SEQUENTIAL MACHINES

A sequential machine is represented by:

Q	a set of states	
I	a set of input symbols	
Z	a set of output symbols	
δ	a map: $\delta(I \times Q) \rightarrow Q$	(next state)
E	a map: $\in (I \times Q) \rightarrow Z$	(output)

Such a machine is a Mealy machine – if $\in$ is restricted to $\in (Q) \rightarrow Z \Rightarrow$ Moore machine.

* We assume the machine is deterministic $\Rightarrow$

E, δ is single-valued for all $I \times Q$.

Generally we assume that Q, I, Z are finite sets –

for some applications, we may have restrictions on the sequences of allowed inputs $\Rightarrow$

δ will have don't care outputs.

Alternatively the outputs of the machine may be ignored for certain sequences of inputs:

$\Rightarrow$ $\in$ has don't care or unspecified outputs.

We represent a state machine as a table (STT) with four columns:

a) the Primary inputs

b) the current state(s)

c) the next state

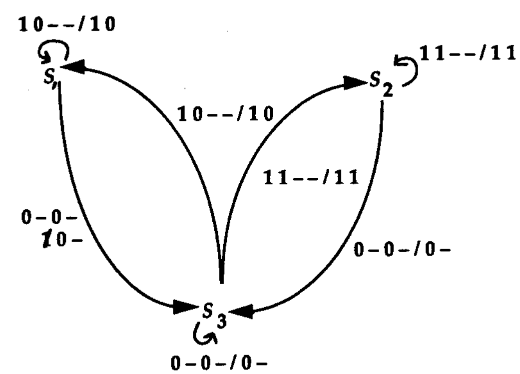
d) the Primary outputs

Conventionally, each row in the STT corresponds to a single transition of the machine. A slightly more compact form allows b) to list all of the state sharing c), d); valid for a). Note: We can further reduce the inputs by allowing don't cares to determine only the inputs which must be present for the transition.

Eq:

PI	PS(s)		NS	PO
10--	S_1,S_3	$\rightarrow$	S_1	10
11--	S_2,S_3	$\rightarrow$	S_2	11
0-0-	S_1,S_2,S_3	$\rightarrow$	S_3	0-

This technique allows a much larger space of valid machines.



A state machine defined in an ad/hoc manner may have a non-minimal # of states in its STT. In the case where the transition function δ and the output function ϵ are completely specified this reduction to a minimal machine is simple:

def: q_i and q_j are equivalent $q_i \equiv q_j$ iff
$\epsilon(J,q_i) = \epsilon(J,q_j)$ for all sequences $J \in I^*$.
i.e. two states are equivalent iff they cannot be distinguished by any set of inputs and outputs.

def: q_i and q_j are k-equivalent $q_i =_k q_j$ iff
$\epsilon(J,q_i) = \epsilon(J,q_j)$ for all sequences J of length k.

lem: both equivalence and k-equivalence are equivalence relations.

lem: if $q_i =_k q_j \Rightarrow q_i =_m q_j \ \forall_m \leq k$.
if $q_i \equiv q_j \Rightarrow q_i =_k q_j \ \forall_k$.

k-equivalence forms a partition π_k of the set of states S into equivalence classes for each k. Since $|\pi_k|$ can only increase with k, and since a partition class element is k-distinguishable from all elements outside its partition,

π_k is a refinement of π_r for $\forall_{k,r}\ k > r$.

π is a refinement of π_k for all k.

* If π contains partitions in which there is more than one state, each of those states are redundant since we could describe a machine with identical characteristics on input and output with exactly one state per partition.

Minimization: for each machine form the sequence $\pi_1, \pi_2, \pi_3 \ldots \pi_k \ldots$ a series of refinements of partitions of state. Consider the process at step k: each partition is checked for states whose successor belongs to a different partition than that which is the successor to the other states. (States with differing outputs are partitioned in π_1.) If this occurs, the partition must be split since the states are k-distinguishable. If no refinement occurs, then each partition transits to a unique partition on each input symbol so the π_k partitions are the π_{k+1} partitions. An identical procedure will show that $\pi_k = \pi_{k+1} = \pi_{k+2} \ldots$ for any number of trials. Thus π_k must $= \pi$. Finally, at each step at least one partition must be created until π_k is found $\Rightarrow$ process must terminate since there are only a finite # of states and each partition must contain at least one state.

(Note that in this case, this alg. is at worst $\theta(p \cdot r)$ where $p = |S|$ and r is the number of distinct transitions on S.)

def: two machines S, T are equivalent iff for each state $s_i \in S$ $\exists t_j \in T \ni s_i \equiv t_j$ and for each state $t_j \in T\ \exists\ s_i \in S \ni t_j \equiv s_i$.

def: two machines S, T are isomorphic if we can find a one to one mapping f such that: $f(q_s) \rightarrow q_T$

1. $\epsilon_s(i,q) = \epsilon_T(i, f(q))$

2. $f(\delta_s(i,q)) = S_T(i, f(q))$

for all $i \in I$, $q \in Q$.

<u>lemma</u>: two minimal, equivalent machines are isomorphic.

Note: two equivalent machines need not have the same number of states so may not be isomorphic.

We will often be concerned with machines which have no isolated states or proper submachines $\Rightarrow$ each state is recurrent. These machines have strongly connected STG's and are called strongly connected state machines.

Incompletely Specified Machines

(inclusion of don't cares)

We assume that the specified δ and $\in$ are subsets of the complete relations – –

In this case the previous state equivalence approach won't work since not all input sequences are <u>applicable</u>. We shall have to find a new relation: <u>compatible</u> which, unfortunately is <u>not</u> an equivalence relation.

<u>def</u>: $M_i(J)$ is the last-output function. $M_i(J)$ is the last output of the machine if initially in state i and is then subjected to the set of inputs J.

In an incompletely specified machine, we cannot ask if $w_i(J) = w_j(J)$ for all J since some states have no defined outputs and some transitions are undefined.

<u>def</u>: a sequence of inputs J is applicable iff

1) The sequence of states $q_1 = q$, $q_2 = \delta(i_1, q), q_k = \delta(i_k, q_{k-1})$ is well defined.

2) $\in (i_k, q_k) = M_i(J)$ is defined.

i.e. the output of the <u>last</u> transition is defined.

We can now partition I^* into those sequences which are applicable and those which are not.

<u>def</u>: Γ_i is the subset of I^* for which $M_i(J)$ is defined.

i.e. the applicable subset starting from state i.

