:
$f_3(w, x, y, z) = wx + wy + x'z + y'z'$
$f_3(0, x, y, z) = x'z + y'z'$
$f_3(1, x, y, z) = x + y + x'z + y'z'$

Definition: A function $f(x_1, x_2, \cdots, x_n)$ is said to be *negative* in x_i if it is possible to write a sum-of-products expression for f in which x_i does not appear.

$f_2(w, x, y, z) = w + xy + xz'$ is negative in z.

Threshold Functions

The value of a threshold function is 1 if the arithmetic sum of its inputs is not smaller than a bound called the threshold.

Example: The 5-variable majority function is a threshold function with threshold 3.
If three or more of the variables are 1, then the function is 1.

Example: An OR gate is a threshold function with a threshold of 1.

Example: A k-input AND gate is a threshold function with a threshold of k.

Definition: A *threshold function* is a function that can be defined by an inequality
$f(X) = 1$ iff $a_1 x_1 + a_2 x_2 + \cdots + a_n x_n \geq T$
The a_i are called weights.
T is the threshold value.
$+$ is an arithmetic sum.

Example:
$a_1 = a_2 = 1$, $a_i = 0$ for $i \neq 1, 2$, $T = 2$
$f = 1$ iff $x_1 + x_2 \geq 2$
$f = x_1 x_2$

$a_1 = a_2 = a_3 = 1$, $a_i = 0$ for $i \neq 1, 2, 3$, $T = 2$
$f = 1$ iff $x_1 + x_2 + x_3 \geq 2$
$f = x_1 x_2 + x_1 x_3 + x_2 x_3$

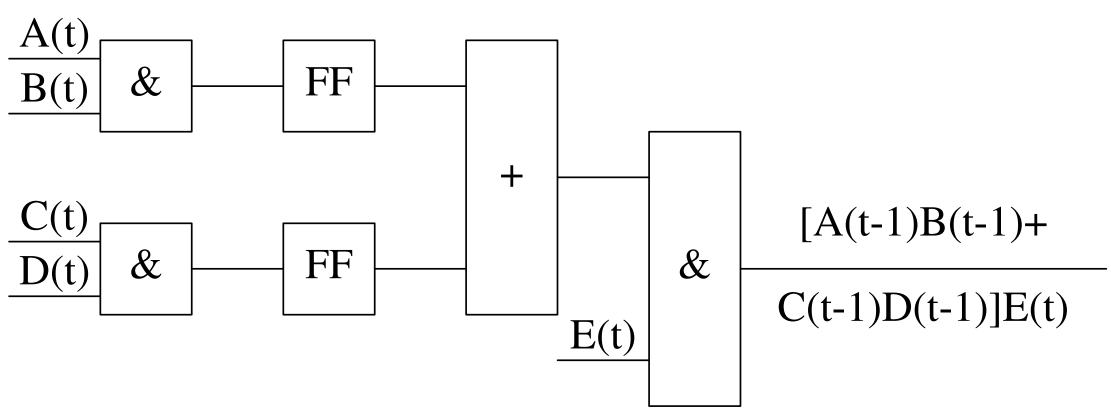
State Assignment for Finite-State Machines

