

Equivalency

Given same sequence input, if 2 states give the same final output \Rightarrow equivalent states

k-string: # of inputs in string
 form (ei 3string = 111 (a,b,c))

k-equivalent (using k-string)

No sequence of inputs of length k that can distinguish between the two states

k-distinguishing sequence

- If 2 states equivalent up to k; @ k+1 different \Rightarrow k+1 is distinguishing string // sequence
- at least 1 distinguishing sequence (ei k+1 string)

Binary relation

$\mathcal{E}^k(s,t)$
 $\mathcal{E}^k = \{(s,t) \mid s \equiv_k t\}$ $s \equiv_k s$
 equivalence relation - reflexive, symmetric & transitive
 $(s \equiv_k t) \Rightarrow (t \equiv_k s)$ $(s \equiv_k t) \wedge (t \equiv_k p) \Rightarrow (s \equiv_k p)$

equivalence class

- set of all equivalent elements
 B_i^k , k = length k of class (k-string)
 i = element #
 P^k , partition (just different notation than B_i^k)

example

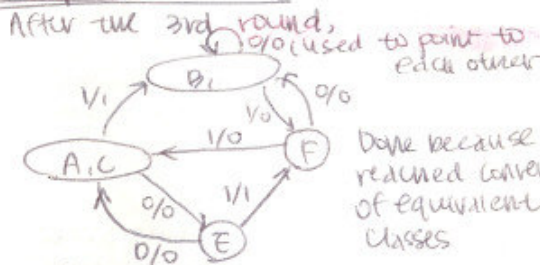


$B_1^1 = \{A, C, E\}$ // all have 0/0 & 1/1
 $B_2^1 = \{B, D, F\}$ // all have 0/0 & 1/0
 B_1^1 & B_2^1 are 1-equivalent
 A & B are 1-distinguished
 $\therefore 1$ is a distinguishing sequence

$B_1^2 = \{A, C, E\}$ // // 1/0, 00/00
 $B_2^2 = \{B, D\}$ // 00/00 & // 1/00
 $B_3^2 = \{F\}$

$B_1^3 = \{A, C\}$ // 001/001, 011/010, 111/100
 // 110/100
 $B_2^3 = \{B, D\}$
 $B_3^3 = \{F\}$
 $B_4^3 = \{E\}$

example (continued)



Algorithm for State Equivalence

using partition refinement
 - If all pairs are 1-equivalent, you don't need a sequence of inputs. It's a combinatorial logic & everything is a self loop. \therefore Algorithm does not need to be continued.

STATE-EQUIVALENCE (X, S, D, λ)

- $P^0 = \{B_i^0\}$
- $B_i^0 = S$ // $B_i^0 = \{A, B, C, D, E, F\}$
- $P^1 = P^0$ // $P^1 = P^0 = \{B_i^0\}$
- for $(x \in X)$ // $X = \{0, 1\}$
- for $(s_j \in S)$
- ① $O_j^x = \lambda(s_j, x)$ // $O^0 = (0, 0, 0, 0, 0, 0)$
- ② $P_j^x = \text{PARTITION}(O_j^x, S)$
- ③ $P^x = \text{REFINE}(P^x, P_j^x)$

COMMENTS

- ① for input 0, all outputs for every state is 0.
 $\therefore O^0 = (0, 0, 0, 0, 0, 0)$
- ② $P^0 = S$ since all outputs are the same & no distinguishing value
- ③ $P^1 \cap P^1 = P^1$
 $P^0 \cap P^0 = S$

- for $x=1$
- ① $O^1 = \{1, 0, 1, 0, 1, 0\}$
 $S = \{A, B, C, D, E, F\}$
- ② $B_1^1 = \{A, C, E\}$
 $B_2^1 = \{B, D, F\}$
 $P^1 = \{B_1^1, B_2^1\}$
- ③ $P^1 \cap P^1 = \{A, C, E, B, D, F\}$

Note: If you had obtained $P^1 \cap P^1 = \{A, B, C, D, E, F\}$, the algorithm stops since it is a combinatorial circuit