$\tilde{\Gamma}_i$ is the complement set in I^*. $\tilde{\Gamma}_i \cap \Gamma_i = \phi$

(Note: if $\Gamma_i = \phi$ for some i, $\Rightarrow$ the state has <u>no</u> observable care set $\Rightarrow$ degenerate state. We can simply remove it W.L.O.G.)

<u>def</u>: <u>inclusion</u> for S,T two machines $q \in S$, $p \in T$ are states; $p \le q$ (q includes p) iff $\Gamma_p \le \Gamma_q$ and $M_p(J) = M_q(J)$ for $J \in \Gamma_p$.

Note: this is clearly <u>not</u> symmetric.

<u>def</u>: $S \le T$ for machines S and T iff each state $p \in T$ has a corresponding state $q \in S$ such that $q \le p$.

problem stmt.:

We wish to find a representative minimal state machine:

$$S \ni T \le S \text{ and for all } Y \ni T \le Y$$

the number of states of Y is greater than or equal to the number of states of S. Then S is a minimal state representative of the incompletely specified machine T. (Note that S need not be unique!)

def: Compatible States

q_a, q_b are two states of S with Γ_a, Γ_b being their applicable sets.

$q_a \sim q_b$ (compatible) iff $M_{q_a}(J) = M_{q_b}(J)$ for all $J \in \Gamma_a \cap \Gamma_b$.

although this is reflexive and symmetric – it is not transitive and so is not an equivalence relation.

Eq:

0	$q_1 \to q_1$	–
0	$q_2 \to q_3$	0
0	$q_3 \to q_2$	1
1	$q_1 \to q_2$	0
1	$q_2 q_3 \to q_1$	0

$q_1 \sim q_2$: ~ 1 on $\underline{1}$ $q_1 \to q_2$, $q_2 \to q_1$ output 0.
~ 2 on 01 $q_1 \to q_2$, $q_2 \to q_1$ output 0.
on 11 $q_1 \to q_2$, $q_2 \to q_1$ output 0.
~ 3 on 001 $q_1 \to q_2$, $q_2 \to q_1$ output 0.

$q_1 \sim q_3$: ~ 1 on $\underline{1}$ $q_1 \to q_2$, $q_3 \to q_1$ 0.
~ 2 on 01 $q_1 \to q_2$, $q_3 \to q_1$ 0.
on 11 $q_1 \to q_2$, $q_3 \to q_2$ 0.
~ 3 on 001 $q_1 \to q_2$, $q_3 \to q_2$ 0.

but $q_2 \not\sim q_3$ since ~ 1 $q_2 \to q_3$, $q_3 \to q_2$ but $\underline{0} \ne 1$.

def: we define $q_1 \sim k\ q_2$ (k–compatible) iff $M_{q_1}(J) = M_{q_2}(J)$ for all $J \in \Gamma_1 \cap \Gamma_2$ of length $\le k$.

def: a set of states is compatible (k–compatible) iff every pair is compatible (k–compatible)

def: a maximal–compatible set is a compatible set not contained in any larger compatible set.

For any machine S there is a set (class) of maximal compatible sets of states. As well, there is a set of all k-compatible maximal sets of states. These sets are not disjoint necessarily.

We denote $\zeta_k = \{B_1, \ldots B_u\}$ is a cover of the states of S and is the set of all maximal k-compatible sets of states for S.

This is an analog to the Boolean relation covering problems we have seen before. Since compatibility was not an equivalence relation, we get covers (not disjoint) instead of partitions.

Since a state which is incompatible with all others forms a maximal-compatible set of one element, the set of all maximal-compatible or maximal k-compatible sets form a cover for the states of S.

We can generate ζ systematically as follows:

1. Generate ζ_1 by grouping states with the same outputs for those input symbols applicable to both states (~1).

2. Check each set B_i (assumed to be k-compatible) for compatibility $k+1$.

   if the set is k+1 compatible $\Rightarrow$

   $\exists$ some $B_j \in \zeta_k \ni \delta(i,q) \in B_j$ for all $q \in B_i$ and all i which are applicable. $i \in I$.

   if the set is <u>not</u> k+1 compatible $\Rightarrow$

   Split Bj into (maximal) subsets each of which are k+1 compatible. (Note that two states will be in a common subset iff $\delta(i,q_1)$ and $\delta(i,q_2) \in B_j \leftarrow \underline{\zeta_k}$ for some j and all applicable i.)

3. Remove all non-maximal sets B_i from the generated set of k+1 compatible sets. This will form the new ζ_{k+1} set of maximal k+1 compatible states.

if $\zeta_{k+1} = \zeta_k \Rightarrow$ no pair states in cover can be observably different for any applicable set of inputs – $\zeta_k = \zeta$.

Example: Consider the machine:

0	q_1	q_6	0
0	q_2	q_5	1
0	q_3,q_4	q_2	–
0	q_5	q_1	1
0	q_6	–	–
1	q_1,q_3	q_5	1
1	q_2	q_6	1
1	q_4	–	–
1	q_5	q_4	–
1	q_6	q_4	1

ζ_1: 1:(q_1, q_3, q_4, q_6), 2:$(q_2, q_3, q_4, q_5, q_6)$

	q_1	q_3	q_4	q_6	q_2	q_3	q_4	q_5	q_6
$I=0$	1,2	2	2	–	2	2	2	1	–
$I=1$	2	2	–	1,2	1,2	2	–	1,2	1,2

$\leftarrow$ Blocks in ζ_1 on next state

ζ_2: all states in 1: transit to 2 $\Rightarrow$ 2-compatible $\Rightarrow$ 1: (q_1,q_3,q_4,q_6)
q_5 is not 2-compatible with q_2,q_3,q_4 on $I=0$.

ζ_2: 1:(q_1, q_3, q_4, q_6), 2:(q_2, q_3, q_4, q_6), 3:(q_5, q_6)

$I=0$	1,2,3	2	2	—	3	2	2	—	1	—
$I=1$	3	3	—	1,2	1,2,3	3	—	1,2	1,2	1,2

1:(q_1,q_3,q_4) 2:(q_4,q_6) (3,4) 4:(q_5,q_6)

3:$(q_2 q_6)$

ζ_3: 1:(1,3,4) 2:(4,6) 3:(2,6) 4:(5,6)

2,3,4	2 2	2	—	4	—	1	—
4	4 —	—	1,3	2,3,4	1,3	1,3	1,3

$\zeta_4 = \zeta_3 \Rightarrow \zeta_3 = \zeta$.