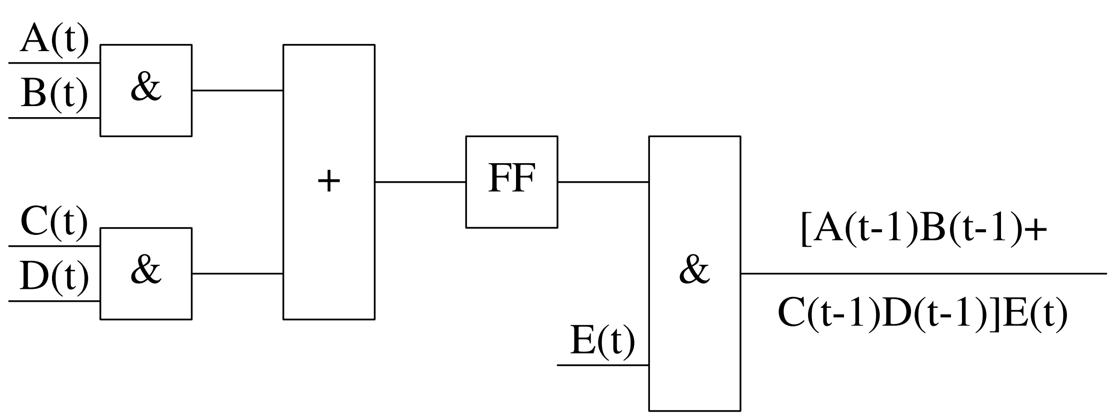
Decomposition of Finite-State Machines

State Minimization for Incompletely-Specified Machines

Retiming of Synchronous Sequential Circuits

Retiming is a circuit transformation that removes flip-flops from certain locations and adds flip-flops at other locations so as to preserve the functionality of the circuit, and reduce a cost function associated with the circuit.





Input-Output Experiments for Finite-State Machines

In input-output experiments, a machine is available as a "black box", with no access to internal components.
The machine is studied based on its output response to input sequences supplied to it.

Assumptions: the machine is reduced, strongly-connected and completely specified.

At the beginning of an experiment, the machine is in an *initial state*.

At the end of an experiment, the machine is in a *final state*.

There are two types of experiments according to the number of copies of the machine available:

Simple experiments - performed on a single copy of the machine.

Multiple experiments - performed on two or more identical copies of the machine.

There are two types of experiments according to the way they are performed:

Adaptive experiments - the input at any time depends on the previous outputs.

Preset experiments - the entire input sequence is predetermined independent of the output of the experiment.

Synchronizing Experiments

A *synchronizing sequence* of a machine M is a sequence that takes M to a specified final state, regardless of the output or the initial state.

A synchronizing sequence can be used to bring the machine to a state from which its normal operation can begin.

Example: A sequence detector for the sequence 0101.

	NS, z	
PS	$x = 0$	$x = 1$
A	$B, 0$	$A, 0$
B	$B, 0$	$C, 0$
C	$D, 0$	$A, 0$
D	$B, 0$	$C, 1$

The machine must start from state A to operate correctly.
Before applying the input sequence in which 0101 should be detected, we must bring the machine to state A.

To find a synchronizing sequence, we construct a *synchronizing tree*.

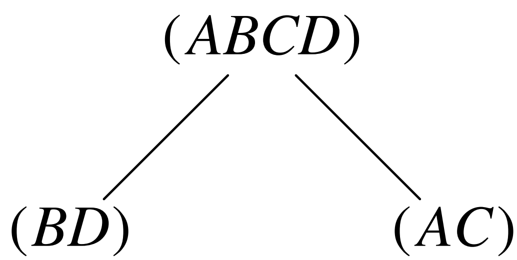
At the beginning of the sequence, the machine can be in either one of the states ($ABCD$).
This is called the *initial uncertainty*.

We can now apply one of two input symbols:

After applying $x = 0$, the machine can be in one of the states ($BBDB$) or simply (BD), depending on its initial state.

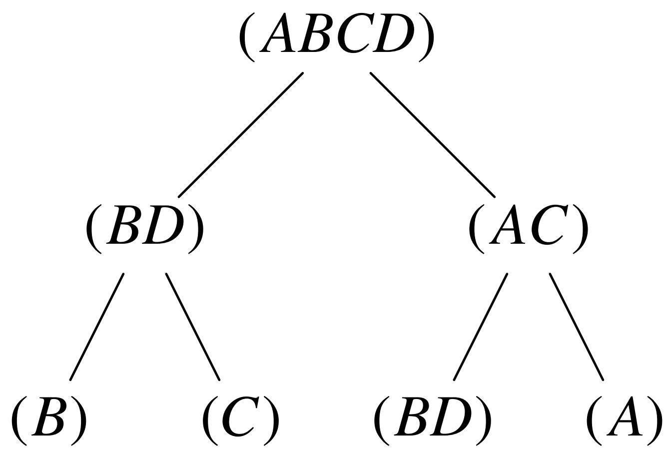
After applying $x = 1$, the machine can be in one of the states ($ACAC$) or simply (AC), depending on its initial state.

