STATE ASSIGNMENT FOR MULTILEVEL LOGIC IMPLEMENTATION — MUSTANG

* Optimize Area of Multilevel Circuits.

* "Mustang: State Assignment of Finite State Machines Targeting Multilevel Logic Implementations," IEEE Trans. CAD, v. 7, n. 12, December 1988.

* Basic notion: The cardinality of the PLA (2-level) cover is an estimator for the size of a multilevel network – but a cover with larger cardinality and more common cubes should produce better multilevel implementations.

  This is done by maximizing the number of literals in the encoded representation by finding pairs and clusters of states which should be kept minimally distant in the Boolean encoding space.

  In particular, the literals of interest are those arising from common cube decomposition of the functions – as these are the simplest to analyze.

Consider the following machine:

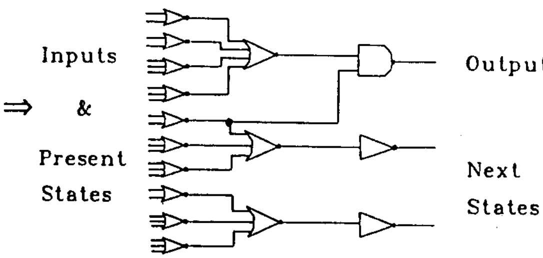
-0	st0	st0	0
11	st0	st0	0
01	st0	st1	-
0-	st1	st1	1
11	st1	st0	0
10	st1	st2	1
1-	st2	st2	1
00	st2	st1	1
01	st2	st3	1
0-	st3	st3	1
11	st3	st2	1

We can realize the states of this machine in several ways:

If we minimize the logic cover cardinality:

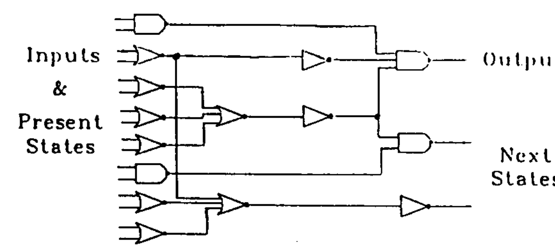
10	-1	10	0
11	1-	01	0
0-	10	10	1
01	0-	01	1
-0	-1	01	1
-1	1-	10	1

⇒



However, if we try to assign to maximize the # of common cubes, we obtain a cover (7 terms rather than 6) which has a simpler multilevel implementation:

10	10	01	1
0-	11	01	0
01	--	10	0
1-	01	01	1
0-	1-	10	1
-1	-1	01	1
0-	-1	10	1



We wish to formalize this procedure.

<u>def</u>: The Hamming distance $d(c_1,c_2)$ is defined between two code words of the same length. For codes of length k, the distance is:

$$d(c_1,c_2) = \sum_{i=0}^{k-1}\left(1 \text{ if } c_1^i \neq c_2^i\right) (0 \text{ else})$$

where c_1^i is the i^{th} bit in the encoding of c_1.

<u>Eq</u>: 1 0 1 1 and 1 0 0 1 have Hamming distance = 1
0 1 0 1 and 1 0 1 0 have Hamming distance = 4.

Note: The Hamming distance provides a discrete metric on the Boolean space of dimension n, since,

1. $d(c_1,c_2) = d(c_2,c_1) \quad \forall_{c_1,c_2}$
2. $d(c_1,c_1) = 0 \quad \forall_{c_1}$

The number of codes at distance k from some code is a rapidly increasing function of k for $k << n$. n = dim. of B.

1. Number of codes <u>at</u> distance $1 = n$ (one bit charge per word)

   Number of codes <u>at</u> distance $i = \binom{n}{i}$ (i bit charges per word)

Note: If two codes c_1 and c_2 represent states which share similar outputs then if $d(c_1,c_2) = d$ the logic implementing those outputs share a cube of dimension $n-d$, since the transiting states are similar in that many places.

Problem: Assign an encoding to states so that the maximum # of common cubes are generated in the output and transition functions.

We wish to capture 4 effects in common cube sharing:

1. If s_1 and s_2 both transit to s_3 and we assign codes to s_1 and s_2 with $d(s_1, s_2) = N_d$ then the logic generating s_3 will have a common cube with $N - N_d$ literals.

2. If s_1 and s_2 are both successors to s_3, the logic for s_1 and s_2 both contain a cube whose size is the <u>weight</u> of s_3. (weight of c_1 is $d(c_1,0) \equiv$ # of on-bits in c_1)

3. If two inputs i_1 and i_2 produce the same state from either the same or different states, then there is a common cube corresponding to state $i_1 \cap i_2$.

4. Finally if multiple states assert similar outputs on different states – there is a common cube among those states to generate each of the implicant codes.

Strategy: Between each pair of states set a weight used to determine if the two states should be placed in a local cluster (of Hamming – distance close state encodings). Then use this matrix of weights to grade embeddings of these states in the Boolean lattice.

Note: We do <u>not</u> know which cubes will be used, so we must set the heuristic to create as large a set as possible.

Construction of a matrix representing all 4 types of cube sharing appears to be difficult –

2 algs. are proposed: Fanin and Fanout oriented:

Fanout Alg.:

Depends on the outputs and Fanout of each state. Pairs of present states with similar outputs or successors are given large edge weights. $\Rightarrow$ Maximize the <u>size</u> of cubes.

```
for (i = 1 ; i ≤ N_o ; i = i+1)          {
    foreach (edges e(v_k, v_l) ∈ G) {
        if (W(e)·output[i] is 1) {
            OUTPUT_SET_i = OUTPUT_SET_i ∪ v_k
            nw(OUTPUT_SET_i , v_k) = nw(OUTPUT_SET_i ,
            v_k)+1
        }
    }
}
foreach(edges e(v_k, v_l) ∈ G) {
    N_STATE_SET_l=N_STATE_SET_l ∪ v_k
    nw(N_STATE_SET_l, v_k ) = nw(N_STATE_SET_l, v_k)
    +1
}
```

<u>N_0</u> counts the number of distinct outputs –

$W(e)\cdot \text{output}[i] \equiv 1 \Rightarrow e(v_k, v_l)$ asserts output i.

<u>nw</u> stores the weight of each node in the sets.

OUTPUT_SETS_i is set of all states with an edge whose output is output(i).

N_STATE_SET_i is set of all states whose successor is state i.

CONSTRUCTING THE WEIGHTS

```
foreach(v_k, v_l) ∈ G_M) {
    for(i = 1 ; i ≤ N_s ; i = i+1)
        we (e_M(v_k, v_l))  =  we (e_M(v_k, v_l)) +
          nw(N_STATE_SET_i , v_k) * nw(N_STATE_SET_i ,
          v_l)
    we (e_M(v_k, v_l)) = we(e_M(v_k, v_l)) * N_b / 2
    for (i = 1 ; i ≤ N_o ; i = i+1)
        we (e_M(v_k, v_l))  =  we (e_M(v_k, v_l)) +
        nw(OUTPUT_SET_i , v_k) * nw(OUTPUT_SET_i ,
        v_l)
}
```

<u>we</u> is an edge weight for each edge of G_M: i.e. for each pair of states.