Once we have found ζ, we have already found all maximal sets of states which can be merged into single states of the (a) reduced representative.

def: preserved (closed) cover of S. a collection C of sets

$\{C_1, C_2, \ldots, C_r\}$ of states of S is a closed cover iff

1. It is a cover of the states of S
   i.e.,

$$Q = \bigcup_{i=1}^{r} C_i \ ; \ C_i \not\subseteq C_j \text{ for } j \neq i.$$

2. $\exists_k \ni \delta(i, C_j) \subseteq C_k$ for every $i \in I$
   (note: $\delta(i, C_j) = \phi \Rightarrow \subseteq$ all $C_k \ldots$)

where $\delta(i, C_j) = \{q \mid q = \delta(i,p) \text{ for all } p \in C_j \text{ where } \delta(i,p) \text{ exists}\}$

i.e., a closed cover is a cover where for each applicable input symbol, the image of δ for the cover element merges to a unique cover element.

Eq: For our previous example

$B_1 = 1,2,3 \quad B_2 = 4,6 \quad B_3 = 2,6 \quad B_4 = 5,6$

	$I=0$	$I=1$
B_1	B_3	B_4
B_2	B_3	B_1 or B_2
B_3	B_4	B_2
B_4	B_1	B_1 or B_2

So ζ we calculated was a closed cover.

Note: ζ is always a closed cover from the construction.

For any closed cover of states $C = \{B_1 \ldots B_r\}$ for a machine $S = (I, Q, Z, \delta, \in)$ we can define a new machine $S_c = (I_c, Q_c, Z_c, \delta_c, \in_c)$ where $S_c \geq S$ as follows:

1. $1_c = I,\ Z_c = Z$

2. $Q_c = \{b_1, \ldots b_r\}$

3. $S_c(i, b_j) = b_k$ if $\delta(i, B_j) \subseteq B_k$

4. $\in(i, b_j) = \in(i, q_j)$ for $q_j \in B_j$ if $\in(i, q_j)$ is defined for some $q_j \in B_j$.

   else <u>undefined</u>.

Note: $S_c \geq S$ since we constructed each B_j to be closed on C.

From this argument, it would appear that we are finished ... but we do not know $|C|$ for $|\zeta|$. In general $|\zeta|$ can be larger than $|Q|$ – we must find a closed cover of S with $|C|$ <u>minimal</u>. Worse, the elements of C need not be maximal compatible sets to be a member of a minimal cover, however we know:

1. Each $C_i \in C_{opt}$ is $C_i \subseteq B_j$ for at least one j.

2. Every state in S must occur in some C_i.

to form a legal closed cover, we must have B_i
$\ni\ \delta(i, B_j) \subseteq B_i$ for each B_j in the cover.

<u>Eq</u>: S

0	q_1	q_3	0
0	q_2	q_4	0
0	q_3, q_6	–	–
0	q_4	q_6	–
0	q_5	q_6	0
1	$q_1 q_6$	–	–
1	q_2	q_3	0
1	q_3	q_5	1
1	q_4	q_6	0
1	q_5	q_6	–

$\zeta = \{(q_1 q_4 q_5 q_6), (q_2, q_4, q_5, q_6), (q_1, q_3, q_5, q_6)\}$
$= \{B_1, B_2, B_3\}$

	$I=0$	$I=1$
B_1	B_3	B_1, B_2, B_3
B_2	B_1, B_2	B_3
B_3	B_3	B_1, B_2, B_3

Clearly $\zeta \Rightarrow S_\zeta$:

0	$b_1, b_3 \to b_3$	0
0	$b_2 \to b_2$	0
1	$b_1 \to b_1$	0
1	$b_2 \to b_3$	0
1	$b_3 \to b_3$	1

$|S| = 3$

However: B_2, B_3 cover all the states and:

	$I=0$	$I=1$
B_2	B_2	B_3
B_3	B_3	B_2, B_3

is closed so

$$\Rightarrow S_2: \begin{array}{cccc} 0 & b_2 & b_2 & 0 \\ 0 & b_3 & b_3 & 0 \\ 1 & b_2 & b_3 & 0 \\ 1 & b_3 & b_3 & 1 \end{array} \quad |S| = 2$$

Neither of these machines are equivalent but

$S_\zeta \geq S$, $S_2 \geq S$ so S_2 is a minimal representative.

State Assignment

The representation of <u>state</u> in an FSM is of prime importance in synthesis of suitable machines. Consider the two machines below:

<u>M</u>

0	A,B	A	0
0	C,D	C	0
1	A	D	1
1	B	C	0
1	C	B	0
1	D	A	1

α		y_1	y_2
A	$\rightarrow$	0	0
B	$\rightarrow$	0	1
C	$\rightarrow$	1	1
D	$\rightarrow$	1	0

β		y_1	y_2
A	$\rightarrow$	0	0
B	$\rightarrow$	0	1
C	$\rightarrow$	1	0
D	$\rightarrow$	1	1

$$\alpha: \quad \begin{aligned} y_1' &= \bar{x}y_1 + x\overline{y_1} && = f_1(x, y_1) \\ y_2' &= \bar{x}y_1 + xy_2 && = f_2(x, y_1, y_2) \\ z &= xy_2' && = f_3(x, y_2) \end{aligned}$$

$$\beta: \quad \begin{aligned} y_1' &= \bar{x}y_1 + x\overline{y_1} && = f_1(x, y_1) \\ y_2' &= x\bar{y}_2 && = f_2(x, y_2) \\ z &= x\overline{y_1}\overline{y_2} + xy_1y_2 && = f_3(x, y_1, y_2) \end{aligned}$$

By changing the encoding of state variables we can change the dependencies of the output functions on their basis variables. For completely specified machines, we can find state assignments which minimize dependencies.

By use of Partition theory:

1. Hartmonis and Stearns, "Algebraic Structure Theory of Sequential Machines," Prentice–Hall, 1966.

2. Karp, R.M., "Some Techniques of State Assignment for Synchronous Sequential Machines," IEEE Trans. Elec. Comp., v. EC-13, n. 5, pp. 507–518, Oct. 1964.

3. Kohavi, Z. and Smith, "Decomposition of Sequential Machines," Proc. Sixth Annual Symp. Switching Theory and Logical Design, Ann Arbor, MI, Oct. 1965.

4. Kohavi, Z., "Switching Theory and Finite Automatica," McGraw–Hill, 1970, 1975.

However, these techniques do not lead to systematic techniques for larger circuits. State Assignment can be viewed as either an encoding problem or as a partitioning problem on FSM's. Consider a state machine consisting of two separate submachines. The state encoding can be selected to allow separate decomposition of these variables or can be mixed to force joint decoding. In general, machines which do not have viable submachines can still be encoded as such, by making use of don't cares and by possibly augmenting the ensemble states of the machine. Such encoding reduces the logical complexity of the machine. Recently, logic synthesis has enabled automated construction of much larger machines and State Assignment has become more important and needs for good fast heuristics grows.