This information is represented in a tree, as follows:



The left branches represent $x = 0$ and the right branches represent $x = 1$.

Starting from each new subset we obtained, we can continue to apply $x = 0$ and $x = 1$.
We obtain:



The subsets (B), (C) and (A) indicate synchronizing sequences 00, 01 and 11, respectively.

Verifying that 00 is a synchronizing sequence into state B:

$A \xrightarrow{0} B \xrightarrow{0} B$

$B \xrightarrow{0} B \xrightarrow{0} B$

$C \xrightarrow{0} D \xrightarrow{0} B$

$D \xrightarrow{0} B \xrightarrow{0} B$

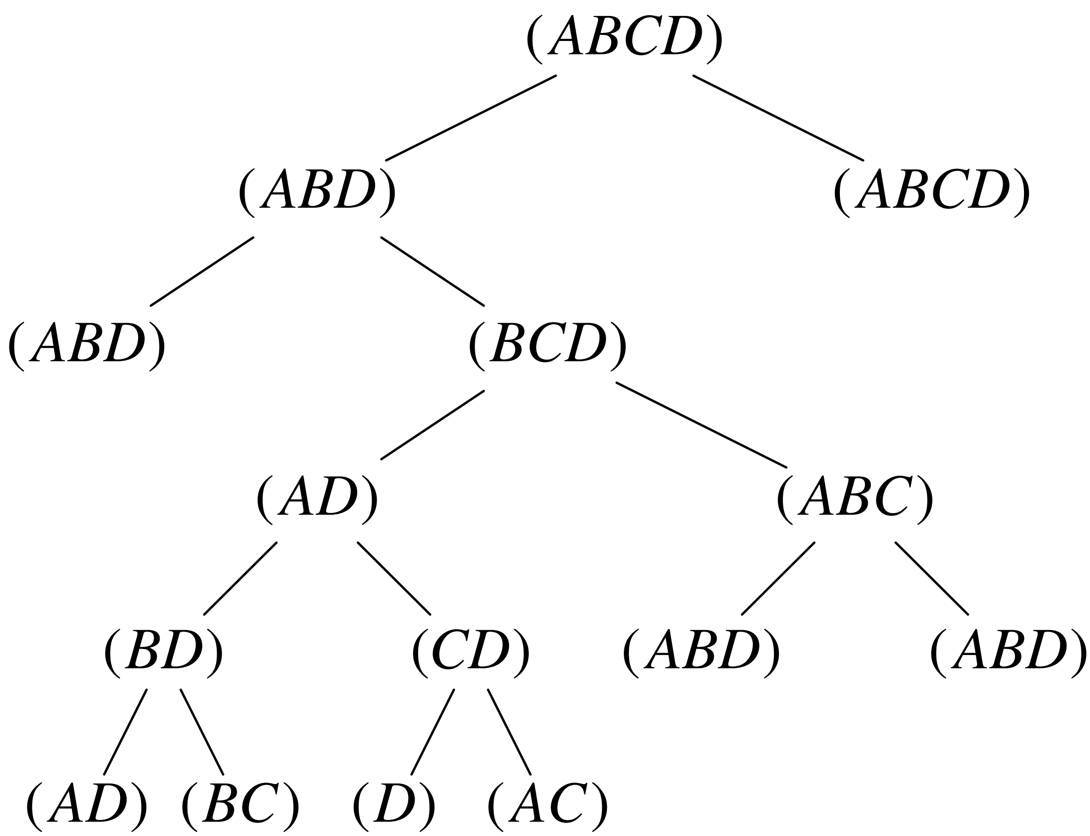
11 is a synchronizing sequence into state A.

Even if we had not obtained synchronizing sequences, there is no need to continue with the set (BD), since it was already obtained before.