The algorithm accumulates edge weights for each edge as the product of each pair of state's node weights summed over all states.

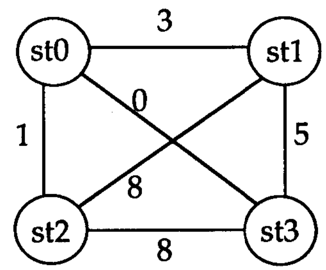
$\rightarrow$ *nw* counts the number of <u>occurrences</u> of a common cube in the state or output parts.

The edge weight is set to the number of occurrences <u>times</u> the occurrences common to the other edge.

The $N_b/2$ factor reflects the "average" number of bits in common between two states.

(Assuming the number of states is relatively close to the <u>size</u> of the Boolean encoding.)

Example:	-0	st0	st0	0
	11	st0, st1	st0	0
	01	st0	st1	-
	0-	st1	st1	1
	10	st1	st2	1
	1-	st2	st2	1
	00	st2	st1	1
	01	st2	st3	1
	0-	st3	st3	1
	11	st3	st2	1



<u>output = 1:</u> (st1^2, st2^3, st3^2)
output = 0: (← <u>no need to implicate</u>)

N_State_Set
0: st0^2,st1^1
1: st0^1, st1^1, st2^1
2: st1^1, st2^1, st3^1
3: st2^1, st3^1

so for (st1, st3):

nss(0,1) * nss(0,3) + = 1 * 0 +
nss(1,1) * nss(1,3) + = 1 * 0 +
nss(2,1) * nss(2,3) + = 1 * 1 +
nss(3, 1) * nss(3,3) = 0 * 1 = 1
$N_b / 2 = 2/2 = 1$
os(1,1) * os(1,3) = 2 * 2 = 4

⇒ Total st1 ↔ st3 = 5

Fanin-alg: We wish to exploit the input similarities in the machine: we shall count the number of similar inputs leading to a common state and the number of similar states leading to a new common state. ⇒ Maximize the # (number) of common cubes.

```
foreach(edge e(v_k,v_l) ∈ G) {
    P_STATE_SET_k = P_STATE_SET_k ∪ v_l
    nw(P_STATE_SET_k, v_l) = nw(P_STATE_SET_k, v_l)
    +1
}
```

We construct sets of similar present states which fan-in to next states.

<u>*nw*</u> will count the number of times (edges) in which state k transited to state e.

Inputs: Count #'s of next states when input is 1, or 0 for each bit:

```
for(i = 1 ; i ≤ N_i ; i = i+1) {
    foreach(edge e(v_k,v_l) ∈ G) {
        if (W(e).input[i] is 1) {
            INPUT_SET_i^ON = INPUT_SET_i^ON ∪ v_l
            nw(INPUT_SET_i^ON, v_l) = nw(INPUT_SET_i^ON,
            v_l)+1}
        if (W(e).input[i] is 0) {
            INPUT_SET_i^OFF = INPUT_SET_i^OFF ∪ v_l
            nw(INPUT_SET_i^OFF, v_l)  =  nw(INPUT_SET_i^OFF,
            v_l)+1
        }
    }
}
```

Finally, calculate the edge weights for the graph as before –

this time the scaling is slightly different and the ON(1) and OFF(0) sets of inputs are both counted.

```
foreach(v_k, v_l) ∈ G_M) {
   for(i = 1; i ≤ N_s ; i = i+1)
      we (e_M(v_k,v_l)) = we (e_M(v_k,v_l)) + nw(P_
        STATE_SET_i , v_k) * nw(P_STATE_SET_i , v_l)
   we (e_M(v_k,v_l)) = we(e_M(v_k,v_l)) * N_b
   for (i = 1; i ≤ N_i ; i = i+1) {
      we (e_M(v_k,v_l)) = we (e_M(v_k,v_l)) + nw
      (INPUT_SET_i^ON , v_k) * nw(INPUT_SET_i^ON , v_l)
      we (e_M(v_k,v_l)) = we (e_M(v_k,v_l)) + nw
      (INPUT_SET_i^OFF , v_k) * nw(INPUT_SET_i^OFF ,
      v_l)
   }
}
```

The edge weight for present states is counted by noting that each time a present state produces multiple next states, the present state (cube's) are common to the next state's encodings. The number of occurrences depend on the intersection of the two state codes. So next-state pairs with many common present states are given large edge weights to maximize the intersection.

Edge weights for inputs are counted for next states by first finding the number of similar inputs for a given next state. Then edge weights are assigned to state pairs with many common inputs thus forcing the encoding to place these states with similar encodings.

Ex. Fanin alg: (Same Example), input sets:

$i_1(0) \rightarrow (st1^3, st3^2)$;
$i_1(1) \rightarrow (st0^2, st3^3)$;
$i_2(0) \rightarrow (st0^1, st1^1, st2^1)$;
$i_2(1) \rightarrow (st0^2, st1^1, st2^1 st3^1)$;

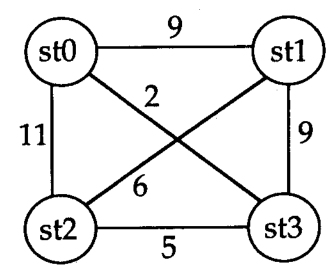
pss: $st0 \rightarrow st0^2, st1^1$
$st1 \rightarrow st0^1, st1^1, st2^1$
$st2 \rightarrow st1^1, st2^1, st3^1$
$st3 \rightarrow st2^1, st3^1$

so: for (st0, st1):

(states):
$$pss(0,0)*pss(0,1) = 2*1+$$
$$+pss(1,0)*pss(1,1) = 1*1+$$
$$+pss(2,0)*pss(2,1) = 0*1+$$
$$+pss(3,0)*pss(3,1) = 0*0 = \underline{3} \cdot N_b = 6$$

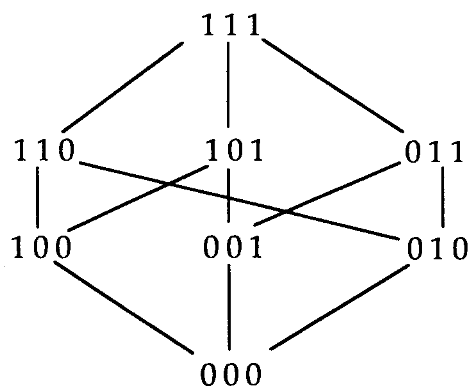
(inputs):
$$iw^0(0,0)*iw^0(0,1)+ = 0*3+$$
$$iw^1(0,0)*iw^1(0,1)+ = 2*0+$$
$$iw^0(1,0)*iw^0(1,1)+ = 1*1+$$
$$iw^1(1,0)*iw^1(1,1) = 2*1 = 3$$

$\Rightarrow w = 9.$



We are now left with the classical problem of assigning codes to the states to minimize the sum of the edge weights times the Hamming distances between the codes.