Encoding Techniques:

Jodi, Dolotta and McCluskey

Embedding Techniques:

Kiss, Cappiciono, NOVA

Assignment on Homing Distance:

Mustang, Lost Commitment.

We shall discuss PLA embedding first. (Kiss)

Synthesis of control using 2–level (PLA) logic

<u>Ref</u>: Giovanni de Micheli, CAD–5, no. 4, 1986
CAD–4, no. 3, 1985

Today we will examine problems in applying 2–level PLA structures to designs (which may be incompletely specified) of FSM's with simultaneous minimization of both the internal logic and the state encodings.

Control units arise in many VLSI designs (most of them!) Internal control helps the chip to exploit the speed of on–chip communication vs. the much slower off–chip connectibility. In particular we most often support:

- Communication Protocols
- Time–Sequenced Behavior (Controllers, Timers, etc.)
- Exceptions and data driven processing (eq. computers).

Control is often specified as a high level specification on activities of the process under control.

Eq: If ($\overline{\text{halt}}$ and $\overline{\text{Reset}}$) {
    If ($\overline{INT}$) {
        AR ← PC++;
    If (E x c ≠ Decode (AR)) {
            Execute();
            else
            Flush-pipe();
            } else
        Vect_interrupt(); }
    else decode (Reset/Halt cond)
                Executed();
}

However, this is often very far from the direct implementation of the description as a finite state machine since no concept of states (or even timing) is directly apparent. In addition, there is an <u>encoding</u> problem associated with the proper control of the bound function units and a <u>timing</u> problem associated with efficient implementation.

This level of design is called <u>Functional Design</u> of the control structure and comprises translation of the formal language level to a structure consisting of logic block and registers.

In most commercial designs this is done manually, even today.

Automated methods for control functional designs usually start with allocation of a data-path to perform the operations, scheduling of the operations on the D.P. and finally inferring the timing and signals needed to implement the schedule. This is done by equating states of the controller to microcode states of the data path and during the timing and encoding from simulation of the desired behavior on the data-path.

Note: The control unit timing affects the data-path as well-- one must design the two simultaneously for good results.

Several levels of optimation are possible:

Tradeoffs between control complexity and data-path size.
between speed of operations and size.
length of pipeline and complexity of control...

We will assume that this level of design has been done and we have a specification of the structure and states of the control system. We will assume that this data is tabular but symbolic in nature.

Several Problems Remain:

1. The derived specification of control is usually incompletely specified ⇒ state minimization. (NP-complete).

2. State assignments to Boolean values must be made.

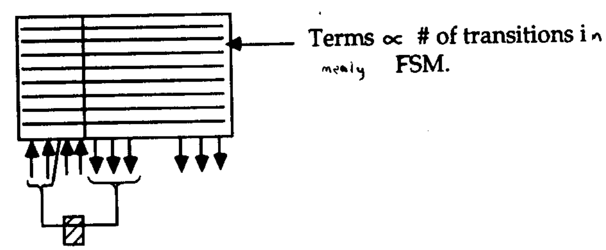
3. Efficient encodings of the states to control lines must be done.

State encoding is important for optimization of the control logic:

PLA area ∝ (# of terms) (# inputs and outputs)

The minimum # of terms is the # of terms in a minimal cover of the transition logic (dependent on the assignment).

The minimum # of columns is directly related to the encoding of the next-state.



* The particular notion of optimal state encoding depends <u>critically</u> on the implementation technology! i.e. FSM / Random Logic / Sequential machine

1. Find assignments of minimum code length among those which minimize the # of terms.

2. Find the assignments of minimal # of terms among those with minimal code length.

Symbolic Design:

We will simplify the Boolean problem of minimization by representing inputs and outputs symbolically then find the optimal switching function and finally encode the symbols in a minimal way.

This will allow us to solve:

1) Find optimal encoding of inputs.

2) Find optimal encoding of outputs.

3) Find optimal encoding of inputs and outputs such that some set of outputs and inputs have the same encoding.

Minimal here implies 2-level sum of products minimum.

Address mode	OP	Output
INDEX	AND	CNTA
INDEX	OR	CNTA
INDEX	JMP	CNTA
INDEX	ADD	CNTA
DIR	AND	CNTB
DIR	OR	CNTB
DIR	JMP	CNTC
DIR	ADD	CNTC
IND	AND	CNTB
IND	OR	CNTD
IND	JMP	CNTD
IND	ADD	CNTC

2 inputs and 1 output function of symbols onto symbols. Instead of assigning bits to the symbols first, we will examine what can be done symbolically.

Disjoint minimization: treat the outputs as separate tables:

INDEX	AND OR ADD JMP	CNTA
DIR	AND OR	CNTB
IND	AND	CNTB
IND	OR JMP	CNTD
DIR IND	ADD	CNTC
DIR	JMP	CNTC

Compaction by refinement. (Note cannot combine 2-3 since IND/OR ⇒ CNTD ≠ CNTB.

If we assign:

INDEX = 00	AND = 00	CNTA = 11
DIR = 01	OR = 01	CNTB = 01
IND = 11	ADD = 10	CNTC = 10
	JMP = 11	CNTD = 00

we get:

00	**	11
01	0*	01
11	00	01
11	*1	00
*1	10	10
01	11	10

for our table: note * ⇒ don't care.

We can delete line 4 as its output is 0.
⇒ 5 terms minimal covering.

If we assign instead:

CNTA = 00
CNTB = 01
CNTC = 10
CNTD = 11

we get:

(Pause for suspense)

00	**	00
01	0*	01
11	00	01
11	*1	11
*1	10	10
01	11	10

– again we can remove line 1: output = 0 but there we can also reduce the Boolean cover to:

*1	0*	01
*1	1*	10
11	*1	11

since covers 01, 10: *1 could be 11

(Note: 01 0* 01
11 00 01
1 0 01 eq:)

Leaving 3 terms instead of 5 which was minimal before.

Symbolically this is equivalent to the table:

INDEX	AND, OR, JMP, ADD	CTA
DIR, IND	AND, OR	CTB
DIR, IND	ADD, JMP	CTC
IND	JMP, OR	CTD

where we assume that CTD overrides CTB, CTC when both are specified. → This leads directly to the 3 term machine above.

Symbolic Minimization

def a symbolic variable s can take a single value from a finite set: S

$\phi \Rightarrow$ no value was taken by the variable.

if f: $S^i \rightarrow S^0$ is a completely specified function of the n input variables and m output variables if every input maps to some output.

a k-input function on S^i has domain: $S^i \equiv S_1^i \times S_2^i \times \cdots \times S_n^i$

similarly, for output for range: $S^0 \equiv S_1^0 \times S_2^0 \times \cdots \times S_n^0$

def if for some inputs, some outputs are unspecified $\Rightarrow$ don't care.