Example:

	NS, z	
PS	$x = 0$	$x = 1$
A	$B, 0$	$D, 0$
B	$A, 0$	$B, 0$
C	$D, 1$	$A, 0$
D	$D, 1$	$C, 0$

A synchronizing tree:



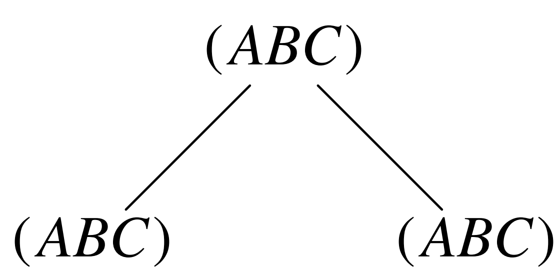
The synchronizing sequence: 01010.

Some machines do not have synchronizing sequences.
In such a case, it should be possible to reset the flip-flops to bring the machine to its initial state.

Example:

	NS, z	
PS	$x = 0$	$x = 1$
A	$B, 0$	$A, 0$
B	$C, 1$	$C, 0$
C	$A, 1$	$B, 0$

A synchronizing tree:



Theorem: If a synchronizing sequence exists for an n-state machine, then its length is at most $(n-1)^2 n/2$.

Proof: Let the initial uncertainty be
$U_0 = (S_1 S_2 \cdots S_n)$.

Select any pair of states $S_i S_j$ and apply to them a sequence t_1 that brings them into a state S_k.
t_1 exists since M has a synchronizing sequence.

The length of t_1 is as follows. Starting from $S_i S_j$, we may have to traverse all pairs of states before reaching a pair with a next state S_k.
The number of all state pairs is $n(n-1)/2$. Therefore, the length of t_1 is at most $n(n-1)/2$.

Apply t_1 to the initial uncertainty to obtain an uncertainty U_1 that contains at most $n-1$ states (since S_i and S_j were replaced by a single state S_k).

Select a pair of states in U_1 and apply to them a sequence t_2 that takes them to a single state. The length of t_2 is also $n(n-1)/2$.
Apply t_2 to U_1 and obtain U_2 that has at most $n-2$ states.

After using at most $n-1$ sequences t_i, U_{n-1} contains a single state and the machine is synchronized.

The total length of $t_1, t_2 \cdots$ is at most
$(n-1)(n(n-1)/2) = (n-1)^2 n/2$.

Homing Experiments

An input sequence Y_0 is said to be a *homing sequence* if the final state of the machine can be determined uniquely from the output response of the machine to Y_0, regardless of the initial state.

A homing sequence differs from a synchronizing sequence in that the final state for a homing sequence is determined based on the output sequence.

Example: The following machine does not have a synchronizing sequence.

	NS, z	
PS	$x = 0$	$x = 1$
A	$B, 0$	$A, 0$
B	$C, 1$	$C, 0$
C	$A, 1$	$B, 0$

00 is a homing sequence:

$$A \overset{0}{\underset{0}{\rightarrow}} B \overset{0}{\underset{1}{\rightarrow}} C$$

$$B \overset{0}{\underset{1}{\rightarrow}} C \overset{0}{\underset{1}{\rightarrow}} A$$

$$C \overset{0}{\underset{1}{\rightarrow}} A \overset{0}{\underset{0}{\rightarrow}} B$$