Consider the Boolean <u>lattice</u>: The elements of an n-dimensional Boolean space connected by the Hamming metric:



We wish to assign states to vertices of this lattice to minimize the induced total weight of the graph. The induced weight is the Hamming distance between two codes times the edge weight in the weight matrix.

i.e. Minimize $\sum_{i=1}^{N_s} \sum_{j=1}^{N_s} we(v_i, v_j) * d(enc(v_i), enc(v_j))$.

This problem is NP-hard as it contains a specialized but NP-complete graph embedding problem.

There are several heuristics $\rightarrow$ here we shall use <u>wedge clustering</u>. This should prove an effective technique since the constraint matrices constructed have strong structure. i.e. we expect several strongly connected clusters only weakly interacting.

Wedge clustering is essentially greedy assignment.

<u>Wedge Clustering</u>:

$G = G_M$
while $(|G| \neq 0)$ {
  Select $v_\ell \in G$ and y_i's $\in G \ni \sum_{i=1}^{N_b} we(v_\ell, y_i)$ is <u>maxima</u>.
  Assign v_ℓ and the y_i's minimally distant codes from those which are unassigned;
  (note v_ℓ and the y_i's may already have assignments.)
  delete v_ℓ from G and all edges (v_ℓ, y_i).
}

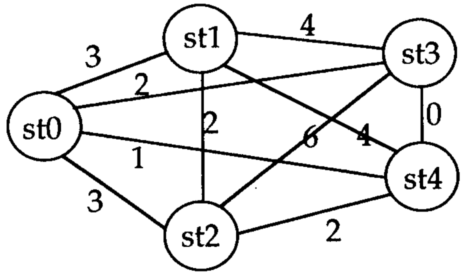
Note: This essentially finds $v_\ell \ni$ there are exactly $N_b + 1$ codes to assign in the cluster (ideally). This can be done if no other codes in the family are assigned.

In the case where we can assign unidistant codes to the y_i's from v_ℓ and when:

$$we(v_\ell, y_i) \geq we(y_i, y_j) + we(y_i, y_k) \forall_{i,j,k}.$$

This alg. makes a minimal cost assignment for this cluster.

Example: $N_b = 3$ for 5 states:

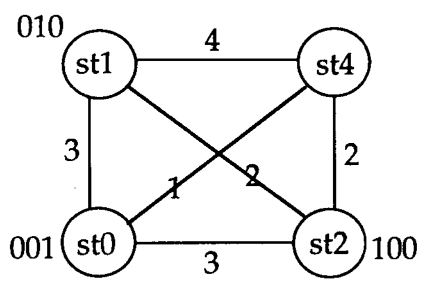


$N_b = 3$

st3 (st0, st1, st2)

st3 $\rightarrow$ 000 st0 $\rightarrow$ 001

st1 $\rightarrow$ 010 st2 $\rightarrow$ 100



st1 (st0, st2, st4)

st4 $\rightarrow$ 110

st3 is chosen as it has the maximum set of 3 edge weights.
$\Rightarrow$ st3 $\rightarrow$ 000 and st2,st1,st0 are unidistant.

delete st3 and edges: now choose st1 (010)

we need st4 to be 1 unit distant from
st1 (010) and 110, 111, 011, 101 are free.

so st4 could be either 110 or 011.

Results:

RESULTS OBTAINED USING SIMPLE CHART

EXAMPLE	RANDOM-A		RANDOM-B		KISS		MUST-P	MUST-N	MUSTANG
	#lit	RAT	#lit	RAT	#lit	RAT	#lit	#lit	#lit
bbara	120	1.48	91	1.12	103	1.27	81	108	81
bbsse	214	1.48	190	1.31	145	1.01	144	177	144
bbtas	37	1.15	26	0.81	34	1.06	51	32	32
cse	405	1.33	339	1.11	264	0.87	304	319	304
dk15x	122	1.17	109	1.04	91	0.87	104	128	104
dk16x	553	1.59	516	1.49	411	1.18	383	346	346
keyb	810	2.45	663	2.00	474	1.43	495	330	330
lion	20	1.11	18	1.00	21	1.16	18	22	18
lion9	61	3.05	52	2.60	37	1.85	25	20	20
mark1	112	1.28	89	1.02	114	1.31	130	87	87
mc	40	1.11	37	1.02	43	1.19	37	36	36
modulo12	43	1.19	40	1.11	49	1.36	40	36	36
planet	1063	1.24	1012	1.18	869	1.01	1033	854	854
s1	852	4.26	805	4.02	690	3.45	639	200	200
s1a	649	4.0	583	3.59	382	2.35	514	162	162
scf	1674	1.31	1596	1.25	1441	1.13	1390	1274	1274
shiftreg	37	18.5	32	16.0	8	4.00	2	8	2
tav	25	1.04	24	1.00	24	1.00	25	24	24
tbk.min	540	1.12	532	1.10	563	1.16	515	482	482
train11	67	1.34	53	1.06	46	0.92	50	55	50
TOTAL	7444	1.62	6807	1.48	5809	1.26			4586

STATISTICS OF BENCHMARK EXAMPLES

EXAMPLE	#inp	#out	#states	#enc
bbara	4	2	10	4
bbsse	7	7	16	4
bbtas	2	2	6	3
cse	7	7	16	4
dk15x	3	5	4	2
dk16x	2	3	27	5
keyb	7	2	19	5
lion	2	1	4	2
lion9	2	1	9	4
mark1	5	16	14	4
mc	3	5	4	2
modulo12	1	1	11	4
planet	7	19	48	6
s1	8	6	20	5
s1a	8	6	20	5
scf	27	56	128	7
shiftreg	1	1	8	3
tav	4	4	4	2
tbk.min	6	3	16	4
train11	2	1	10	4

RESULTS AFTER INTENSIVE SCRIPT AND TECHNOLOGY MAPPING

EXAMPLE	RANDOM-B		KISS		MUSTANG-N	
	#lit	#gate	#lit	#gate	#lit	#gate
cse	240	115	203	95	220	105
dk16x	394	175	315	143	290	124
keyb	311	158	213	112	210	112
planet	654	290	547	249	563	267
s1	354	174	352	173	160	93
s1a	337	169	258	131	141	83
scf	922	445	861	401	852	393
tbk.min	342	170	381	169	297	130

Boolean Graph Embedding

- Wedge Clustering is a fast algorithm but is not necessarily a good minimizer of the cost.
- Annealing provides a much better solution due to a simple move and reasonably smooth cost manifold.

Problem (Undirected Graph Embedding): Given a graph G(V,E) with weighted edges E, assign unique binary vectors to the vertices such that the cost function:

$$C = \sum_{(v_1, v_2) \in E} w(v_1, v_2) H(v_1, v_2)$$

is minimized. H(a,b) is the Hamming distance between a and b.

A fast annealing move can be found by noticing that if a random bit in some encoding vector is inverted, there are only two possibilities (all codes are unique):

1. the new code (after the inversion) is a new unique vector which is not any other code.

2. the new code is identical to an already existing code assignment. In this case, we can simply swap the code assignments to finish the move.

All that is necessary to complete the move is to determine the change in the cost of the embedding. Since only a single bit has been changed, (in at most 2 codes) updating the cost is simple.