Eq: Ex. 4.1 is completely specified with $n = 2, m = 1$

$$S_1^i = \{DIR, IND, INDEX\} \quad S_2^i = \{AND, OR, ADD, JMP\}$$
$$S_1^0 = \{CNTA, CNTB, CNTC, CNTD\}$$

Boolean functions are symbolic functions on $\{0,1\}^n$.

Incompletely specified functions: $\{0,1,d\}^m$ form the range.

Operations on Boolean functions can be defined by order relations on a map from the Boolean rep. to the natural numbers:

Let $r: S \Rightarrow N$ {Natural Numbers}

Eq: AND: $s_1 \wedge s_2 \equiv r^{-1}(\min(r(s_1), r(s_2)))$
OR: $r^{-1}(\max(r(s_1), r(s_2)))$
NOT: $\sim s_1 \equiv r^{-1}(p-1-r(s_1))$

Symbols, however, have no apriori ordering relation, so we choose to relate them by orders of the words representing the symbols.

def a sum of products for a single valued symbol function is a sum of products of symbolic literal functions.

if S is the set of admissible values for $s \Rightarrow$ a symbolic literal is the non-empty subset $\sigma \subseteq S$. For any variable $s \in S$ we define

$$\ell(s,\sigma) \equiv \begin{cases} \text{TRUE if } s \in \sigma \\ \text{False else .} \end{cases}$$

Eq: $S_1^I = \{A, B, C, D\}$ if $\sigma = \{B, D\}$ $\quad \ell(S,\sigma) = \text{True}$ for $s = B$.

Note that $\ell(s,\sigma)$ is independent of the order of the S^I.

However, the order of S^0 is important and we relate the symbolic function to a partial order of relations among outputs. Order is important to implementation size.

Let $R \subseteq \{(s,s'): s,s' \in S^0\}$ be a partial order on S^0, then s covers s' if $s = s'$ or $s' = \phi$ or $(s,s') \in R^T$ where R^T is the transitive closure of R.

The symbolic sum of s and s' is defined only if s covers s' or s' covers s.

Eq: $$s \vee s' \equiv \begin{cases} s & \text{if } s \text{ covers } s' \\ s' & \text{if } s' \text{ covers } s \\ \text{ill defined} & \text{else.} \end{cases}$$

The symbolic product of s and s' is not defined, instead we define products of symbolic literals $\ell_i(s,\sigma)$.

def a symbolic product term of literals is the $n+1$ tuple $(\sigma_1,\ldots,\sigma_n,\tau)$ where $\sigma_i \subseteq s_i^I \quad i = 1\ldots n; \quad \tau \in S^0$ τ is called the output part.

def a symbolic product $p(s^I,\tau)$ of the literal fn :

$$\ell_i(s_i^I,\sigma_i) \text{ is } p(s^I,\tau) = \begin{cases} \tau & \text{if } \ell_i(s_i,\sigma_i) = \text{TRUE } \forall_i \\ \phi & \text{else .} \end{cases}$$

Eq: In the example above DIR AND CNTB is a product term since the fn takes the value CNTB when the inputs are DIR and AND.

Two products intersect $(P_1 \cap P_2 \neq \phi)$ if $\exists s^I \in S^I \ni P_1(S^I,\tau_1) \neq \phi$ and $P_2(s^I,\tau_2) \neq \phi$.

Two sets of products P_1 and P_2 intersect $(P_1 \cap P_2 \neq \phi)$ if $\exists p_1 \in P_1, \exists p_2 \in P_2 \ni p_1$ and p_2 intersect.

Two products are output-disjoint if either they do not intersect or they have the same output symbols.

i.e. $P_1(s^I,\tau_1) \cap P_2(s^I,\tau_2) \Rightarrow \tau_1 = \tau_2$

A set of products is output disjoint if each pair is disjoint.

Eq:	DIR, IND	ADD	CNTC
	DIR	ADD, JMP	CNTC

Intersect because for s^I = DIR, ADD the product functions have values. These functions are also output-disjoint since they have the same output part.

A symbolic function can be represented in sum of product form if $\forall s^I \in S^I$ the output sum is well-defined. Such a representation will always exist if

1. Any linear order on S^0 (since $\Rightarrow$ well ordered).

2. The representation is a sum of pair-wise output-disjoint products (since sum is only identical values).

We will write sum of product fn s as a table of product terms.

<u>def</u> <u>symbolic implicant</u> is a product $p(s^I, \tau) \ni \forall s^I \in S^I$ for which the function is specified: $f(s^I)$ covers $p(s^I, \tau)$

A symbolic cover of a function is a set of implicants $P = \{P_1, P_2 \ldots P_{(P)}\}$ whose sum is $f(S^I)$, $\forall s^I \in S^I$ for which the fn f is specified.

Note: The sum depends on the <u>order relation</u> among symbols S^0, we write the cover: $C(P, R)$, the cardinality of C is $|P|$.

A minimum symbolic cover is a cover of minimum cardinality.

A minimal symbolic cover is a cover ∍ no proper subset is a cover.

<u>Eq</u>: for the function defined in the example above:

a <u>disjoint</u> symbolic cover is:

INDEX	AND, OR, ADD, JMP	CNTA
DIR	AND, OR	CNTB
IND	AND	CNTB
IND	OR, JMP	CNTD
DIR, IND	ADD	CNTC
DIR	JMP	CNTC

⇒ this cover is compatible with <u>any</u> order relation R.

However:

INDEX	AND, OR, ADD, JMP	CNTA
DIR, IND	AND, OR	CNTB
DIR, IND	ADD, JMP	CNTC
IND	JMP, OR	CNTD

is a cover if $R = \{$(CNTD, CNTB); (CNTD, CNTC)$\}$ since the fourth term intersects with the second and third terms.

<u>Symbolic Minimization</u> (1-output)

As in Boolean minimizations, we are now to try to find a minimal <u>cover</u> for the symbolic representation. We expect that this problem is NP-complete.

Our plan is to iteratively reduce the cover cardinality by detecting and listing order relations (the set R) among the words, when these relations lead to smaller cardinality covers. i.e. we generate R to reduce the cardinality of P.

$C^0(P^0, R^0)$ is the initial cover, P^0 is output-disjoint; $R^0 = \phi$.

We will first describe the output = 1 term version to simplify the presentation.