initial state	response to 00	final state
A	01	C
B	11	A
C	10	B

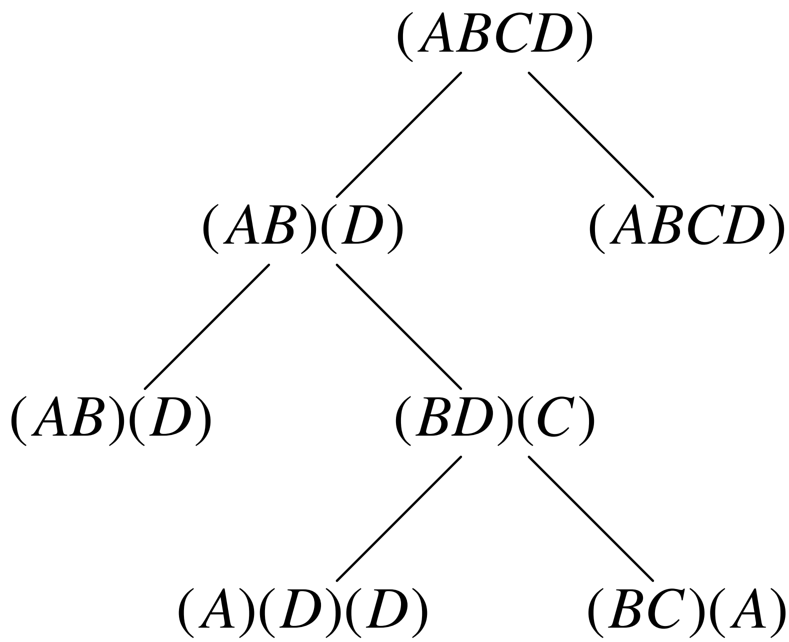
Finding Homing Sequences

Based on a homing tree. Similar to a synchronizing tree, except that here, states are placed in separate groups if they produce different outputs.

Example:

	NS, z	
PS	$x = 0$	$x = 1$
A	$B, 0$	$D, 0$
B	$A, 0$	$B, 0$
C	$D, 1$	$A, 0$
D	$D, 1$	$C, 0$

A homing tree:



A homing sequence: 010

Theorem: Every reduced n-state machine has a preset homing sequence whose length is at most $(n-1)^2$.

Proof: Let the initial uncertainty be $(S_1 S_2 \cdots S_n)$.

Since M is reduced, for every pair of states (S_i, S_j) there is a pairwise distinguishing sequence of length at most $n-1$. Denote the pairwise distinguishing sequences by $\gamma_1, \gamma_2, \cdots$.

Starting from the initial uncertainty, apply γ_1. This results in an uncertainty vector that has at least two components.

Select any two states in one component and apply γ_2 that distinguishes them. This results in an uncertainty vector that has at least three components.

Continue in the same way. After applying $\gamma_1\gamma_2 \cdots \gamma_{n-1}$, we obtain an uncertainty vector with n components, each containing a single state.

Therefore, the sequence $\gamma_1\gamma_2 \cdots \gamma_{n-1}$ is a homing sequence. Its length is at most $(n-1)^2$.

Distinguishing Experiments

An input sequence X_0 is a *distinguishing sequence* if the output sequence produced by M in response to X_0 is different for each initial state.

Example:

	NS, z	
PS	$x = 0$	$x = 1$
A	$C, 0$	$D, 1$
B	$C, 0$	$A, 1$
C	$A, 1$	$B, 0$
D	$B, 0$	$C, 1$

Responses to 100:

$$A \overset{1}{\underset{1}{\rightarrow}} D \overset{0}{\underset{0}{\rightarrow}} B \overset{0}{\underset{0}{\rightarrow}} C$$

$$B \overset{1}{\underset{1}{\rightarrow}} A \overset{0}{\underset{0}{\rightarrow}} C \overset{0}{\underset{1}{\rightarrow}} A$$

$$C \overset{1}{\underset{0}{\rightarrow}} B \overset{0}{\underset{0}{\rightarrow}} C \overset{0}{\underset{1}{\rightarrow}} A$$

$$D \overset{1}{\underset{1}{\rightarrow}} C \overset{0}{\underset{1}{\rightarrow}} A \overset{0}{\underset{0}{\rightarrow}} C$$

initial state	response to 100	final state
A	100	C
B	101	A
C	001	A
D	110	C

Output sequence 100 implies initial state A
Output sequence 101 implies initial state B
Output sequence 001 implies initial state C
Output sequence 110 implies initial state D

Every distinguishing sequence is also a homing sequence:
If X_0 is a distinguishing sequence, then we can determine the initial state based on X_0 and the output response.
Once the initial state is known, we can uniquely determine the final state.
Therefore, X_0 is a homing sequence.