Consider the set of edges which impinge on v_1, the initial value and on v_2, the new value. Since only one bit has been changed, the Hamming distance for each of these edges in the final cost can have been changed by at most the weight of each edge. Again there are 2 cases:

1.The edge is adjacent to a vertex which has just decreased its Hamming distance by one (it agrees in the swapped bit with the changing encoding).

2. The edge is adjacent to a vertex which has just increased its Hamming distance by one (it disagrees in the swapped bit place with the new encoding).

So to update the cost, simply add the weight for each edge to a vertex with a bit agreeing with the old encoding and subtract a weight for each bit agreeing with the new encoding. This requires O(n) time for n codes.

In the case of swapping codes, you must do this for each of the two codes (also O(n)).

A simple way to determine if a given code is used is to make a matrix of vertices, indexed by the codes themselves. If the code is allocated, the vertex index saved as the value, else a value corresponding to no code is saved at that address.

If a fast method for calculating the weight of a Boolean Vector exists (say a table), this can be improved by randomly selecting a new code the replace the one in question and scaling the changes by the fast Hamming metric which is just the weight of the bit-wise exclusive-or of the two codes. This results in a faster annealing, at the cost of a slight increase in size of the program.

Based on this move several variant algorithms are possible:

- Annealing-- best cost performance, relatively easy to get optimal solutions for non-negative weight matrices.
- Kernighan-Lin based constrained swapping (swap and continue until no more are possible, choose best sequence of moves).
- Greedy move selection.

Surprisingly, all three of these algorithms out perform wedge clustering in terms of the cost of the results. Kernighan-Lin and greedy are both typically fairly fast, but K-L gives consistently better results. Note that for most matrices generated by these methods, minimal code lengths are almost always generated as minimal cost; extra bits, when present, are not used. This is one of the primary faults of this technique-- the current heuristic weights and embeddings select against larger (possibly simpler) encodings.

Comparison of different Embedding Strategies

sample	greedy	gtime	K-L	Ktime	Anneal	Atime	states
mouse2	1402	0.1s	1400	0.1s	1304*	10.1s	9
foo2	11568	0.1s	11561	0.3s	11440*	17.5s	16
cor	27842	3.3s	27866	4.1u	27466	37.3s	45
midi	88068	50.1s	88439	69.6s	88023	97s	111

Applications of Boolean Relations

- We will revisit several encoding problems for state machines from the context of Boolean Relations will allow a much simpler and more consistent approach.
- We will define and use Symbolic (Multiple-Valued) input and output relations as well.
- To set the stage:

def: a Boolean Relation is a one to many mapping $R \subseteq B^n \times B^m$. For each

$$x \in B^n,\ R(x) = \left\{ y \in B^m \middle| (x,y) \in R \right\}$$

is the set of possible mappings for x (the image of x).

B^n denotes the domain of R
B^m denotes the co-domain of R

for a set $X \subseteq B^n$, the image of X with respect to R is $R(X) = \left\{ y \in B^m \middle| \exists x \in X : (x,y) \in R \right\}$.

def: R is well defined if $\forall x \in B^n \ \ R(x) \neq \phi$.

*def: A multi output function f is a mapping compatible with R if $\forall x \in B^n$, $f(x) \in R(x)$.
This is written: $f \prec R$

def: Let $D_1 \ldots D_r$ be sets of symbols and $\Sigma_1 \ldots \Sigma_n$ be sets of symbols, then a mapping f:

$$f : D \to \Sigma \quad \text{where } D = D_1 \times D_2 \times \ldots \times D_r$$
$$\Sigma \doteq \Sigma_1 \times \Sigma_2 \times \ldots \times \Sigma_n$$

is an r input, n output Symbolic function if for each minterm $x \in D$ f maps exactly one $y \in \Sigma$.

f is completely specified if each y is a specific element of Σ.

*def: a Symbolic Relation $R \subseteq D \times \Sigma$ is a one to many mapping of D into Σ. As before D is the domain of R and Σ is the co-domain.

def: for each minterm $x \in D$, the image of x is the set of mappings $R(x) = \{ y \in \Sigma | (x,y) \in R \}$

the image of a set $X \subseteq D$ is the set $R(X)$

$$= \{ y \in \Sigma | \exists x \in X : (x,y) \in R \}.$$

def: $R \subseteq D \times \Sigma$ is well defined if $\forall x \in D, R(x) \neq \phi$.

*def: a Boolean function $f : B^r \to B^n$ is a compatible mapping for $R \subseteq D \times \Sigma$ if there exist $\xi : D \to B^r$ an input mapping and $\psi : \Sigma \to B^n$ an output mapping such that $\forall x \in D, \exists y \in R(x)$ with

$$f(\xi(x)) = \psi(y) \Leftrightarrow f \prec_{\xi,\psi} R$$

ξ and ψ are also called encodings of the input symbols D and the output symbols Σ.

def: a finite state machine FSM is defined as a 6-triple: $M = (I, 0, \Sigma, \delta, \lambda, \sigma_s)$

where I, 0 are the sets of inputs and outputs and Σ is the set of states.

$\delta : I \times \Sigma \to \Sigma$ is the next state function.
$\lambda : I \times \Sigma \to 0$ is the output function.

σ_s is assumed for the machine to be a start set of states or state.

def: a state is reachable if $\exists a$ sequence of inputs taking the start state to a reachable state.

def: two states σ_1 and σ_2 are equivalent if no sequence of inputs applied to σ_1 and σ_2 result in conflicting outputs.

def: if the machine is incompletely specified then two states are compatible if no admissible sequence of inputs applied to both σ_1 and σ_2 can force a conflicting output.

def: a sequence is admissible iff each transition is from a well defined state to a well defined state.

State Assignment Revisited

Consider the following machines:

$$\begin{array}{llll} M_1: & 0 \;;\; s_1 & \rightarrow s_2 & ;\; 1 \\ & 1 \;;\; s_1 & \rightarrow s_3 & ;\; 0 \\ & - \;;\; s_2,s_3 & \rightarrow s_4 & ;\; 1 \\ & - \;;\; s_4 & \rightarrow s_1 & ;\; 1 \end{array} \qquad \begin{array}{llll} M_2: & 0 \;;\; s_1 & \rightarrow s_2 & ;\; 1 \\ & 1 \;;\; s_1 & \rightarrow s_2 & ;\; 0 \\ & - \;;\; s_2 & \rightarrow s_4 & ;\; 1 \\ & - \;;\; s_4 & \rightarrow s_1 & ;\; 0 \end{array}$$

M_2 is equivalent to M_1 since the two machines produce identical outputs given the same inputs. However M_2 is state minimal.

Note that <u>no</u> conventional state assignment encoding for M_1 will produce the logic N_2 below:

$$\begin{array}{lccccccc} N_2: & 0 & 0 & 0 & \rightarrow & 0 & 1 & 1 \\ & 1 & 0 & 0 & \rightarrow & 0 & 1 & 0 \\ & - & 0 & 1 & \rightarrow & 1 & 0 & 1 \\ & - & 1 & 0 & \rightarrow & 0 & 0 & 0 \end{array}$$

However, this machine can be constructed directly from M_2.

