Synthesizing circuits from finite state machines

FSM = specification of the behavior of a circuit (the WHAT)
Circuit = implementation (the HOW)
Logic synthesis: derive the structure of a circuit (netlist) from its behavioral specification
   ideally **fully automatically, efficiently, and optimally**.

This requires several steps:
1. Specification (come up with the FSM)
2. State minimization
3. State encoding
4. Logic/timing optimizations

Rest of this lecture
State redundancy and equivalence

Consider the following FSM:

Are all states necessary?

States $s_1$ and $s_2$ are equivalent:
For any input sequence, the output sequence obtained starting from $s_1$ = the output sequence obtained starting from $s_2$.

State minimization

Remove redundancy by merging equivalent states
State minimization with the partition-refinement algorithm

General idea:
1. Start with considering all states potentially equivalent (i.e., create a partition of the state-space with a single equivalence class containing all states)
2. Iteratively refine the partition by splitting classes where not all states are equivalent, until no more splitting is needed.
3. Construct a new FSM whose states are the equivalence classes in the final partition.

When aren’t two states equivalent?

When there exists an input sequence which separates them, i.e., which generates different output sequences depending on which state we start from.

We call these separating input sequences.
When aren’t two states equivalent?

When there exists an input $x$, such that, either:
1. $s_1$ and $s_2$ give different outputs when fed with $x$, or
2. $s_1$ and $s_2$ lead to non-equivalent states when fed with $x$.

Example:

\[
\begin{array}{c|c|c}
\text{s1} & \text{s2} & \text{s3} \\
\hline
0/0 & 0/0 & 0/0 \\
1/1 & 0/0 & 1/0 \\
1/1 & 1/1 & 0/0 \\
0/0 & 0/0 & 1/1 \\
\end{array}
\]

The partition-refinement algorithm

Input: an FSM $M = (I, O, S, s_0, \delta, \lambda)$.
Output: a partition $P = \{S_1, ..., S_k\}$ of $S$ into $k$ equivalence classes.

1. Init: $P := \{S\}$
2. while $\exists A \in P$ s.t. $A$ is splittable {
   1. split $A$ into $\{A_1, ..., A_n\}$
   2. $P := P - \{A\} \cup \{A_1, ..., A_n\}$
   3. }

EECS 144/244, UC Berkeley: 8
The partition-refinement algorithm

When is a set $A$ splittable?

When either

$$\exists s_1, s_2 \in A : \exists x \in I : \lambda(s_1, x) \neq \lambda(s_2, x)$$

or

$$\exists s_1, s_2 \in A : \exists x \in I : \exists S_i \neq S_j : \delta(s_1, x) \in S_i \land \delta(s_2, x) \in S_j$$

The partition-refinement algorithm

How to split a set $A$?

Partition $A$ into minimal number of subsets $A_1, \ldots, A_n$ such that for every subset $A_i \subseteq A$, for every two states $s_1, s_2 \in A_i$, and for any input $x \in I$,

Case (a): $s_1, s_2$ generate identical output when given input $x$.

or

Case (b): $s_1, s_2$ lead to states in the same class $P_j$ when given input $x$. 
Illustration on our example

Input: an FSM $M = (I, O, S, s_0, \delta, \lambda)$.
Output: a partition $P = \{S_1, ..., S_k\}$ of $S$ into $k$ equivalence classes.

1. Init: $P := \{S\}$
2. while $\exists A \in P$ s.t. $A$ is splittable
   
   {  
   1. split $A$ into $\{A_1, ..., A_n\}$
   2. $P := P - \{A\} \cup \{A_1, ..., A_n\}$
   3. }

Complexity of the partition-refinement algorithm

Assume $|S| = n$ and $|I| = p$ (i.e., FSM has $n$ states, $p$ inputs).

At every round of splitting:
1. We iterate over all inputs
2. For each input, we iterate over all states
3. Which gives a complexity $O(pn)$ per splitting round

How many splitting rounds are there in the worst case?
$A: n - 1$, since there are $n$ states.
Generating separating input sequences

A concatenation of the input symbols during splitting provides a separating sequence for every two separated states.

How long can this sequence be in the worst case?

A: $n - 1$.

Generating the minimal FSM

From the resulting partition $P = \{S_1, ..., S_k\}$ construct a new FSM $M' = (I, O, P, S_{i_0}, \delta', \lambda')$ where:

- The set of states of $M'$ is $P$.
- The initial state $S_{i_0}$ is the class $S_i$ containing $s_0$.
- $\delta'$ maps a state $S_i$ to next state $S_j$ by input $x$, when all states in $S_i$ lead to some state in $S_j$ (we know that all or none will, since they are equivalent).
- $\lambda$ maps a state $S_i$ to output $z \in O$ by input $x$, when all states in $S_i$ that output (we know that all or none will, since they are equivalent).
References