State Minimization: Completely Specified Machines

STGs may contain redundant states, i.e. states whose function can be accomplished by other states.

State minimization is the transformation of a given machine into an *equivalent* machine with no redundant states.

State Minimization: Completely Specified Machines

Two states, s_i and s_j of machine M are *distinguishable* if and only if there exists a finite input sequence which when applied to M causes different output sequences depending on whether M started in s_i or s_j.

Such a sequence is called a *distinguishing sequence* for (s_i, s_j). If there exists a distinguishing sequence of length k for (s_i, s_j), they are said to be k- *distinguishable*.

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

Example:

- states A and B are 1-distinguishable, since a 1 input applied to A yields an output 1, versus an output 0 from B.

- states A and E are 3-distinguishable, since input sequence 111 applied to A yields output 100, versus an output 101 from B.

State Minimization: Completely Specified Machines

States s_i and s_j are said to be **equivalent** iff no distinguishing sequence exists for (s_i, s_j).

If s_i is equivalent to s_j and s_j is equivalent to s_k, then s_i is equivalent to s_k. So state equivalence is an **equivalence relation** (i.e. it is a reflexive, symmetric and transitive relation).

An equivalence relation partitions the elements of a set into **equivalence classes**.

Property. If s_i and s_j are equivalent states, their corresponding X-successors, for all inputs X, are also equivalent.

Procedure: Partition states of M so that two states are in the same equivalence class iff they are equivalent.

State Minimization: Completely Specified Machines

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

P_i: partition using distinguishing sequences of length i.

Partition	Distinguishing Sequence
$P_0 =$ (A B C D E F)	
$P_1 =$ (A C E) (B D F)	$x = 1$
$P_2 =$ (A C E) (B D) (F)	$x = 1; x = 1$
$P_3 =$ (A C) (E) (B D) (F)	$x = 1; x = 1; x = 1$
$P_4 =$ (A C) (E) (B D) (F)	

Algorithm terminates when $P_k = P_{k+1}$.

State Minimization: Completely Specified Machines

Outline of state minimization procedure:

- All states equivalent to each other form an equivalence class. All the states in an equivalence class may be combined into one state in the reduced (quotient) machine.

- These equivalence classes form a partition of the set of states.

- Start with all states in a partition of a single block. Iteratively refine this partition by separating the 1-distinguishable states, 2-distinguishable states and so on.

- In general, when obtaining P_{k+1} from P_k, place in the same block of P_{k+1} the states that are $(k+1)$-equivalent, and in different blocks states that are $(k+1)$-distinguishable.

State Minimization: Completely Specified Machines

Theorem. The equivalence partition is unique.

Theorem. If two states, s_i and s_j, of machine M are distinguishable, then they are distinguishable by a sequence of length $n-1$ or less, where n is the number of states in M.

Definition: Two machines, M_1 and M_2, are said to be *equivalent* iff, for every state in M_1 there is a corresponding equivalent state in M_2 and vice versa.

Theorem. For every machine M there is a minimal machine M_{red} that is equivalent to M and is unique up to isomorphism.

Reduced machine obtained from previous example:

PS	*NS, z*	
	$x = 0$	$x = 1$
α	β, 0	γ, 1
β	α, 0	δ, 1
γ	δ, 0	γ, 0
δ	γ, 0	α, 0

State Minimization of CSMs: Complexity

Algorithm DFA $\rightsquigarrow$ DFA$_{min}$
Input: A finite automaton $M = (Q, \Sigma, \delta, q_0, F)$ with no unreachable states.
Output: A minimal finite automaton $M' = (Q', \Sigma, \delta', q'_0, F')$.
Method:

1. $t := 2; Q_0 :=$ { undefined }; $Q_1 := F; Q_2 := Q \setminus F$.

2. **while** there is $0 < i \leq t$, $a \in \Sigma$ with $\delta(Q_i, a) \nsubseteq Q_j$, for all $j \leq t$
   do

   (a) Choose such an i, $a \in \Sigma$, and $j \leq t$ with $\delta(Q_i, a) \cap Q_j \neq \emptyset$.