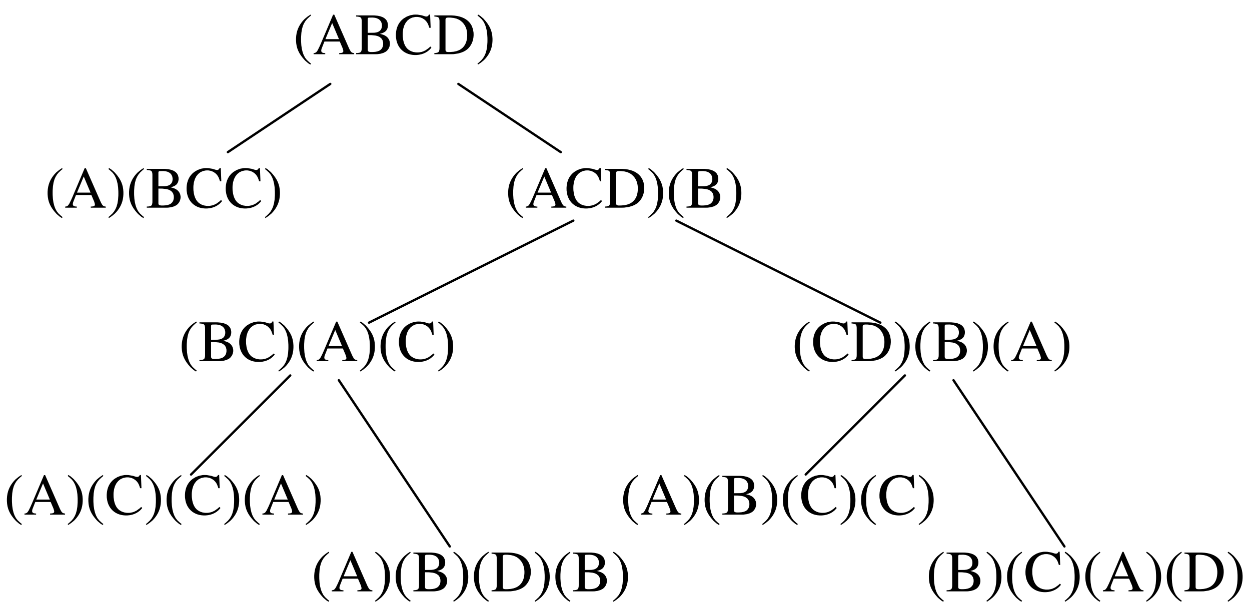
Finding a Distinguishing Sequence

Based on a distinguishing tree. Same as a homing tree, except that here, we stop if two identical states are placed in the same block.

Example:

	NS, z	
PS	$x = 0$	$x = 1$
A	$C, 0$	$D, 1$
B	$C, 0$	$A, 1$
C	$A, 1$	$B, 0$
D	$B, 0$	$C, 1$

A distinguishing tree:



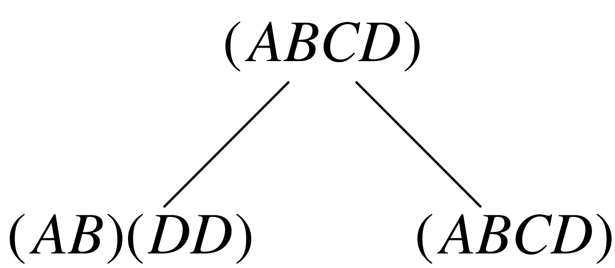
Distinguishing sequences:
100, 101, 110, 111

There are machines that do not have distinguishing sequences.