⇒ We must take <u>equivalent</u> machines into account when synthesizing a new state machine.

However, there is no guarantee that choosing a minimal machine will lead to a smaller design . . .

Consider the following machines:

$$\begin{array}{lcccc} M_1: & 0 & s_0 & s_2 & 0 \\ & 1 & s_0 & s_6 & 0 \\ & 0 & s_1 & s_3 & 0 \\ & 1 & s_1 & s_7 & 0 \\ & 0 & s_2 & s_0 & 0 \\ & 1 & s_2 & s_5 & 0 \\ & 0 & s_3 & s_1 & 0 \\ & 1 & s_3 & s_3 & 0 \\ & 0 & s_5 & s_0 & 1 \\ & 1 & s_5 & s_1 & 1 \\ & - & s_6 & s_3 & 0 \\ & - & s_7 & s_2 & 0 \end{array} \quad \Leftrightarrow \quad \begin{array}{lcccc} M_2: & 0 & s_0 & s_2 & 0 \\ & 1 & s_0 & s_6 & 0 \\ & 0 & s_1 & s_3 & 0 \\ & 1 & s_1 & s_7 & 0 \\ & 0 & s_2 & s_0 & 0 \\ & 1 & s_2 & s_5 & 0 \\ & 0 & s_3 & s_1 & 0 \\ & 1 & s_3 & s_4 & 0 \\ & 0 & s_4 & s_1 & 0 \\ & 1 & s_4 & s_3 & 0 \\ & 0 & s_5 & s_0 & 1 \\ & 1 & s_5 & s_2 & 1 \\ & - & s_6 & s_3 & 0 \\ & - & s_7 & s_2 & 0 \end{array}$$

(8 terms) (7 terms)

Although they are equivalent, the reduced machine takes 8 terms in an optimal 2-level encoding while the larger machine can be built in 7 terms.

(Note that state s_3 of M_1 is equivalent to s_3,s_4 of M_2.)

Note that the state mapping in which s_3 could transit to its successors or to those of s_4 defines a <u>relation</u> and not a function since there are several possible mappings for s_3 even though the machines are completely specified.

- We will define the encoding problem for a state machine in terms of Symbolic Relations of the STG. This will allow more degrees of freedom to encode the machine since we may or may not give the same encoding to two equivalent states.

We can solve several kinds of Symbolic Relation Encoding problems for 2-level logic exactly using BDD's.

1. Generate all candidate primes for the relations.

2. We must solve a covering problem with weighted cover. However, unlike the function case, we must impose constraints on the covering to ensure that the encoding does maintain compatibility with the Relation.

3. After all constraints have been specified, we solve the covering problem as an instance of Binate Covering. (Actually Weighted Binate Cover.)

- Generation of Primes   Ref: "Exact Minimizer for Boolean Relations," ICCAD-89, Nov., pp. 316-319.

(Boolean Case)

def: a cube $C \subseteq B^r \times B^n$ is a Boolean Relation denoted

$$\begin{aligned} C = (c/I) \ &\ni \ c \text{ is a cube of } B^r,\ I \subseteq B^n \text{ and} \\ C(x) \ &= \ I \ \forall x \in c \\ &= \ \{0\},\ \forall x \notin c. \end{aligned}$$

c is the support set of C and I is the influence set. The size of C is $|c|$.

*def: a candidate prime of a Boolean Relation R is a prime of a compatible mapping $f \prec R$.

def: Let $R \subseteq B^r \times B^n$ be a Boolean Relation and $C \subseteq B^r \times B^n$ be a cube. C is an implicant of R iff:

$$\forall x \in B^r,\ \forall y' \in C(x),\ \exists y \in R(x) \ni y' \leq y \quad (y \text{ bit-wise contains } y')$$

def: a prime implicant (C/I) of R is a cube with the property that $i \in I$ iff (c/i) is a c-prime of R.

def: a fundamental implicant of a Relation $R \subseteq B^r \times B^n$ is an implicant (X/I) where $x \in B^r$ and $I = R(x)$.

Note that we could enumerate primes of R by choosing every mapping and counting all primes of each mapping – but this is very expensive. Instead we will form the fundamental implicants and use pair-wise consensus to generate the larger primes.

def: 2 cubes C' and C'' are adjacent iff their supports c', c'' are hamming distance one from each other. i.e., they differ in only one bit.

def: the merge of 2 adjacent cubes c' and c'' denoted $C = C' \circ C''$ is a cube C with support set

$$c = c' \cup c''$$

and influence set

$$I = \Big\{ y \in B^n : y = \text{greatest lower bound } (y', y'') \quad \text{where } y' \in I', y'' \in I'' \Big\}$$

here the glb is simply bit-wise AND of the elements.

input: The set of fundamental implicants of R.
output: The set P of all prime implicants of R.

```
A_0 = {all fundamental implicants of R};
P = 0;
for  s = 1, . . . , r {
    B_s = 0;
    for each pair (a',a'') ∈ A_{s-1} × A_{s-1}, a' ≠ a''{
        if a' is adjacent to a'' {
            b = a' ∘ a'';
            mark all i' ∈ I' such that ∀i'' ∈ I'', i' ≤ i'';
            mark all i'' ∈ I'' such that ∀i' ∈ I', i'' ≤ i';
            B_s = B_s ∪ {b};
        }
    }
    remove all marked vertices from the influence sets of
    the cubes in A_{s-1};
    P = P ∪ A_{s-1};
    A_s = B_s;
}
P = P ∪ A_r;
```

- s ranges from 1 to r; after each iteration all maximal implicants of size s are produced.
- A_{s-1} or A_s refers to the set of maximal implicants of size s.

- Each member of B_s is produced by merging the adjacent cubes of A_{s-1} and each is a maximal cube.
- As each new cube is produced, mark the vertices of the influence sets that are covered by the adjacent influence set. This ensures that implicants from the earlier iteration A_{s-1} which are contained by the current set are deleted. Thus each primed set is prime elements.

<u>Ex</u>: Consider the following relation:

R: 000	00			
001	00			
010	00			
100	00	the fundamental implicants of R are:		
011	10	c_1	011	10
101	01	c_2	101	01
110	00,11	c_3	110	11
111	00,11	c_4	111	11

c_1 is adjacent to $c_4 \Rightarrow$ form $c_5 = c_1 \circ c_4 = -11|10$
c_2 is adjacent to $c_4 \Rightarrow$ form $c_6 = c_2 \circ c_4 = 1-1|01$
c_3 is adjacent to $c_4 \Rightarrow$ form $c_7 = c_3 \circ c_4 = 11-|11$

There are no more adjacencies so $c_1 \ldots c_7$ are the complete set of c-primes.

Note that these primes cannot be used in the conventional way to construct a cover since we must insure that the function f so produced is compatible with R.