   (b) $Q_{t+1} := \{q \in Q_i \mid \delta(q, a) \in Q_j\}$;
   $Q_i := Q_i \setminus Q_{t+1}$;
   $t := t + 1$.

   do.

3. (* Let $[q]$ denote the equivalence class the state q is in and $\{Q_i\}$ denote the set of all equivalence classes. *)
   $Q' := \{Q_1, Q_2, \ldots, Q_t\}$.
   $q'_0 := [q_0]$.
   $F' := \{[q] \in Q' | q \in F\}$.
   $\delta'([q], a) := [\delta(q, a)]$ for all $q \in Q$, $a \in \Sigma$.

State Minimization of CSMs: Complexity

Standard Implementation: $O(kn^2)$, where $n = |Q|$ and $k = |\Sigma|$

Modification of the body of the while loop:

1. Choose such an i, $a \in \Sigma$, and choose $j_1, j2 \leq t$ with $j_1 \neq j_2$, $\delta(Q_i, a) \cap Q_{j_1} \neq \emptyset$, and $\delta(Q_i, a) \cap Q_{j_2} \neq \emptyset$.

2. **If** $|\{q \in Q_i \mid \delta(q,a) \in Q_{j_1}\}| \leq |\{q \in Q_i \mid \delta(q,a) \in Q_{j_2}\}|$
   then $Q_{t+1} := \{q \in Q_i \mid \delta(q,a) \in Q_{j_1}\}$
   else $Q_{t+1} := \{q \in Q_i \mid \delta(q,a) \in Q_{j_2}\}$ **fi**;
   $Q_i := Q_i \setminus Q_{t+1}$;
   $t := t + 1$.

Note: $|Q_{t+1}| \leq 1/2|Q_i|$. Therefore, for all $q \in Q$ the name of the class which q contains changes at most $\log n$ times.

Goal: Develop an implementation such that all work can be assigned to transitions containing a state for whcih the name of the corresponding class is changed.

Suitable data structures achieve an $O(kn \log n)$ implementation
Details in N. Blum IPL '96 [Original $O(kn \log n)$ algorithm in Hopcroft 1971]

State Minimization of CSMs: BDD Implementation

$$
\begin{aligned}
E_0(x,y) &= \prod_{i=1}^{|S|}(x_i \sim y_i) \\
E_{j+1}(x,y) &= E_j(x,y) \wedge \\
&\quad \forall i \exists (o,z,w)[T(x,i,z,o) \wedge \\
&\quad T(y,i,w,o) \wedge E_j(z,w)]
\end{aligned}
$$

State Minimization: Incompletely Specified Machines

Statement of the problem: given an incompletely specified machine M, find a machine M' such that:

- on any input sequence, M' produces the same outputs as M, whenever M is specified.
- no machine M'' with fewer states than M' does the job.

State Minimization: Incompletely Specified Machines

Machine M:

PS	*NS*, *z*	
	$x = 0$	$x = 1$
s1	s3, 0	s2, 0
s2	s2, -	s3, 0
s3	s3, 1	s2, 0

- Attempt to reduce this case to usual state minimization of completely specified machines.
- Force the don't cares to all their possible values and choose the smallest of the completely specified machines so obtained.

In our case it means to state minimize two completely specified machines obtained from M, by setting the don't care to either 0 or 1.

State Minimization: Incompletely Specified Machines

Suppose that the – is set to be a 0.

Machine M':

PS	*NS*, *z*	
	$x = 0$	$x = 1$
s1	s3, 0	s2, 0
s2	s2, 0	s3, 0
s3	s3, 1	s2, 0

States $s1$ and $s2$ are equivalent if $s3$ and $s2$ are equivalent, but $s3$ and $s2$ assert different outputs under input 0, so $s1$ and $s2$ are not equivalent.

States $s1$ and $s3$ are not equivalent either.

So this completely specified machine cannot be reduced further.

State Minimization: Incompletely Specified Machines

Suppose that the – is set to be a 1.