Example:

	NS, z	
PS	$x = 0$	$x = 1$
A	$B, 0$	$D, 0$
B	$A, 0$	$B, 0$
C	$D, 1$	$A, 0$
D	$D, 1$	$C, 0$

A distinguishing tree:



Adaptive Distinguishing Sequences

Example:

	NS, z	
PS	$x = 0$	$x = 1$
A	$C, 0$	$A, 1$
B	$D, 0$	$C, 1$
C	$B, 1$	$D, 1$
D	$C, 1$	$A, 0$

Initial uncertainty ($ABCD$)
Apply $x = 0$
If $z = 0$: the new uncertainty is (CD)
If $z = 1$: the new uncertainty is (BC)

Uncertainty (CD)
Apply $x = 1$
If $z = 0$: the new uncertainty is (A)
If $z = 1$: the new uncertainty is (D)

Uncertainty (BC)
Apply $x = 0$
If $z = 0$: the new uncertainty is (D)
If $z = 1$: the new uncertainty is (B)

Adaptive distinguishing sequences are computed based on an adaptive tree.

Develop uncertainties under different output values separately.

Make sure that for every uncertainty corresponding to a selected input value there is a sequence that resolves it.

Machine Identification

Goal - determine the state table of an unknown machine based on input-output experiments.

Application - verification and testing. Ensure that a given synthesized machine corresponds to its state-table specification.

To be able to solve the problem, we need to know:
All the input symbols of the machine.
An upper bound on the number of states.

In addition, the machine has to be:
Strongly-connected so that we can explore all its states.
Reduced so that each state can be uniquely identified.

Example: A single-input two-state machine produced the following output sequence in response to the given input sequence. Find the state table of the machine.

input:	1	1	1	0	1	0	1
output:	0	1	0	0	1	0	0

There are two different responses to 1: 0 and 1. Therefore, the machine must be in two different states when 1 is applied. Let us call them A and B, respectively.

	NS, z	
PS	$x = 0$	$x = 1$
A	?, ?	?, 0
B	?, ?	?, 1

Every time there is a 0 in response to 1, the machine is in state A.
Every time there is a 1 in response to 1, the machine is in state B.

Assigning the states along the given sequence:

input:		1		1		1		0		1		0		1
state:	A		B		A				B				A	
output:		0		1		0		0		1		0		0

We can deduce the transitions

$A \underset{0}{\overset{1}{\rightarrow}} B$ and

$B \underset{1}{\overset{1}{\rightarrow}} A$.

	NS, z	
PS	$x = 0$	$x = 1$
A	$?, ?$	$B, 0$
B	$?, ?$	$A, 1$

Assigning the states along the given sequence:

input:		1	1	1	0	1	0	1
state:	A	B	A	B	B	A	A	
output:		0	1	0	0	1	0	0

We can deduce the other transitions,

$A \underset{0}{\overset{0}{\rightarrow}} A$ and

$B \underset{0}{\overset{0}{\rightarrow}} B.$

The complete machine:

PS	NS, z $x = 0$	$x = 1$
A	$A, 0$	$B, 0$
B	$B, 0$	$A, 1$

Another example: A single-input four-state machine produced the following output sequence in response to the given input sequence. Find the state table of the machine.

input	state	output
0		0
1		0
0		1
1		1
0		1
1		0
0		0
1		1
0		1
1		0
0		0
0		1
1		0
1		0
0		1
1		1
0		1
0		1
1		1
0		1
0		1
0		0
1		0
0		1
0		0
0		0
1		1

First identify four states.

responses to 01	responses to 10
00	01
11	11
10	00
01	

Use 01 to identify the states:

state	responses to 01
A	00
B	11
C	10
D	01

Assigning these responses into the state sequence:

input	state	output
	A	
0		0
1		0
	B	
0		1
1		1
	C	
0		1
1		0
	D	
0		0
1		1
	C	
0		1
1		0
0		0
	C	
0		1
1		0
1		0
	B	
0		1
1		1
0		1
	B	
0		1
1		1
0		1
0		1
	A	
0		0
1		0
0		1
0		0
	D	
0		0
1		1

From the state sequence we can conclude the following.