R will be represented by a digraph $G(V, A)$ where $V = S^0$, $A \subseteq \{s_1 \in S^0, s_2 \in S^0 \neq s_1 : (s_1, s_2)\}$

Let $S^0 = \{s_i^0 : i = 1..q\}$ let $ON_i = \{p \in P^0 \ni \tau_p = s_i^0\}$. i.e. the set with a common output.

Note $ON_i \cap ON_j = \phi$ if $i \neq j$ since output disjoint.

The function to be covered is a collection of q multi-valued input, binary valued output functions whose on-set corr. to the points of the domain mapped into s_i^0, whose offset corresponds to those points mapped into s_j^0, $j \neq i$, and whose don't care set corresponds to the unspecified points.

We can solve the problem of finding a minimal symbolic representation of product terms by using multi-valued input, single (binary) output minimization techniques as in rect. cover reduction. We can do this q-times (once for each output symbol) to get a minimal disjoint rep.

→ But we can do better than this!

P need not be output disjoint as long as there is a cover relation between the non-disjoint words.

For example: suppose that $(s_j^0, s_i^0) \in R$ then any point in domain $\rightarrow ON_j$ can be used to reduce the cardinality of ON_i. $i \neq j$.

In the minimal symbolic representation, this point is still mapped into s_j^0 since $(s_j^0, s_i^0) \in R$. Thus we can augment the don't care set of ON_i by adding the care set of ON_j for $j \neq i$ and $(s_j^0, s_i^0) \in R$.

Eg: Let s_i^0 = CNTB and s_j^0 = CNTD, then ON_i is:

DIR	AND, OR	CNTB
IND	AND	CNTB

if (CNTD, CNTB) $\in R \Rightarrow DC_i$ includes:

IND	OR, JMP	CNTD

so the point in the domain s^I = (IND OR) can be used to reduce the cardinality of ON_i:

DIR, IND	AND, OR	CNTB

We need the don't_care set explicitly: we will do this by complementing the offset generalized to include cover relations:

the off_set $OFF_i = \bigcup_J ON_j$; $J = \{ j \ni \exists (v_i, v_j) \in G^+ \}$

i.e. there is a path from v_i to v_j in G or that v_i covers v_j.

i.e. the OFF_i set is the subset of s^I that is mapped by f into a value different than s_i^0 and which is covered by s_i^0.

At each iteration, we minimize M_i by minimizing ON_i using a routine that performs multi-valued input, binary output minimization.

We invoke minimize (ON_i, OFF_i) so that the don't care set $DC_i = \overline{ON_i} \cap \overline{OFF_i}$ is as large as possible.

$$\text{if } M_i \cap ON_j \Rightarrow \text{add } (s_j^0, s_i^0) \text{ to } R.$$

<u>Symbolic Minimization</u>:

```
DATA ONi  i = 1..q
DATA G(V, A) ; A = φ ; P = φ ;
     for (k = 1 to q)  {
          i = select (k) ;
          OFFi = Uj ONj ;   J = { j | ∃ a path for vi to vj in G}
          Mi = minimize (ONi, OFFi);
          A = A ∪ {(vj, vi) ∋ (Mi ∩ ONj) ≠ φ} ;
          P = P ∪ Mi ;
     }
```


Select is a heuristic ordering criterion for this routine ⇒ note that this 1-pass routine is essentially using greedy reduction.

The graph G produced above is <u>acyclic</u>.

<u>pff</u>: Initially G is empty. If at stage k, the graph is acyclic, then it will be acyclic at stage $i + 1$ since we only add edges: $(v_j, v_i)\{ j \in J \ni M_i \cap ON_j \neq \phi \}$. Since initially acyclic, any cycle must include one of these new edges. ⇒ a path must exist from v_i to some v_j. But by construction such a path would be in the OFF_i set and $ON_i \cap OFF_i = \phi$.

The $C(P, R)$ generated above is a cover (minimal) of $f = C^0(P^0, R^0)$ where $|P| \leq |P^0|$.

<u>pff</u>: Each s_i^0 must be represented since $s_i^0 \Rightarrow ON_i$ in P^0 is mapped into M_i. For any element s^I of the domain let $P(s^I) = \{ p \in P \ni p(s^I, \tau) \neq \phi \}$ since $C(P, R)$ is not necessarily output disjoint, the τ' of $P(s^I)$ can conflict, but if they do then $M_i \cap ON_j \Rightarrow (v_j, v_i) \in R$ so the sum of $P(s^I)$ given R is $f(s^I)$. Since M_i is minimal,

c is minimal.

Example:

in1	in2	out1
A	D	1
A	E	1
A	F	1
A	G	1
B	D	2
B	E	2
B	F	3
B	G	3
C	D	2
C	E	4
C	F	4
C	G	3

disjoint minimization: i.e. for each output symbol, find minimal cover.

A	D,E,F,G	1
B	D,E	2
C	D	2
B	F, G	3
C	G	3
C	E,F	4

$P^0 = \phi \quad A = \phi$

$ON_1 =$	A	D,E,F,G	1
$ON_2 =$	{ B	D,E	2
	C	D	2
$ON_3 =$	{ B	F,G	3
	C	G	3
$ON_4 =$	C	E,F	4

Select 1: $OFF_1 = \bigcup_J ON_j \;;\; J = \{\in G\} = \phi$

$$DC_i = \overline{ON_i} \cap \overline{OFF_i} = \overline{ON_i} = ON_2 \cup ON_3 \cup ON_4$$

find M_i = minimize (ON_i, OFF_i)
= A   D,E,F,G   1   Since null intersection with other terms.

$P = P \cup (A \;;\; D,E,F,G \;;\; 1)$
$M_i \cap ON_j = \phi$.

Select (2) $OFF_2 = \phi$
$DC_2 = \overline{ON}_2 = ON_1 \cup ON_3 \cup ON_4$
M_i = minimize (ON_2, OFF_2) = B, C   D, E   2 since we have C E, F → don't care.

$$M_i \cap ON_j = ON_4 \;\Rightarrow\; (s_4^0, s_2^0) \in R : (4,2) \in R$$

$P = P \cup (B, C \;;\; D, E \;;\; 2)$

Select (3) $OFF_3 = \phi$
$DC_3 = \overline{ON}_3 = ON_1 \cup ON_2 \cup ON_4$

M_i = minimize (ON_3, OFF_3) = B, C   F, G   3 since we have C ; E, F → don't care.

$$M_i \cap ON_j = ON_4 \;\Rightarrow\; (s_4^0, s_3^0) \in R \; (4,3) \in R$$

$P = P \cup (B, C \;;\; F, G \;;\; 3)$

Select (4) $OFF_4 = ON_2 \cup ON_3$

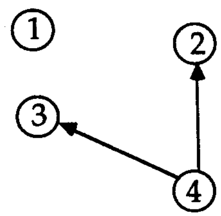
$$DC_4 = (ON_1 \cup ON_2 \cup ON_3) \cap (ON_1 \cup ON_4) = ON_1$$

$M_i = C, ; E, F$ (no intersections)

$M_i \cap ON_j = \phi$

P = A	DEFG	1
BC	DE	2
BC	FG	3
C	EF	4

R:



Boolean Encoding

1. We need to encode the input symbols so that each term in the symbolic cover is represented by a single implicant in the Boolean rep.