In general $f = \alpha_1 c_1 + \alpha_2 c_2 + \alpha_3 c_3 + \cdots + \alpha_7 c_7$

However, we must have: $f(x) \in R(x),\ \forall x \in B^r$

We can form these constraints in conjunctive form using 2^r terms:

$$000,010,001,100 \Rightarrow 1$$
$$011 \Rightarrow (\alpha_1 + \alpha_5)$$
$$101 \Rightarrow (\alpha_2 + \alpha_6)$$
$$110 \Rightarrow (\alpha_3 + \alpha_7 + \overline{\alpha_3}\,\overline{\alpha_7})$$
$$111 \Rightarrow (\alpha_4 + \alpha_7 + \alpha_5 \alpha_6 + \overline{\alpha_5}\,\overline{\alpha_6}\,\overline{\alpha_7}\,\overline{\alpha_4})$$

We can write:

$C=(\alpha_1+\alpha_5)(\alpha_2+\alpha_6)(\alpha_3+\alpha_7+\overline{\alpha_3}\,\overline{\alpha_7})(\alpha_4+\alpha_7+\alpha_5\alpha_6+\overline{\alpha_4}\,\overline{\alpha_5}\,\overline{\alpha_6}\,\overline{\alpha_7})$
which must be true for $f \in R$. Each cube has a number of literates related to its size, we wish to satisfy C with an assignment to $\alpha_i \ni \sum_i \alpha_i \omega_i$ is minimized. This is an instance of the Binate Cover Problem (weighted). Note that $\alpha_1=\alpha_2=1 \quad \alpha_3=\alpha_4=\alpha_5=\alpha_6=\alpha_7=0$ is a solution with weight 6

$$\alpha_5=\alpha_6=1 \quad \alpha_1=\alpha_2=\alpha_3=\alpha_4=\alpha_7=0$$

is a solution with weight 4.

We will show a very fast way to solve BCP which is linear time in the size of the constraint BDP.

- C-Primes for the Symbolic Case

In the case of symbolic relations we must account for all possible combinations of code choices (encodings) and output choices. Compatibility here is:

$$\forall x \in D,\ \exists y \in R(x) \ni f(\xi(x))=\psi(y)$$

for encoding functions ξ and ψ.

def: a fundamental implicant of a symbolic relation

$R \subseteq D\times\Sigma$ is an implicant $(x \mid \sigma)$ where $x \in D$, $\sigma = R(x)$.

def: a symbolic c-prime (or gc-prime) of a symbolic Relation R is a cube $(c \mid \sigma) \subseteq D\times\Sigma \ni$ there exists an input encoding $\xi : D \to B^r$ and an output encoding $\psi : \Sigma \to B^n$ for which $(\xi(c) \mid \psi(\sigma))$ is a prime of f where

$$f \prec_{(\xi,\psi)} R.$$

We will adopt a similar strategy as before and generate a set of prime implicants for these symbolic relations. Then we will derive a set of constraints which allow the realization of a two-level minimal compatible function f to be found over all encodings of inputs and outputs. This will also be cast as a Binate Cover Problem.

- To generate all prime implicants for a symbolic Relation we use an encoding trick proposed in Devadas and Newton, "Exact Algorithms for Output Encoding, State Assignment and Four-level Boolean Minimization," ICCAD, January 1991.

The idea is to encode the output symbols as 0-hot encodings:

i.e. $\sigma_1 = 011111...1 \quad \sigma_2 = 101111...1 \quad \sigma_3 = 110111...1 \quad ...$

Given $N=|\Sigma|$ we use N bits to encode the outputs.

It can be shown that the prime implicants of this encoding have the same correspondence as those of the original function.

For Relations, we may have the choice of several outputs for a given implicant so:

Encode the relation as above (output encoding) to make the fundamental implicants. Then apply the *c*-prime generation procedure with the following changes:

1. *c*-primes with all-1 output can (and should) be removed.

   Note: in a 0-hot encoding, all-ones does not assert any output symbols $\Rightarrow$ it cannot be a member of a primal cover.

2. If we know a-priori that the eventual output encoding length is L we can remove several cubes since:

   for $n > 2^{L-1}$, any n district codes have a zero intersection.

   $\Rightarrow$ for each new cube, the number of zeros in its output part are counted, if greater than 2^{L-1} $\Rightarrow$ cube is non-prime for final encoding of length ℓ.

This technique produces all cube prime implicants of the original Relation for Boolean encoded inputs and can be generalized for multiple value inputs.

- Derivation of constraints: Given *c*-primes from above, we wish to derive the necessary constraints for their inclusion in a compatible function *f*.

- Output Encoding

  Given a Binary input, symbolic output relation:

  $$R \subseteq B^r \times \Sigma \text{ where } \Sigma = \{\sigma_1 \ldots \sigma_N\}$$

  Encode Σ with at most L bits $\ni \sigma_k = b_{k_1} b_{k_2} \ldots b_{k_L}$ such that the number of product terms in the minimized relation is a minimum.

There are 2 constraints which are needed to validate a given encoding assuming $N \leq 2^L$:

1. We must have compatibility to a realizable function *f* subject to the output choices of R.

2. We must insure that each symbol is uniquely encoded.

(1) Consider a minterm $x \in B^r$ ωlog let x have $R(x) = \{\sigma_1, \ldots \sigma_p\}$ as possible mappings. Let $g_1 \ldots g_n$ be the set of all prime implicants which cover x.

Let $\prod_{j=1}^{q_i} \sigma_{i_j}$ be the output of g_i

Finally let $I_K = \left\{ i \mid i \in I \wedge (\exists_j)(\sigma_{i_j} = \sigma_K) \right\}$ i.e., I_K is a list of indices for which a prime covering x also includes σ_K in its output port.

$\bar{I}_K$ is the complements set: $I - I_K$ for $I = \{1 \ldots n\}$

We must have

$$\sum_{K=1}^{p} \left(\left(\prod_{i \in \bar{I}_K} \bar{g}_i \right) \left(\left(\sum_{i \in I_K} g_i \left(\prod_{j=1}^{q_i} e(\sigma_{i_j}) \right) \right) \equiv e(\sigma_K) \right) \right) = 1.$$

$e(\sigma_{i_j})$ represents the encoding of the given symbol.

Each g_i represents a Boolean selection variable corresponding to the inclusion of the related g-prime in the cover, the b_{i_j}'s below represent the encoding variables. i.e., b_{i_j} is the j-th bit of the i-th symbol.

Note: the first term $\left(\prod_{i \in \bar{I}_K} \bar{g}_i \right)$ ensures no illegal set of simultaneous assignments are made from the set of Relation choices.

This set of Boolean equations can be simplified to:

$$\sum_{K=1}^{p} \left(\prod_{i \in \bar{I}_K} \bar{g}_i \right) \prod_{\ell=1}^{L} \left(\left(\sum_{i=I_K} g_i \left(\prod_{j=1}^{q_i} b_{i_j}, \ell \right) \right) \overline{\oplus} b_{K\ell} \right) = 1$$

$$= \sum_{K=1}^{p} \left(\prod_{i \in \bar{I}_K} \bar{g}_i \right) \prod_{\ell=1}^{L} \left(\bar{b}_{K\ell} + \sum_{i \in I_K} g_i \prod_{j=1, i_j \neq K}^{q_i} b_{i_j}, \ell \right) = 1$$

(2) <u>Disjointness Constraint</u>:

We ensure that the final codes are orthogonal in L bits for N values:

$$\prod_{i=1}^{N-1} \prod_{j=i+1}^{N} \sum_{K=1}^{L} \left(b_{iK} \oplus b_{jK} \right) = 1$$

to build an instance of BCP from these constraints, we can construct a BDD which represents the set of constraints and find the path to 1 which has lost weight. We shall assign weight $(g_i) = 1$, weight $(b_{i_j}) = 0$ for this problem to produce minimal 2-level cover.