Machine M'':

PS	*NS, z*	
	$x = 0$	$x = 1$
s1	s3, 0	s2, 0
s2	s2, 1	s3, 0
s3	s3, 1	s2, 0

States $s1$ is incompatible with both $s2$ and $s3$.

States $s3$ and $s2$ are equivalent.

So number of states is reduced from 3 to 2.

Machine M''_{red}:

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

State Minimization:
Incompletely Specified Machines

Can this always be done?

Machine M:

PS	*NS*, *z*	
	$x = 0$	$x = 1$
s1	s3, 0	s2, 0
s2	s2, -	s1, 0
s3	s1, 1	s2, 0

State Minimization: Incompletely Specified Machines

Machine M_2:

PS	*NS*, *z*	
	$x = 0$	$x = 1$
s1	s3, 0	s2, 0
s2	s2, 0	s1, 0
s3	s1, 1	s2, 0

Machine M_3:

PS	*NS*, *z*	
	$x = 0$	$x = 1$
s1	s3, 0	s2, 0
s2	s2, 1	s1, 0
s3	s1, 1	s2, 0

Machines M_2 and M_3 are formed by filling in the unspecified entry in M with 0 and 1, respectively.

Both machines M_2 and M_3 cannot be reduced.

Conclusion: M cannot be minimized further!

But is it a correct conclusion?

State Minimization: Incompletely Specified Machines

Notice that we want to 'merge' two states when, for any input sequence, they generate the same output sequence, **but only where both outputs are specified**.

This suggests the notion of a **compatible set of states:** a set of states that agree on the outputs where they are all specified.

Machine M:

PS	*NS, z*	
	$x = 0$	$x = 1$
s1	s3, 0	s2, 0
s2	s2, -	s1, 0
s3	s1, 1	s2, 0

In this case we have two compatible sets: A = (s1,s2) and B = (s3,s2). A reduced machine M_{red} can be built as follows.

Machine M_{red}:

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

State Minimization: Incompletely Specified Machines

Can we simply look for a set of compatibles of minimum cardinality, such that any original state is in at least one compatible? (This would be nice since it would lead to a simple unate covering problem.)

No. To build a reduced machine we must be able to send compatibles into compatibles. So choosing a given compatible may imply that some other compatibles must be chosen too.

PS	*NS, z*			
	I_1	I_2	I_3	I_4
s1	s3,0	s1,-	-	-
s2	s6, -	s2, 0	s1, -	-
s3	-, 1	-, -	s4, 0	-
s4	s1,0	-, -	-	s5, 1
s5	-, -	s5, -	s2, 1	s1, 1
s6	-, -	s2, 1	s6, -	s4, 1

A set of compatibles that cover all states is: ($s3s6$) , ($s4s6$) , ($s1s6$) , ($s4s5$) , ($s2s5$). But ($s3s6$) requires ($s4s6$), ($s4s6$) requires ($s4s5$), ($s4s5$) requires ($s1s5$), ($s1s6$) requires ($s1s2$), ($s1s2$) requires ($s3s6$), ($s2s5$) requires ($s1s2$). So, this selection of compatibles requires too many other compatibles...

State Minimization: Incompletely Specified Machines

PS	*NS, z*			
	I_1	I_2	I_3	I_4
s1	s3,0	s1,-	-	-
s2	s6, -	s2, 0	s1, -	-
s3	-, 1	-, -	s4, 0	-
s4	s1,0	-, -	-	s5, 1
s5	-, -	s5, -	s2, 1	s1, 1
s6	-, -	s2, 1	s6, -	s4, 1

Another set of compatibles that covers all states is $(s1s2s5)$, $(s3s6)$, $(s4s5)$. But compatible $(s1s2s5)$ requires $(s3s6)$, $(s3s6)$ requires $(s4s6)$ (which requires $(s4s5)$) and $(s4s5)$ requires $(s1s5)$. So must select also compatible $(s4s6)$.

Selection of minimum set of compatibles closed with respect to next state implication is a *binate* covering problem !!!

State Minimization: Incompletely Specified Machines

More formally:

When a next state in unspecified, the future behaviour of the machine is unpredictable. This suggests the definition of admissible input sequence.