2. We need to encode the output symbols to preserve the cover relations R of the symbolic cover. i.e. we need the Boolean sop to map the outputs reps. so that the don't care covers are maintained.

Note: a Boolean implicant term is a cube or product of literals!

Note: we also need (for an *fsm*) to have a set of symbols whose encodings satisfy both 1 and 2 above. (state variables)

Formally:

let S be the set of symbols to encode, $n_s = |S|$.
let $n_p = |P|$ of the symbolic cover.
n_b = encoding length (i.e. # of bits)

We will need extended Boolean logic: $\{1, 0, \phi, *\}$
$* \Rightarrow$ don't care $\phi \rightarrow$ empty

AND	0	1	*	ϕ
0	0	ϕ	0	ϕ
1	ϕ	1	1	ϕ
*	0	1	*	ϕ
ϕ	ϕ	ϕ	ϕ	ϕ

OR	0	1	*	ϕ
0	0	*	*	0
1	*	1	*	1
*	*	*	*	*
ϕ	0	1	*	ϕ

def word-literal incidence matrix A: $A_{n_p, n_s} \in \{0, 1, *\}$

$$A = \begin{bmatrix} a_{1.} \\ a_{2.} \\ \cdots \\ a_{n_p.} \end{bmatrix} = [a_{.1} | a_{.2} \cdots | a_{.n_s}] = \text{where } a_{ij} = \begin{cases} 1 & \text{if word } j \text{ belongs to lit } i \\ * & \text{if word } j \text{ is don't care * in } i \\ 0 & \text{else.} \end{cases}$$

* don't care symbol in a literal is a symbol that may appear or not in a literal w/o affecting the representation.

Eq: let S be the set of op-codes: ={AND, OR, ADD, JMP}

for:

INDEX	AND,OR,ADD,JMP	CTA
DIR	AND,OR	CTB
IND	AND	CTB
IND	OR JMP	CTD
DIR IND	ADD	CTC
DIR	JMP	CTC

$$\Rightarrow A = \begin{array}{c} \text{AND} \; \text{OR} \; \text{ADD} \; \text{JMP} \\ \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 \end{bmatrix} \end{array} \begin{array}{l} \text{CTA} \\ \text{CTB} \\ \text{CTC} \\ \text{CTD} \end{array}$$

def $B \equiv$ partial order adjacency matrix $\in \{0,1\}^{n_s \times n_s}$ is the adjacency graph representation for the transitive closure of R. i.e. if word i covers word $j \Rightarrow b_{ij} = 1$. Since this is defined for the transitive closure: if $i \to j, j \to k \Rightarrow b_{i_k} = 1$ also. Although $b_{ii} = 0 \; \forall_i$.

Eq: for example with $R \neq \phi$: $B = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 \end{bmatrix}$

i.e. $d \to c, d \to b$

def E ≡ Encoding matrix $\in \{0,1\}^{n_s \times n_b}$ where each row is an encoding of the words of the set S.

def selection for $x, a \in \{0,1,*,\phi\}$. The selection of x according to a is:

$$a \cdot x = \begin{cases} x & \text{if } a = 1 \\ \phi & \text{else} \end{cases}$$

This can be generalized to matrix valued selections:

$$A \in \{0,1,*,\phi\}^{p \times q} \qquad X \in \{0,1,*,\phi\}^{q \times r}$$

$$A \cdot X = C = \{c_{ij}\}^{p \times r} \quad \text{where } c_{ij} = OR_{k=1}^{q} \; a_{ik} \cdot x_{kj}$$

$$= (a_{i1} \cdot x_{1j}) \text{ OR } (a_{i2} \cdot x_{2j}) \text{ OR } (\cdots) \text{ OR } (a_{iq} \cdot x_{qj})$$

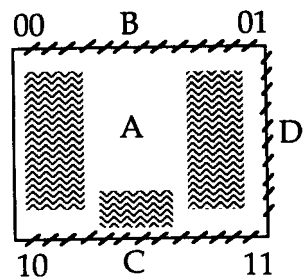
we will use selection to constrain the choices of the encodings to meet cover requirements.

def Face Matrix: $F \in \{0,1,*,\phi\}^{n_p \times n_b}$ each row f_n is a face of the n_b-dim Boolean hypercube corresponding to the face which encodes the symbolic literal.

$F = A \cdot E$ for an incidence matrix A and some encoding E. (Selection of E by A)

Eq:

$$\text{if } E = \begin{bmatrix} 00 \\ 01 \\ 10 \\ 11 \end{bmatrix} \quad F = A \cdot E = \begin{bmatrix} ** \\ 0* \\ 1* \\ *1 \end{bmatrix} \text{ for } A = \begin{bmatrix} 1111 \\ 1100 \\ 0011 \\ 0101 \end{bmatrix}$$



Each f_i row is the minimum subspace containing the state encodings for group i .

if we now form $\overline{A} = \{\overline{a}_{ij}\}$ $(\overline{a}_{ij} \equiv 1 \text{ iff } a_{ij} = 0 \text{ else} = 0)$ then $\overline{F}_i$

$\overline{F}^i \equiv \overline{a}_{ji} \cdot e_{ik}$ is a matrix whose rows are:

i) the encoding of word i if word i doesn't belong to literal j and is not a don't care.

ii) empty.