(next time: state encoding)

- Sideline: Where did those constraints come from?

→ In our earlier study of Encodings we used dominance relations among the codes to reduce cover cardinality – there is another way to reduce the size of these covers:

Consider the following machine:

1 0 1	out 1	1
1 0 0	out 2	1
1 1 1	out 3	1

This machine is output disjoint so that by our previous technique no reduction is possible. However, if we assign:

out 1	1 1
out 2	1 0
out 3	0 1

we can realize the function in 2 cubes ...

1 0 –	out 2	(1 0)	1
1 – 1	out 3	(0 1)	1

this is because the case for out 1 satisfies both cubes : resulting in 1 1 = out 1.

Our earlier techniques failed to make use of this possibility – so could not necessarily find optimal exact encodings.

- So ... consider the problem of constructing a 2-level cover for an encoding of a Boolean Function.

We can construct all Generalized Prime duplicants for this problem in direct analog to the construction for Boolean Relations we saw last time:

1. Merge of 2 adjacent symbolic output cubes by forming union of the cube part and intersecting the symbol part. (union of the symbols)

   Eq. if the output port of c_1 is (out 1, out 3) and of adjacent c_2 : (out 2, out 3)

   that of $\tilde{c} = c_1 \circ c_2 \Rightarrow$ (out 1, out 2, out 3)

2. Remove all cubes from n–1 generation who's output port is identical.

This leads to a list of cubes of successively larger size and smaller effectivity outputs. The removal process ensures that only prime implicants are kept. ?

Eq:

1011	a	101–	a,c
1000	b	10–0	b,c
0101	a	0–01	a,c
1010	c	010–	a,b
~~0110~~	~~b*~~	01–0	b
0001	c		
~~0100~~	~~b*~~		

(Note: 1 0 1 – is not adjacent to 1 0 – 0 → don't care symbols must agree for adjacency.)

Once we have built all these generalized prime implicants, we are free to construct a cover for the function. Note that even in this case, there are constraints which must be met. We could check for proper (non-cyclic) dominance and for correct disjunction, however, there is a simpler solution.

- We wish to formulate exact constraints on the primes so that the combination selected results in a realizable Boolean implementation (in 2-levels).

  Note that if we encode the symbols as Boolean vectors, we could simply check that for each minterm of f the 2-level result is identical to the selected encoding.

  Let $p_i(x)$ be the ith prime which covers x.
  Let $g_{i,x}$ be the Boolean Variable which selects is $p_i(x)$ appears in the cover.

Let $e(\sigma_x)$ be the encoding for the output symbol on minterm x.

Finally each prime $p_i(x)$ has several symbols in its output port so $p_i(x), j$ is the jth such symbol, and $e(p_i(x),j)$ is the encoding of the jth such symbol.

$\prod_j e(p_i(x),j)$ is the bit-wise intersection of all the encodings of all the symbols in the output port of the ith prime covering x.

$\sum_i g_{i,x}\left(\prod_j e(p_i(x),j)\right)$ is the bit-wise OR of all selected primes' intersected output encodings for all <u>selected</u> primes covering x ⇒ note this must be the same as the desired encoding of the output of x in the original cover.

$$\Rightarrow \forall x \sum_i g_{i,x}\left(\prod_j e(p_i(x),j)\right) = e(\sigma_x)$$

This simply ensures that any input minterm produces a valid output encoding.

- As before we can simplify this if we assume encoding of length L, $b_{i,\ell} = e(\sigma_i)_\ell$

We get: $$\forall x \prod_{\ell=1}^{L} \left(\left(\sum_i g_{i,x} \left(\prod_{j \neq \sigma_x} b_{i,x,j,\ell} \right) \right) b_{x,\ell} \right) = 1$$

<u>Ex</u>: for the initial cover:

1 0 1	a
1 1 0	b
1 1 1	c

we get:

g_0	1 0 1	a
g_1	1 1 0	b
g_2	1 1 1	c
g_3	1 - 1	a,c
g_4	1 1 -	b,c

the constraints are then: (L=2)

$$101 \rightarrow a \Rightarrow \left(\overline{b_{a,1}} + g_0 + g_3 b_{c,1}\right)\left(\overline{b_{a,2}} + g_0 + g_3 b_{c,2}\right)$$
$$110 \rightarrow b \Rightarrow \left(\overline{b_{b,1}} + g_1 + g_4 b_{c,1}\right)\left(\overline{b_{b,2}} + g_1 + g_4 b_{c,2}\right)$$
$$111 \rightarrow c \Rightarrow \left(\overline{b_{c,1}} + g_2 + g_3 b_{a,1} + g_4 b_{b,1}\right)\left(\overline{b_{c,2}} + g_2 + g_3 b_{a,2} + g_4 b_{b,2}\right)$$

plus, we must also insure that the b's are disjoint (different) output symbols.

Note that $g_0 = g_1 = g_2 = 0$, $g_3 = g_4 = 1$

requires that: $b_{c,1}$ and $b_{c,2} = 1$

finally we can let $b_{a,1} = 0$ and $b_{b,2} = 0$

this gives the solution:

$1-1, (a \cap c) = (01 \cap 11) = 01$

$11-, (b \cap c) = (10 \cap 11) = 10$

which is the <u>exact</u> minimal solution.

- Back to <u>Symbolic Relations</u>

For this problem we have the difficulty that there are now symbolic inputs as well as outputs and these inputs will cause a change in the previous prime generation alg.

It has been shown that for a pure symbolic input Boolean output problem, we can transform the symbolic input into a multiple-value input (Sasoo).

We will use a <u>hot-one</u> encoding for the symbolic input and then we can apply the previous prime generation algorithm as before with the caviat:

2 cubes can be merged only if:

1. identical input binary parts and different symbolic parts

2. distance – 1 binary parts and <u>identical</u> symbolic parts

Cancellation can occur <u>only</u> when:

the state, next state, and output parts are identical.

<u>Eq:</u>

0	S1	S1	1	–	100	(S1, S2)	0	note:	S1 = 100
1	S1	S2	0	0	110	(S1, S3)	0		S2 = 010
1	S2	S2	0	0	101	(S1, S3)	0		S3 = 001
0	S2	S3	0	1	110	(S2)	0		
1	S3	S3	1		. . .				
0	S3	S3	1						

Note: each prime with an input symbolic encoding which has more than 1 1 (eq: 0 <u>110</u> → (S1, S3) 0) implies a constraint on the input encoding since if this prime is selected, we must ensure that the smallest cube which covers *e*(S1) & *e*(S2) does <u>not</u> cover *e*(S3).