Definition. An input sequence is *admissible*, or applicable, for a starting state of a machine if no unspecified next state is encountered, except possibly at the final step.

Definition. State s_i of machine M_1 is said to *cover*, or *contain*, state s_j of M_2 provided every input sequence applicable to s_j is also applicable to s_i, and its application to both M_1 and M_2 when they are initially in s_i and s_j, results in identical output sequences whenever the outputs of M_2 are specified.

Definition. Machine M_1 is said to *cover* machine M_2 iff, for every state s_j in M_2, there is a corresponding state s_i in M_1 such that s_i covers s_j.

The problem of state minimization for an incompletely specified machine M is to find a machine M' which covers M such that for any other machine M'' which covers M, the number of states of M' does not exceed the number of states of M''.

State Minimization: Incompletely Specified Machines

Machine M:

PS	*NS, z*	
	$x = 0$	$x = 1$
s1	s3, 0	s2, 0
s2	s2, -	s1, 0
s3	s1, 1	s2, 0

Machine M':

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

State A of M' covers states $s1$ and $s2$ of M and state B of M' covers states $s2$ and $s3$ of M. Therefore M' covers M.

Note that M started in $s1$ under input sequence 1 0 0 generates 0 - -, while M' started in A under input sequence 1 0 0 generates 0 0 1.

The output generated by M' corresponding to the don't care entry in M is not always the same !!!

State Minimization: Incompletely Specified Machines

Machine M_2:

PS	*NS, z*	
	$x = 0$	$x = 1$
s1	s3, 0	s2, 0
s2	s2, 0	s1, 0
s3	s1, 1	s2, 0

Machine M_3:

PS	*NS, z*	
	$x = 0$	$x = 1$
s1	s3, 0	s2, 0
s2	s2, 1	s1, 0
s3	s1, 1	s2, 0

M_2 and M_3 are formed by filling in the unspecified entry in M with 0 and 1, respectively. Neither state A nor B of M' covers state $s2$ of M_2 or M_3 and hence M' would not cover M_2 or M_3.

State Minimization: Incompletely Specified Machines

If machine M can be covered by M' containing fewer states than M, then some state of M' must cover more than one state of M. If a set of states of M can be covered by the same state of M', this set is called a **compatible set**.

Intuitively: the states in a compatible set can be combined into a single state in the reduced machine.

Definition: States s_i and s_j are *compatible* iff they never generate different specified output sequences for any admissible input sequence.

Example:

PS	*NS, z*			
	I_1	I_2	I_3	I_4
A	-	-	E, 1	-
B	C, 0	A, 1	B, 0	-
C	C, 0	D, 1	-	A, 0
D	-	E, 1	B, -	-
E	B, 0	-	C, -	B, 0

(AC) is a compatible pair, (AD) is a compatible pair if and only if (BE) is a compatible pair. (ACD) is a compatible set.

State Minimization: Incompletely Specified Machines

A set of states is compatible if and only if every pair of states in that set is compatible.

(BC) is a compatible pair, (AC) is a compatible pair, but (AB) is not compatible pair, so (ABC) is not a compatible set.

The compatibility relation is not an equivalence relation!

State Minimization: Incompletely Specified Machines

Compatible sets are computed as those sets of states that do not contain any incompatible pair of states.

States s_i and s_j are *incompatible* iff they are not compatible.

Definition: States s_i and s_j are *output incompatible* iff $\exists i_k$ such that $\lambda(i_k, s_i) \neq \lambda(i_k, s_j)$, if both λ are specified.

The set of all pairs of incompatible states can be computed as follows:

1. Compute output incompatible pairs.
2. Add any pair of states (s_i, s_j) if $\exists i_k$ such that $(\delta(i_k, s_i), \delta(i_k, s_j))$ is a previously determined incompatible pair of states.
3. Repeat 2. until no new pairs can be added to the incompatible state pairs set.