A given state encoding matrix E is a solution to the <u>input constrained</u> encoding problem and satisfies the input constraint relation A if:

$$\overline{F}^i \cap F \equiv \begin{bmatrix} \overline{f}^i_{1\cdot} \cap f_{1\cdot} \\ \overline{f}^i_{2\cdot} \cap f_{2\cdot} \\ \overline{f}^i_{n_s} \cap f_{n_s\cdot} \end{bmatrix} = \phi \forall_i \in 1..n_s$$

<u>Eq</u>: The previous example satisfied the constraint, but if we swap:

$$E = \begin{bmatrix} 01 \\ 00 \\ 10 \\ 11 \end{bmatrix}$$

then the word encoding for ADD intersects the 4th face

$$\overline{F}^3 \cap F = (\overline{a}_{\cdot 3} \cdot e_{3\cdot}) \cap (A \cdot E) = (\begin{bmatrix} 0 \\ 1 \\ 0 \\ 1 \end{bmatrix} \cdot [10]) \cap (\begin{bmatrix} 1111 \\ 1100 \\ 0011 \\ 0101 \end{bmatrix} \cdot \begin{bmatrix} 01 \\ 00 \\ 10 \\ 11 \end{bmatrix})$$

$$= \begin{bmatrix} \phi\phi \\ 10 \\ \phi\phi \\ 10 \end{bmatrix} \cap \begin{bmatrix} ** \\ 0* \\ 1* \\ ** \end{bmatrix} = \begin{bmatrix} \phi\phi \\ \phi 0 \\ \phi\phi \\ 10 \end{bmatrix}$$

Remark: We must avoid the violation of the above constraint as this constraint ensures that the input state literal can be uniquely decoded, i.e. no other state not in the group can match the cube. (Free). We wish to find the encoding of minimal length.

E1: Given an incidence matrix A, find E an encoding with minimal number of columns (= encoding length) $\ni \overline{F}^i \cap F = \phi \, \forall_i$.

Note: there is always an encoding E which satisfies the above relation: $E = I$ will work for any A. i.e. encode each symbol as a single Boolean literal.

Also: $E = A^T$ will satisfy A since $A \cdot A^T \Rightarrow f_{jj} = 1 \forall_j$
$\Rightarrow F \cap \overline{F}^i = \Phi$

Finally as in other state encoding systems, one can freely permute the columns and complement columns.

Output Variable Encoding

def: $A \in \{0,1\}^{p \times q}, X \in \{0,1\}^{q \times r}$ Boolean selection is:

$A \circ X = \{C_{ij}\}^{p \times r}$ where $c_{ij} \equiv \mathrm{OR}_{k=1}^{q} a_{ik}$ AND X_{kj}

AND, OR are the normal Boolean functions.

Let $G = B \circ E$. Row i of matrix G is the logical sum of the encoding of the words that must be covered by the encoding of the word i.
$\Rightarrow$ Output constraints are satisfied if E covers G or $\overline{E}$ AND $G = O$.
O is an all zero matrix.

Eq: $B = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 \end{bmatrix}$ for $E = \begin{bmatrix} 0 & 0 \\ 0 & 1 \\ 1 & 0 \\ 1 & 1 \end{bmatrix}$ $B \circ E = G = \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 1 & 1 \end{bmatrix}$

$\overline{E} \cap G = O$. But if $E = \begin{bmatrix} 0 & 0 \\ 1 & 1 \\ 1 & 0 \\ 0 & 1 \end{bmatrix} \Rightarrow B \circ E = G = \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 1 & 1 \end{bmatrix}$ $\overline{E} = \begin{bmatrix} 1 & 1 \\ 0 & 0 \\ 0 & 1 \\ 1 & 0 \end{bmatrix} \neq 0$

Since $3 \rightarrow 2$, $3 \rightarrow 1$ but $3 = 01$ and $1 = 11$. Boolean cover does not agree.

We can always find a solution to above for any B:

Note if $E = \overline{B}^T = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 1 \end{bmatrix}$ solves above trivially.

$B \circ \overline{B}^T \bullet = G = \mathrm{OR}(b_{ik} \cap \overline{b_{ik}}) = 0.$ So $G = 0$.

E2: Given B, find E of minimal columns $\ni$ $\overline{E} \cap (B \circ E) = 0$

E3: Given A, B find E of minimal columns $\ni$ Both input and output constraints are satisfied.

$\Rightarrow$ we can always solve E2:

but: E3 may not have a solution for arb. A, B.

<u>th</u>: Given A, B $\exists E \ni E$ satisfied input and output constraints

<u>iff</u>: $\forall_{r,s,t} \in S$ where $r \to s \to t$ $\exists_k \ni a_{kr} = 1,\ a_{ks} = 0,\ a_{kt} =$

<u>Row Based</u> Heuristic Alg:

Step 1: Select a symbol not yet encoded.
2: Find encoding of that word satisfying the constraint for those words selected so far.
3: If not possible, add column. $\to$ goto 2.

4: Choose best encoding from set generated in 2 and add to list $\to$ goto 1.

This works since:

1. We can often add a new encoding to the set by finding $\alpha \ni \left[\frac{E}{\alpha}\right]$ satisfies A' & B.

2. When we cannot find such an α there is always $T \ni \left[\frac{E|T}{\alpha}\right]$ will for a binary vector T. i.e., never need more than 1 column. (Why) (E2, E1)$\to$E3 may require $\underline{z}$.

   note: $\left[E|\alpha^T\right]$ satisfies $\left[\frac{A}{\alpha}\right]$...

However, this algorithm has poor performance for large examples.

We wish to satisfy both constraints at the same time — we will introduce a <u>partial</u> satisfyability measure on the <u>input</u> constraint relation. (The output relation can be satisfied for any bit width. . .).

i.e., E partially satisfies A if $\exists A' \le A \ni \overline{A'} \Rightarrow \overline{F'} \cap F' = \phi$.

We then count the number of elements A' and use this as a measure of performance.

Step 1: Select column vector $\{0,1\}^{n_s}$ that satisfies the output constraints B and has the largest satisfaction above.

Step 2: Append vector to E ; if all input constraints satisfied, stop.

Notes: if E satisfies the output relations $B \Leftrightarrow$ each column: if E also does. But this is not true for input relations: E may satisfy the input relations but some subsets of E may not. On the other hand, adding a column to E cannot decrease the set which is satisfied.

Column–based encoding

A, B inputs.

$FI,\ FO,\ n_b - \max;$

$FI \Rightarrow E1$ prob. $FO \Rightarrow E2$ prob. $FI \cap FO \Rightarrow E3$ prob.

$n_b = 0;$

```
if (FI)  A = clean (A)
if (FI)  A = compress (A);
if (FI∩FO)  A,B = verify constraints (A,B)
        do{
            e = column–select;
            if (n_b = 0)   E = e
            else  E = [E|e]
            n_b = column coordinality of E
            if (FI) A = reduce – constraints (A);
            } while (not all input constraints satisfied.)
```


where:

clean ( ) deletes duplicate rows of A and removes all rows of 0's or `1 1.

compress ( ) reorders the rows: don't cares to bottom and further reduces A by deleting rows which are products of rows of the reduced A'.

verify–constraint ( ) ensures that a solution can be found for $E3$ by releasing constraints.

release–constraints ( ) allows only the part of A necessary to be used: simplifying the constraints.

HCFB2562813