Each such cube prime introduces <u>face</u> constraints.

- In our previous discussion we performed the minimization of the cover, then determined if the cover was encodeable. i.e., A, B matrices were compatible.

Here, we again use the g_i variables to select the presence or absence of a prime in the cover. Thus for any well defined relation we can find an encoding for <u>some</u> selection of the primes.

For each prime with a merged symbolic input part (except an all-1's input port which is discarded) we add the following constraint to the encoding:

Let $g_1 \ldots g_n$	select the primes which produce a particular face constraint.
Let $\sigma_{t_1} \ldots \sigma_{t_T}$	be the set of states in the face and
$\sigma_{r_1} \ldots \sigma_{r_R}$	be the set of states which must <u>not</u> be in the face.

Then:

$$\prod_{i=1}^{n} \overline{g}_i + \prod_{k=1}^{R} \left(\sum_{i=1}^{L} \prod_{j=1}^{T} \left(b_{t_j i} \oplus b_{r_k i} \right) \right) = 1 \qquad \text{(for each constraint)}$$

Note that this is just an exhaustive check of all satisfying codes.

- <u>Joint State Minimization and State Encoding</u>

We can now <u>generalize</u> the state encoding problem by identifying pairs of equivalent states in a machine description and expanding the relation to include the new degrees of freedom.

1. Apply conventional state minimization to identify state equivalent and implied pairs.

   a. Build an implication graph $G(V, E)$ where each vertex v is an equivalent state pair $(\sigma_{v1}, \sigma_{v2})$ and each edge is a constraint where merging of $v_1 \rightarrow v_2$ implies the merging of v_2.

2. <u>Expand</u> the encoding problem to a relation in which any transition to a state could transition to any of that state's equivalent states.

3. Solve the symbolic relation problem in which we relax the requirement of unique state codes. i.e., Equivalent states need not have disjoint codes.

- This is done by modifying the Disjoint Constraints and adding new implied merging constraints.

1. $$\prod_{i=1}^{N-1} \prod_{j=1+1}^{N} \sum_{\substack{k=1, \\ \sigma_i \neq \sigma_j}}^{L} \left(b_{ik} \oplus b_{jk} \right) = 1$$

here, the only difference is that we don't check the codes that belong to both pairs of equivalent states.

If (σ_i, σ_j) is an equivalent state pair, and $\{(\sigma_{p_1}, \sigma_{q_1}), \ldots (\sigma_{p_I}, \sigma_{q_I})\}$ are implied pairs:

2. $$\prod_{r=1}^{I} \left(e(\sigma_i) \equiv e(\sigma_j)\right) \Rightarrow \left(e(\sigma_{p_r}) \equiv e(\sigma_{q_r})\right) = 1$$

$$\Rightarrow \prod_{r=1}^{I} \left(\sum_{K=1}^{L} \left(b_{ik} \oplus b_{jk}\right) + \prod_{k=1}^{L} \left(b_{p_r k} \overline{\oplus} b_{q_r k}\right) \right) = 1$$

Binate Covering Problem:

Let the Boolean Formula $T(X_1 \ . \ . \ . \ X_n)$ represent the covering constraints where an input vector **X** is satisfying iff $T(X) = 1$.

Let the cost of a positive literal x_i be given by w_i and the cost of a negative literal $\bar{x}_i$ is 0.

Find a minimum cost satisfying assignment **X**.

- Solution using BDD's:

def: the cost of a path in a BDD (D) is defined to be the sum of the cost of the arcs along the path where a '0' arc is free and a '1' arc costs w_v for variable v.

Th: For an ROBDD representing $T(x_1, \ldots x_n)$, the lowest cost (shortest) path connecting the root to a '1' mode is a minimum cost satisfiable assignment, regardless of the variable order.

Pff: Every path from '1' to the root represents a satisfiable assignment (minterm) of T. Each weight is added to the cost of the path iff that particular variable is 1 in the assignment.

We can find this shortest path weighted solution in time $O(V)$ on an ROBDD by simply applying Dijkstra's alg.

However: We must prime the BDD often as a problem with 100 primes (relatively small) will have over 10,000 variables! So BDD size containment is very important!

Example: Consider the state Machines below:

M1:				M2:			
	$0,a$	$\rightarrow$	$a,0$		$0\ a$	$\rightarrow$	$a,0$
	$1,a$	$\rightarrow$	$c,0$		$1\ a$	$\rightarrow$	$b,0$
	$0,b$	$\rightarrow$	$b,0$	$\Rightarrow$	$0\ b$	$\rightarrow$	$(a,b),0$
	$1,b$	$\rightarrow$	$b,-$		$1\ b$	$\rightarrow$	$a,1$
	$0,c$	$\rightarrow$	$b,0$				
	$1,c$	$\rightarrow$	$a,1$				

<u>minimized Machine</u>
representation: a = {a, b}
b = {b, c}

We can encode this machine's state with just 1 bit — but we can also see the appropriate constraints.

Converting $M2$ to primes leads to the following list:

g_1 :	0	a	a	0
g_2	1	a	b	0
g_3	0	b	a	0
g_4	0	b	b	0
g_5	1	b	a	1
g_6	0	aub	a	0
g_7	–	b	a	0

the output constraints are:

$0a$: $(\bar{b}_1 + g_1 + g_6)$
$1a$: $(\bar{b}_2 + g_2)$
$0b$: $(\bar{b}_1 + g_3 + g_6 + g_7)\bar{g}_4 + (\bar{b}_2 + g_4)\bar{g}_3\bar{g}_6\bar{g}_7$
$1b$: g_5 (essential cube)

disjoint coding constraint: $b_1 \oplus b_2 = 1$

<u>No</u> face constraints since either 1 or 2 possible merges, both timed.

note: that the $0b$ minterm and the recurrent $b_1 \oplus b_2 = 1$ are a tautology:

$$(b_1 \oplus b_2)\cdot\left((\bar{b}_1 + g_3 + g_6 + g_7)g_4 + (\bar{b}_2 + g_4)\bar{g}_3\bar{g}_6\bar{g}_7\right) = 1$$

if $b_1 = 1, b_2 = 0 \Rightarrow \bar{g}_4 + g_4 = 1$ (this is because we could
if $b_1 = 0, b_2 = 1 \Rightarrow g_4 + \bar{g}_4 = 1$ write $aub = -$
$\Rightarrow g_6 :\ 0,- \rightarrow a,0)$

So the final set of reduced constraints is:

$$(\bar{b}_1 + g_1 + g_6)(\bar{b}_2 + g_2)\dot{g}_5(b_1 + b_2)(\bar{b}_1 + \bar{b}_2)$$

there are 2 solutions with 2 terms:

$\{g_5, g_6, b_1\} = 1$
$\{g_2, g_5, b_2\} = 1$

which lead to: $1,0 \rightarrow 1,1$
$0,- \rightarrow 1,0$

or: $1,0 \rightarrow 1,0$
$1,1 \rightarrow 0,1$
$((0,-) \rightarrow 0,0)$*