State Minimization:
Incompletely Specified Machines

Example:

PS	*NS, z*			
	I_1	I_2	I_3	I_4
A	-	-	E, 1	-
B	C, 0	A, 1	B, 0	-
C	C, 0	D, 1	-	A, 0
D	-	E, 1	B, -	-
E	B, 0	-	C, -	B, 0

Compatibles of example:
$C_1 =$ (BE),
$C_2 =$ (AD),
$C_3 =$ (CD),
$C_4 =$ (BC),
$C_5 =$ (ACD),
$C_6 =$ (DE),
$C_7 =$ (AC),
$C_8 =$ (A),
$C_9 =$ (B),
$C_{10} =$ (C),
$C_{11} =$ (D),
$C_{12} =$ (E).

State Minimization:
Incompletely Specified Machines

A class of compatibles is of special interest: **maximal compatibles**.

Sets of compatible states which are not subsets of any other compatible set of states are called *maximal compatibles*.

In the case of completely specified machines, each equivalence class is a maximal compatible.

Maximal compatibles of previous example:
(BE), (BC), (ACD), (DE).

If machine M is to be reduced to M', the states of M' must correspond to compatible sets of states of M. If a state of M' corresponds to a compatible set C_i, then the next state of M' under input I_j must correspond to some compatible set C_m such that the next state entries in M under I_j of all states in C_j are contained in C_m. (**Why is this?**)

State Minimization: Incompletely Specified Machines

Definition:
If C_i is a set of compatible states and

$$C_{ij} = \{s_k | s_k = \delta(I_j, s_i) \text{ , and} s_i \in C_i\}$$

i.e. C_{ij} is the set of next states of the states in C_i for input I_j, then C_{ij} is said to be **implied** by the set C_i for input I_j.

Definition:
Let C_i be a compatible set of states and C_{ij} be the set of next states implied by C_i for input I_j.

$$C_{ij} = \{s_k \mid s_k = \delta(I_j, s_l), s_l \in C_i\}$$

The sets C_{ij} implied by C_i for all inputs I_j are the *implied classes* of C_i.

Definition:
A set of compatible sets $C = \{C_1, C_2, ...\}$ is **closed** if for every $C_i \in C$ all the implied sets C_{ij} are contained in some element of C for all inputs I_j.

State Minimization: Incompletely Specified Machines

The problem of minimizing the number of states reduces to finding a closed set C of compatible states, of minimum cardinality, which covers every state of the original machine, i.e. a *minimum closed cover*.

Note that:

1. The set of all maximal compatibles of a completely specified FSM is the unique minimum closed cover.

2. For an incompletely specified FSM, a closed cover consisting of maximal compatibles only, may be a larger cover than a closed cover in which some or all of the compatibles are not maximal.

   This is because each compatible has a different set of implied compatibles, and removing a state from a compatible may result in an implied compatible that is already there, rather than a new one.

   This means that we may have to search over the set of all compatibles to find the minimum cover.

State Minimization: Incompletely Specified Machines

But the set of all compatibles is a very large set. Is there a smaller set of compatibles (larger than the set of all maximal compatibles) that guarantees finding a **minimum** closed cover ?

Yes. Prime compatibles (Grasselli and Luccio, 1965)

Definition: A compatible set of states that is not "dominated" by any other compatible set is called a **prime compatible** set.

Definition similar to that for prime implicants of logic functions. Of course here one must specify what is meant by **dominance** of a compatible.

State Minimization: Incompletely Specified Machines

Let C_i be a compatible set of states, and C_{ij} be the set of next states implied by C_i for input I_j.
Definition: The *class set* P_i implied by C_i is the set of all C_{ij} implied by C_i for all inputs I_j such that:

1. C_{ij} has more than one element
2. $C_{ij} \not\subseteq C_i$
3. $C_{ij} \not\subseteq C_{ik}$ if $C_{ik} \in P_i$

Definition: A compatible C_i *dominates* a compatible C_j if

1. $C_i \supset C_j$, **and**
2. $P_i \subseteq P_j$

i.e. C_i dominates C_j if C_i covers all states covered by C_j and the conditions on the closure of C_i are a subset of the conditions on the closure of C_j.