$A \underset{00}{\overset{01}{\rightarrow}} B$

$B \underset{11}{\overset{01}{\rightarrow}} C$

$C \underset{10}{\overset{01}{\rightarrow}} D$

$D \underset{01}{\overset{01}{\rightarrow}} C$

Assign back into the state sequence:

input	state	output
	A	
0		0
1		0
	B	
0		1
1		1
	C	
0		1
1		0
	D	
0		0
1		1
	C	
0		1
1		0
	D	
0		0
	C	
0		1
1		0
1		0
	B	
0		1
1		1
	C	
0		1
	B	
0		1
1		1
0		1
0		1
	A	
0		0
1		0
	B	
0		1
0		0
	D	
0		0
1		1

From the state sequence we can conclude the following.

$$D \underset{0}{\overset{0}{\rightarrow}} C$$

$$C \underset{1}{\overset{0}{\rightarrow}} B$$

The outputs in response to 0

	NS, z	
PS	$x = 0$	$x = 1$
A	$?, 0$	$?, ?$
B	$?, 1$	$?, ?$
C	$B, 1$	$?, ?$
D	$C, 0$	$?, ?$

Assigning back to the state sequence:

input	state	output
	A	
0		0
1		0
	B	
0		1
1		1
	C	
0		1
	B	
1		0
	D	
0		0
	C	
1		1
	C	
0		1
	B	
1		0
	D	
0		0
	C	
0		1
	B	
1		0
1		0
	B	
0		1
1		1
	C	
0		1
	B	
0		1
1		1
0		1
0		1
	A	
0		0
1		0
	B	
0		1
0		0
	D	
0		0
	C	
1		1

	NS, z	
PS	$x = 0$	$x = 1$
A	$?, 0$	$?, ?$
B	$?, 1$	$D, 0$
C	$B, 1$	$C, 1$
D	$C, 0$	$?, ?$

Assign back to the state sequence:

input	state	output
	A	
0		0
1		0
	B	
0		1
1		1
	C	
0		1
	B	
1		0
	D	
0		0
	C	
1		1
	C	
0		1
	B	
1		0
	D	
0		0
	C	
0		1
	B	
1		0
	D	
1		0
	B	
0		1
	*	
1		1
	C	
0		1
	B	
0		1
1		1
0		1
0		1
	A	
0		0
1		0
	B	
0		1
	X	
0		0
	D	
0		0
	C	
1		1

PS	NS, z $x = 0$	$x = 1$
A	$?, 0$	$?, ?$
B	$?, 1$	$D, 0$
C	$B, 1$	$C, 1$
D	$C, 0$	$B, 0$

No new assignments into the state sequence.

Let us use the sequence 10.

state	responses to 10
A	?
B	00
C	11
D	01

The state marked *:
Its response to 10 is 11.
Therefore, it can be either C or A.

Assume C: Then we obtain $C \underset{1}{\overset{1}{\rightarrow}} C$ and $B \underset{1}{\overset{0}{\rightarrow}} C$.

Assigning $B \underset{1}{\overset{0}{\rightarrow}} C$ in the state marked X, we obtain $C \underset{0}{\overset{0}{\rightarrow}} D$.

However, $C \underset{1}{\overset{0}{\rightarrow}} B$.

A contradiction.

The state marked * must be A.

	NS, z	
PS	$x = 0$	$x = 1$
A	?, ?	$C, 1$
B	$A, 1$	$D, 0$
C	$B, 1$	$C, 1$
D	$C, 0$	$B, 0$

input	state	output
	A	
0		0
1		0
	B	
0		1
	A	
1		1
	C	
0		1
	B	
1		0
	D	
0		0
	C	
1		1
	C	
0		1
	B	
1		0
	D	
0		0
	C	
0		1
	B	
1		0
	D	
1		0
	B	
0		1
	A	
1		1
	C	
0		1
	B	
0		1
	A	
1		1
	C	
0		1
	B	
0		1
	A	
0		0
1		0
	B	
0		1
	A	
0		0
	D	
0		0
	C	
1		1

PS	NS, z $x = 0$	$x = 1$
A	$D, 0$	$C, 1$
B	$A, 1$	$D, 0$
C	$B, 1$	$C, 1$
D	$C, 0$	$B, 0$