Definition: A compatible set of states that is not dominated by any other compatible set is called a **prime compatible** set.

State Minimization: Incompletely Specified Machines

Example:

PS	NS, z			
	I_1	I_2	I_3	I_4
A	-	-	E, 1	-
B	C, 0	A, 1	B, 0	-
C	C, 0	D, 1	-	A, 0
D	-	E, 1	B, -	-
E	B, 0	-	C, -	B, 0

Prime compatibles and respective class sets:

$$
\begin{aligned}
p_1 &= (BE), & &\{(CB)\}; \\
p_2 &= (AD), & &\{(BE)\}; \\
p_3 &= (CD), & &\{(ED)\}; \\
p_4 &= (BC), & &\{(DA)\}; \\
p_5 &= (ACD), & &\{(ED)(BE)\}; \\
p_6 &= (DE), & &\{(BC)\}, \\
p_7 &= (AC), & &\{\}; \\
p_9 &= (B), & &\{\}; \\
p_{11} &= (D), & &\{\}; \\
p_{12} &= (E), & &\{\}.
\end{aligned}
$$

State Minimization: Incompletely Specified Machines

The following procedure computes all prime compatibles. At the beginning, the set of prime compatibles is empty.

1. Order the maximal compatibles by decreasing size, say n is the size of the largest.

2. Add to the set of prime compatibles the maximal compatibles of size n.

3. For $i = 1$ to $n - 1$:

   (a) Generate all compatibles of size $n-i$ and compute their implied classes.
   The compatibles of size $n - i$ are generated starting from the maximal compatibles of size n to $n - i + 1$ (only those that do not have a void class set).

   (b) Add to the set of primes the compatibles of size $n - i$ not dominated by any prime already in the set.

   (c) Add to the set of primes all maximal compatibles of size $n - i$.

State Minimization: Incompletely Specified Machines

The following facts are true:

- A compatible already added to the set of primes, cannot be excluded by a newly generated compatible.

- In the previous algorithm, the same compatible can be generated more than once by different maximal compatibles. The question arises of finding the most efficient algorithm to generate the compatibles.

- Only the compatibles generated from maximal compatibles with non-empty class set need be considered, because a maximal compatible with an empty class set dominates any compatible that it generates.

- A single state s_i can be a prime compatible if every compatible set C_i with more than one state and containing s_i implies a set with more than one state.

Theorem 1 *For any FSM M there is a minimum equivalent FSM M_{red} whose states all correspond to prime compatible sets of M.*

State Minimization: Incompletely Specified Machines

A minimum closed cover can be determined by first constructing a conjunctive form expression and then finding a satisfying assignment to it which has the fewest variables assigned `TRUE`.

This is a binate covering problem.

- The table of the problem has columns that correspond to prime compatibles and rows that correspond to covering and closure conditions.

- It can be solved by branch and bound.

- At each step the table is reduced, by eliminating rows and columns, according to dominance criteria. These generalize the row and column dominance used for solving unate covering problems.

State Minimization: Incompletely Specified Machines

$$
\begin{aligned}
p_1 &= (BE), && \{(CB)\}; \\
p_2 &= (AD), && \{(BE)\}; \\
p_3 &= (CD), && \{(ED)\}; \\
p_4 &= (BC), && \{(DA)\}; \\
p_5 &= (ACD), && \{(ED)(BE)\}; \\
p_6 &= (DE), && \{(BC)\}, \\
p_7 &= (AC), && \{\}; \\
p_9 &= (B), && \{\}; \\
p_{11} &= (D), && \{\}; \\
p_{12} &= (E), && \{\}.
\end{aligned}
$$

Computation of clauses in previous example.

Clauses due to coverage:
A: $(p_5 + p_2 + p_7)$
B: $(p_1 + p_4 + p_9)$
C: $(p_3 + p_4 + p_5 + p_7)$
D: $(p_2 + p_3 + p_5 + p_6 + p_{11})$
E: $(p_1 + p_6 + p_{12})$

Clauses due to closure:
$p_1 \Rightarrow p_4$: $(\bar{p_1} + p_4)$
$p_2 \Rightarrow p_1$: $(\bar{p_2} + p_1)$
$p_3 \Rightarrow p_6$: $(\bar{p_3} + p_6)$
$p_4 \Rightarrow (p_2 + p_5)$: $(\bar{p_4} + p_2 + p_5)$
$p_5 \Rightarrow (p_1 \cdot p_6)$: $(\bar{p_5} + p_1)(\bar{p_5} + p_6)$
$p_6 \Rightarrow p_4$: $(\bar{p_6} + p_4)$

State Minimization: Incompletely Specified Machines

Need to find a satisfying assignment for the following expression with the fewest TRUE assignments.

$$
\begin{aligned}
C \;=\; & (p_5 + p_2 + p_7)(p_1 + p_4 + p_9)(p_3 + p_4 + p_5 + p_7) \\
& (p_2 + p_3 + p_5 + p_6 + p_{11})(p_1 + p_6 + p_{12}) \\
& (\bar{p_1} + p_4)(\bar{p_2} + p_1)(\bar{p_3} + p_6)(\bar{p_4} + p_2 + p_5) \\
& (\bar{p_5} + p_1)(\bar{p_5} + p_6)(\bar{p_6} + p_4).
\end{aligned}
$$

The assignment $p_1 = p_2 = p_4 =$ TRUE and all other $p_j =$ FALSE is a satisfying assignment with the fewest TRUE assignments.

State Minimization:
Incompletely Specified Machines

This corresponds to M_{red} = {(B E), (A D), (B C)} giving the following reduced machine.

PS	*NS, z*			
	I_1	I_2	I_3	I_4
(A D) $\rightarrow \alpha$	-	γ, 1	γ, 1	-
(B C) $\rightarrow \beta$	β, 0	α, 1	β, 0	α, 0
(B E) $\rightarrow \gamma$	β, 0	α, 1	β, 0	β, 0

Note:

1. Minimum form not unique.

2. A state in the original machine may be split between two states in the reduced machine.

3. resulting machine may still be incompletely specified.

State Minimization: Incompletely Specified Machines

0	A	C	1
1	A	E	-
0	B	C	-
1	B	E	1
0	C	B	0
1	C	A	1
0	D	D	0
1	D	E	1
0	E	D	1
1	E	A	0

(a)

B	✓			
C	×	E, A		
D	×	C, D	B, D A, E	
E	C, D	×	×	×
	A	B	C	D

(b)

- An incompletely-specified machine and its implication table

0	(A, B)	C	1
1	(A, B)	E	1
0	C	(A, B)	0
1	C	(A, B)	1
0	D	D	0
1	D	E	1
0	E	D	1
1	E	(A, B)	0

(a)

0	(A, E)	(B, C, D)	1
1	(A, E)	(A, E)	0
0	(B, C, D)	(B, C, D)	0
1	(B, C, D)	(A, E)	1

(b)

- Two minimal realizations of an incompletely-specified machine

State Minimization: Incompletely Specified Machines

Problem: An implied class can be contained by more than one compatible.

Choose as next state in the reduced machine the compatible that optimizes some cost function, for instance the number of rows of the state transition table of the reduced machine.

In general, define a *mapping* problem: given a closed set of compatibles which covers all the states of the original machine, find a mapping of the implied classes into the compatibles, so as to minimize the cost of the resulting machine.

The problem can be modeled as the one of determining the set of unique representatives which minimizes row count after minimization of a symbolic relation (analogous to a boolean relation with symbolic input and output fields).

State Minimization: Incompletely Specified Machines

Complexity of the problem:

- Problem is NP-hard [Pfleeger 1973]
- Exact solution requires computing prime compatibles (in the worst-case, order of $3^{n/3}$) and then a binate covering step.
- Existing tools perform well on examples with few prime compatibles. Are inadequate for examples with many prime compatibles.
- Newly developed tools use BDD's and implicit methods (Kam and Villa). They have been shown to be able to handle some problems with trillions of prime compatibles. [Kam and Villa have developed a package called SILK]